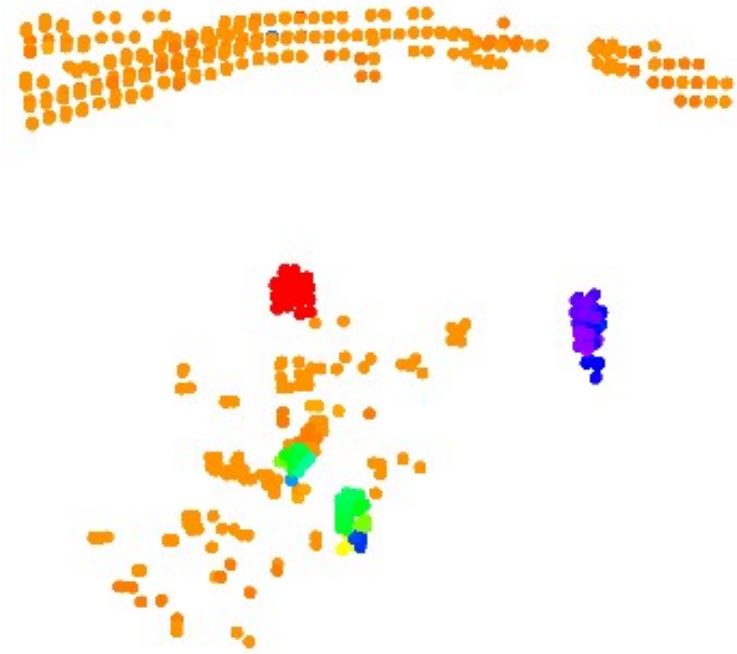


Semantic Segmentation and Classification on 4D-Imaging Radar Data

A Study of Neural Networks and Classical Methods on Radar Point Clouds

Master's thesis in Systems, Control and Mechatronics

Robert Füllemann & Yingjie Huang

Department of Electrical Engineering

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023
www.chalmers.se

Master's thesis 2023

Semantic Segmentation and Classification on 4D-Imaging Radar Data

A Study of Neural Networks and Classical Methods on Radar Point Clouds

Robert Füllemann
Yingjie Huang

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2023

Semantic Segmentation and Classification on 4D-Imaging Radar Data
A Study of Neural Networks and Classical Methods on Radar Point Clouds
Robert Füllemann & Yingjie Huang

Academic Supervisor: Lennart Svensson, Department of Electrical Engineering
Industrial Supervisor: Joakim Sörstedt, Sensrad AB
Examiner: Lennart Svensson, Department of Electrical Engineering

Master's Thesis 2023

Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Sensrad AB
Falkenbergsgatan 3
412 85 Gothenburg, Sweden
Telephone +46 704 34 27 32

Cover: Radar point cloud color-coded based on Doppler value, visible are four objects two bicyclists and two pedestrians.

Typeset in LaTeX
Gothenburg, Sweden 2023

Abstract

4D radar systems attract much attention nowadays, but little research has been done to explore the abilities of deep learning methods on perception tasks in 4D radar point clouds. Therefore, this master's thesis studies semantic segmentation and object classification capabilities of deep neural networks applied to 4D radar data of static applications and compares these networks against each other and classical, non-neural network, techniques. First, eight semantic segmentation neural networks are evaluated in accuracy and speed, a topic to the authors' knowledge not explored extensively before. Then the semantic segmentation task is separated into two subtasks: object masking and classification, both completed by PointNet. The performance of the new approach is compared against PointNet directly for the semantic segmentation task and non-neural network counterparts. As an extension, a sequential classifier is also introduced to handle the sparsity inherent in 4D radar data, and specialized metrics and corresponding loss functions aimed at pushing false positives toward ground truths in semantic segmentation are proposed. The results indicate differences in performances between point- and convolution-based networks. It is found that splitting the semantic segmentation task improves overall accuracy, and sequential data use further enhances classification performance. The investigated deep learning methods indicate higher accuracy than the investigated non-neural network techniques but are computationally heavier. This research provides meaningful insights into the applications of deep learning to 4D radar data, thus benefiting both academia and industry.

Keywords: Deep Learning, Semantic Segmentation, Object Classification, Point Cloud, 4D-Imaging Radar

Acknowledgements

Firstly, we want to thank Sensrad AB for providing a thesis opportunity and our industrial supervisors for initiating such an amazing thesis topic. In particular, we appreciate Joakim for his guidance and help throughout the whole project. Also, we thank Jacob and Tony for sharing radar basic knowledge and supporting us to integrate our neural networks into the perception pipeline of Sensrad AB. Lastly, we would like to thank our academic supervisor and examiner, Lennart, for early navigating us in the right direction and continuously giving us feedback on further improvements. Without any one of you, this thesis project would not be made successful.

Robert Füllemann & Yingjie Huang, Gothenburg, June 2023

List of Acronyms

AEDC	Average Euclidean Distance to the closest Corresponding ground truth
AEDO	Average Euclidean Distance to the closest Object ground truth
bbox	bounding box
CFAR	Constant False Alarm Rate
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DBSAN	Density-Based Spatial Clustering of Applications with Noise
DDM	Doppler-Division Multiplexing
FC	Fully Connected
FFT	Fast-Fourier-Transform
FIFO	First In First Out
FLOPs	FLoating-point Operations
FMCW	Frequency-Modulated Continuous-Wave
FNN	Feed-forward Neural Network
FOV	Field Of View
FPS	Farthest Point Sampling
fps	frames per second
GNN	Global Nearest Neighbor
GPU	Graphics Processing Unit
IF	Intermediate Frequency
IoU	Intersection over Union
KDE	Kernel Density Estimate
LiDAR	Light Detection And Ranging
LSTM	Long Short-Term Memory
mAEDC	mean Average Euclidean Distance to the closest Corresponding ground truth
mAEDO	mean Average Euclidean Distance to the closest Object ground truth

MAP	Maximum A Posteriori
MIMO	Multiple-Input Multiple-Output
mIoU	mean Intersection over Union
MLP	Multi-Layer Perceptron
mmWave	millimeter Wave
MOT	Multi-Object Tracking
NBC	Naive Bayesian Classifier
NLL	Negative Log-Likelihood
NN	Neural Network
Radar	RAdio Detecting And Ranging
RNN	Recurrent Neural Network
SemSeg	Semantic Segmentation
SIMO	Single-Input Multiple-Output
SOTA	State-of-the-Art
TDM	Time-Division Multiplexing
TSN	Two-Stage Network

Contents

List of Acronyms **ix**

List of Figures **xiii**

List of Tables **xvii**

1 Introduction **1**
1.1 Background and Objective . . . 2
1.2 Contributions . . . 2
1.3 Thesis Outline . . . 3

2 Data **5**
2.1 4D-Imaging Radar . . . 5
2.2 Dataset . . . 8
2.2.1 Annotation . . . 10
2.2.2 SemSeg Dataset . . . 10
2.2.3 Classification Datasets . . . 12
2.2.4 Sequential Classification Datasets . . . 13
2.2.5 Analysis . . . 13

3 Theory **17**
3.1 Perception Tasks for Point Clouds . . . 17
3.2 Common Structures of Neural Networks . . . 18
3.3 Neural Networks for Point Clouds . . . 21
3.3.1 Point-based . . . 22
3.3.2 Discretization-based . . . 23
3.3.3 Combinations . . . 24
3.4 Naive Bayesian Classifier . . . 24
3.5 Doppler Masking . . . 25
3.6 DBSCAN . . . 26
3.7 Evaluation metrics . . . 26

4 Comparison of Different Networks **29**
4.1 Accuracy Evaluation . . . 29
4.2 Resource Evaluation . . . 32
4.3 Performance Analysis . . . 35
4.4 Discussion . . . 36

5 Two-Stage Network **37**
5.1 Emergence of Task Decoupling: Addressing Classification Challenges 37
5.2 Comparison: PointNet vs Two-Stage Network 38
5.3 Comparison: Classical vs Neural Network Methods 41
5.3.1 Doppler masking vs PointNet masking 41
5.3.2 Naive Bayesian classifier vs PointNet classifier 43
5.4 Object Classification: Performance Analysis 44
5.5 Discussion 46

6 Sequential Classifier **47**
6.1 Sequential Architectures 47
6.2 Accuracy Evaluation 50
6.3 Resource Evaluation 54
6.4 Performance Analysis 56
6.5 Discussion 61

7 Metric and Loss Function **63**
7.1 Defect of IoU/mIoU 63
7.2 New Metrics 63
7.2.1 Average Euclidean Distance to the Closest Corresponding Ground Truth (AEDC) 64
7.2.2 Average Euclidean Distance to the Closest Object Ground Truth (AEDO) 65
7.3 New Loss Functions 65
7.4 Evaluation 66
7.5 Discussion 67

8 Conclusion **69**

A Appendix 1 **I**
A.1 Technical Specification of the Used 4D Radar I

B Appendix 2 **III**
B.1 Training Conditions for SemSeg Networks III

C Appendix 3 **V**
C.1 Training Conditions for TSNs V
C.1.1 Data Augmentation V
C.1.2 Normalization V
C.1.3 Equal Sampling vs Class Weights VI
C.1.4 Comparison: cluster dataset vs sequential cluster dataset . . . VI

D Appendix 4 **VII**
D.1 Training Conditions for Sequential Classifiers VII
D.2 Parameters for Different Heads VIII
D.3 Noise Clusters in Sequential Classifiers VIII
D.4 Feature Spaces in Sequential Classifiers IX

List of Figures

2.1 An example of point cloud generated by a 4D radar system from Sensrad AB with the corresponding camera image. 6
2.2 An illustration of slow-fast-time matrix on the left and Range-Doppler-map on the right. 7
2.3 An illustration of signal separation in MIMO radar. 7
2.4 Example images of the three available static scenarios. 8
2.5 Point-wise class distribution of the selected frames. 9
2.6 Flowchart of different dataset variants. 10
2.7 Point-wise statistics of SemSeg dataset v1. 11
2.8 Histogram of the number of points in each frame in SemSeg Dataset v1, where the frames with more than 4096 points are marked in red. . 11
2.9 Cluster-wise statistics of classification dataset v1. 12
2.10 Track-wise statistics of sequential classification dataset v1. 13
2.11 Kernel Density Estimate (KDE) over the classes with pedestrians in red, bicyclists in green, and vehicles in blue. 15
2.12 The scatter plot illustrates the combination of features leads to new feature representations. 16

3.1 Calculation scheme and neural structure of a fully connected layer. . 19
3.2 Calculation scheme and neural structure of a convolutional layer. . . . 20
3.3 LSTM diagram [1] and corresponding equations. 20
3.4 PointNet architecture for classification and SemSeg [2]. 23
3.5 An illustration of classical DBSCAN from [3] with core points p_i in red, non-core points $\bar{p}_i$ in yellow, outliers $\dot{p}_i$ in blue, circles of radius ϵ, and the minimum neighbour points $n = 3$. 26
3.6 An example of IoU calculation, where pink and black points represent pedestrian and environment respectively. 27

4.1 Training and validation performance of different SemSeg networks. . . 30
4.2 Confusion matrices of different SemSeg networks with classes of environment (0), pedestrians (1), bicyclists (2), and vehicles (3). 31
4.3 Inference speed of different SemSeg networks with different numbers of input points on GPU (NVIDIA RTX 3090). 34

4.4 Performance overview of each neural network, where the x-axis represents the inference speed in fps, the y-axis represents the converged validation mIoU, the size of each bubble represents the model size, and the transparency of each bubble represents the training time with the fastest network corresponding to 100% transparent, the color represents the network type with point-based in green, discretization-based in blue, and combinations in orange. 35

5.1 Two-Stage Network in blue with comparisons in white. 37
5.2 A comparison of IoU for objects between models trained on 4 classes in orange and models trained exclusively on binary classes in blue. . . 38
5.3 A comparison of SemSeg performance between PointNet and TSN on the same validation dataset. 40
5.4 A comparison of masking performance between classical and deep learning approach on the same validation dataset. 42
5.5 A comparison of classification performance between classical and deep learning approach on the same validation dataset. 44
5.6 A comparison of classification performance between (a) PointNet in TSN pipeline given v2 data (marked in yellow) and (b) PoinNet classifier given v1 data. 45

6.1 PointNet encoder architecture. 48
6.2 CNN head architecture. 49
6.3 FNN head architecture. 49
6.4 LSTM head architecture. 50
6.5 Training and validation performance of different classifiers. 51
6.6 Normalized confusion matrix in the first row and Precision, recall, and F1 score per class in the second row for single frame PointNet classifier and the sequential CNN-, FNN-, and LSTM-based approaches. 53
6.7 Prediction distribution over 52 test tracks for each classifier, where the x-axis represents different tracks, the y-axis represents the stacked predictions for every track with correct predictions marked in blue and wrong marked in orange. 57
6.8 Predicted class probabilities for the track of 50 pedestrian samples followed by 100 bicyclist samples with the red vertical line indicating the switching point. 58
6.9 Predicted class probabilities for the track of 50 pedestrian samples followed by 100 noise samples with the red vertical line indicating the switching point. 59
6.10 An illustration of internal workings of LSTM, where the first row represents the input data (feature space), the second and third rows show the internal states, and the last row displays the model output. 61

7.1 An example of ground truth and different predictions of the same point cloud, where pink represents pedestrian and black represents environment. 64
7.2 Validation mAEDC, mAEDO, and mIoU of three PointNet variants. . 67

C.1 F1-score of PointNet trained on cluster dataset and sequential cluster dataset on the sequential cluster validation dataset. VI

D.1 Sequence of 150 clusters, encoder size 1024 of original PointNet. . . . IX
D.2 Sequence of 150 clusters, encoder size 512 of FNN. X
D.3 Sequence of 150 clusters, encoder size 512 of CNN. X
D.4 Sequence of 150 clusters, encoder size 512 of LSTM. XI

List of Tables

2.1 Average length of sequential classification dataset v1. 13

4.1 Converged validation and training mIoU of different SemSeg networks as well as the difference between them in descending order of validation mIoU. 30
4.2 Number of parameters of different SemSeg networks in ascending order. 32
4.3 Inference speed of different SemSeg networks on GPU (NVIDIA RTX 3090) in descending order. 33
4.4 Training time of different SemSeg networks on GPU (NVIDIA GeForce GTX TITAN X) in ascending order. 34

5.1 Inference speed of PointNet (SemSeg) and TSN on GPU (NVIDIA RTX 3090). 40
5.2 Inference speed of Doppler masking on CPU (AMD Ryzen 9 5950X) and PointNet masking on GPU (NVIDIA RTX 3090) in descending order. 42
5.3 Inference speed of Naive Bayesian classifier on CPU (AMD Ryzen 9 5950X) and PointNet classifier on GPU (NVIDIA RTX 3090) in descending order. 44

6.1 Number of parameters of different modules in a sequential classifier in ascending order. 54
6.2 Number of parameters of different classifiers in ascending order. . . . 54
6.3 FLOPs and inference speed of different modules in a sequential classifier on GPU (NVIDIA RTX 3090) in descending order of inference speed. 55
6.4 FLOPs and inference speed of different classifiers on GPU (NVIDIA RTX 3090) in descending order of inference speed. 55
6.5 Training time for different classifiers on GPU (NVIDIA RTX 3090) in ascending order. 55

A.1 Specification of the used 4D-imaging radar. I

B.1 Batch size of each SemSeg network. IV

C.1 Training Conditions for the networks. V

D.1 Training Conditions for sequential classifiers. VII

D.2 Parameters for the CNN head architecture. VIII
D.3 Parameters for the FNN head architecture. VIII
D.4 Parameters for the LSTM head architecture. VIII

1
Introduction

The study of perception is a critical aspect of understanding how both humans and machines interact with their environment. In recent years, the development of new sensing technologies has led to significant advancements in various fields, such as robotics [4], automotive [5], maritime [6], healthcare [7], and surveillance [8]. One such technology is the 4D radar system, which has garnered attention due to its unique strengths and potential for a wide range of applications, such as robustness in adverse weather. This master's thesis aims to explore the capabilities and limitations of the 4D radar system, focusing on the extraction of valuable information from point cloud data using both classical and deep learning methods.

4D radar systems are sensors that provide three-dimensional spatial as well as speed information by using high-frequency radio waves, a technology with potential in several sectors. Its inherent weather robustness, a beam range of up to 300 meters, the inclusion of the Doppler feature, and the rather low production cost make it a promising candidate for various static and dynamic applications. However, as a relatively early-stage technology, there is a significant gap in the existing literature and research for exclusively designed neural architectures for 4D radar systems.

The first datasets for 4D radar systems have been released in 2019 [9] and onward [10][11][12], which provided a foundation for researchers to explore different detection and classification methods within this area. Not only the datasets but also some neural networks (NNs) have been created to solve different tasks on 4D radar point clouds. E.g., a transformer [13] that is designed to address the object classification problem on 4D radar point clouds came out in 2021, and a self-supervised learning NN [14] for scene flow estimation on 4D radar point clouds has been published in 2022. However, there are still many untouched tasks on 4D radar point clouds such as an in-depth study of the fundamental task of semantic segmentation (SemSeg) and a general evaluation of NNs initially designed for other point cloud domains applied to 4D radar point clouds. Thus, this thesis will delve into the process of extracting crucial information from 4D radar point cloud data, as well as the comparative analysis of classical and deep learning approaches for detecting and classifying objects on this data.

Through an in-depth study of the 4D radar system, this thesis aims to contribute to the growing body of knowledge surrounding this technology, while also identifying areas for future research and development. Specifically, SemSeg and classification tasks are of interest. By doing so, it is expected that this work will provide valuable

insights for both academia and industry, fostering the continued growth and evolution of 4D radar systems and their applications.

1.1 Background and Objective

Sensrad AB, a startup company originating from Qamcom Research And Technology AB, is currently developing a 4D-imaging radar system. The perception pipeline developed by the company relies on non-NN methods, which are referred to as classical methods in the rest of this report, for detecting, tracking, and classifying objects. Therefore, the company initiated this thesis project to investigate the potential of deep learning techniques.

To this end, Sensrad AB has created its own dataset for deep learning training and performance evaluation using its state-of-the-art (SOTA) radar system. The process of labeling the dataset involved projecting point cloud data onto the corresponding labeled camera images, resulting in point-wise labeled point clouds. The task at hand is to examine the detection and classification capabilities of neural and non-neural algorithms given the dataset.

Given the objectives outlined above, this master's thesis aims to:

1. Investigate deep learning approaches for detecting and classifying objects in static 4D radar point cloud data.
2. Identify the challenges and potentials of the deep learning approaches.
3. Address encountered challenges and improve the networks.

With these aims, this thesis aims to facilitate the progression of more accurate and efficient perception in 4D radar systems. Moreover, it seeks to provide meaningful insights not only for Sensrad AB but also for the wider research community.

1.2 Contributions

The contributions of our thesis are twofold: first, it involves implementing and evaluating established methodologies for SemSeg task to a rather novel context, i.e., 4D-imaging radar data; and second, devising approaches to address the issues we encountered.

Implementing established methodologies to a novel context:

1. The deployment of eight distinct NNs, each tested, assessed, and compared against one another.
2. The comparative evaluation of classification task between classic and NN classifiers, i.e., Naive Bayesian Classifier (NBC) and PointNet classifier.
3. The assessment of a conventional segmentation algorithm, Doppler masking, versus the NN approach using PointNet SemSeg.

Devising innovative techniques to address the encountered problems:

1. The development and evaluation of our Two-Stage Network (TSN), designed to handle object detection and classification independently, to enhance the classification performance.
2. The development and evaluation of our sequential classifier, the natural continuation of the TSN, aimed at further raising the classification capabilities.
3. The formulation of a novel metric and loss function to address wrongly classified background points.

1.3 Thesis Outline

Chapter 2 first explains some basic knowledge about 4D-imaging radars, then introduces the dataset and how it is annotated. Lastly, each dataset variant is explained in detail, followed by an analysis of the data.

Chapter 3 explains the theories and technologies used in this project. More specifically, three perception tasks and some general NN structures are included. Thereafter, eight SemSeg models are briefly explained, followed by some conventional techniques which are currently used in the perception pipeline of Sensrad AB. In the end, evaluation metrics for different perception tasks are also involved.

Chapter 4 presents a comparison of eight SemSeg networks in terms of accuracy and resources (model size, inference speed, and training time). Then a performance overview of all networks is illustrated in a bubble plot. A discussion of the obtained results is finally provided.

Chapter 5 is all about the conceptual framework: Two-Stage Network (TSN). We first elaborate on how this idea comes out, and naturally, compare the performance of PointNet with a TSN exclusively using PointNet. Another comparison between the components of a TSN and corresponding classical methods is also made. Finally, we analyze the classification results before discussing the whole chapter.

Chapter 6 is a continuation of Chapter 5 to enhance classification capability by leveraging on consecutive frames. We first explain the architecture of the sequential network, where three different heads are also introduced and compared against each other as well as the PointNet classifier regarding accuracy and resources. Again, we made a comprehensive analysis with three special scenarios, and close this chapter by discussing the findings.

Chapter 7 first demonstrates the defect of Intersection-over-Union (IoU), thus providing two new metrics to compensate for this defect. Consequently, two corresponding loss functions are also invented and evaluated on PointNet. We finally discuss the evaluation results of the new metrics and mean IoU (mIoU).

Chapter 8 is the last chapter where we summarize the whole project. We conclude the findings from every chapter in order, followed by a proposal for future work.

2

Data

The process of dataset creation at Sensrad AB is an ongoing one, and during the course of our thesis, the dataset is continuously improved and extended. Of this data, only a subset has been labeled by an external labeling company, leading to the creation of three separate datasets: a, b, and c. In this chapter, we delve into the details of the data, starting with a brief introduction to the background of the data creation, the signal processing of 4D-imaging radar systems, according to source [15], and internal company knowledge. Then the labeling process is discussed and a general analysis of the datasets is conducted. The aim is to provide a comprehensive understanding of the data used in our research, its quality, and any inherent biases or limitations that may affect the results of our study. This understanding is crucial in ensuring that the data and the derived results are appropriately interpreted.

2.1 4D-Imaging Radar

4D-imaging radar is an advanced type of mmWave radar system that goes beyond traditional radar capabilities. It generates spatial point clouds, which include range, elevation, and azimuth, along with Doppler velocity (range rate) and received power for each point. This provides a comprehensive understanding of the environment, as demonstrated in Fig. 2.1. Refer to Appendix A for technical details on the 4D radar using processing units from Arbe Robotics Ltd.

(a) Point cloud colored by Doppler.

(b) Point cloud colored by power.

(c) Corresponding image.

Figure 2.1: An example of point cloud generated by a 4D radar system from Sensrad AB with the corresponding camera image.

Those 4D point clouds are generated using Frequency-Modulated Continuous-Wave (FMCW), in combination with several signal processing techniques. The FMCW signal consists of 'chirps', which are signals that linearly increase in frequency over time. These signals are reflected by objects and returned to the radar, where the received signals are mixed with the transmitted signals to create the intermediate frequency (IF) signal. The IF signal is then converted into a digital form and arranged in a two-dimensional matrix with each column representing a chirp.

To calculate the distance and speed of surrounding objects, invariant object distance, and no Doppler frequency shift are assumed. Utilizing the Fast-Fourier-Transform (FFT) leads to a Range-Doppler map as Fig. 2.2 shows, which provides a comprehensive representation of the distance and speed of objects around the sensor.

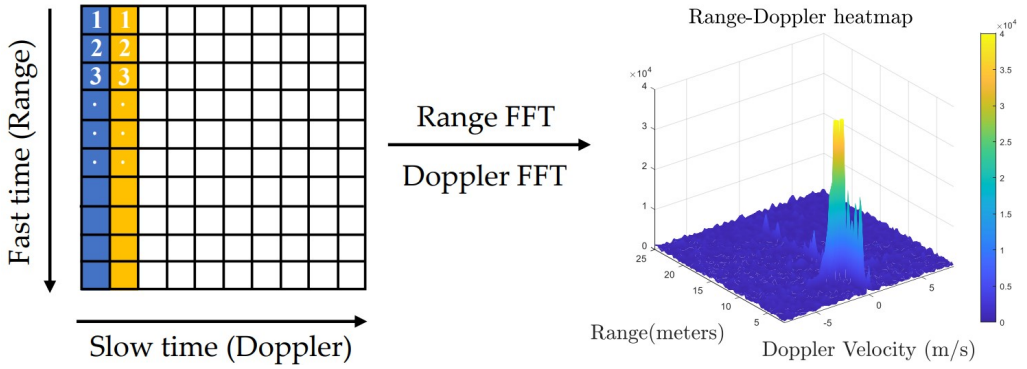
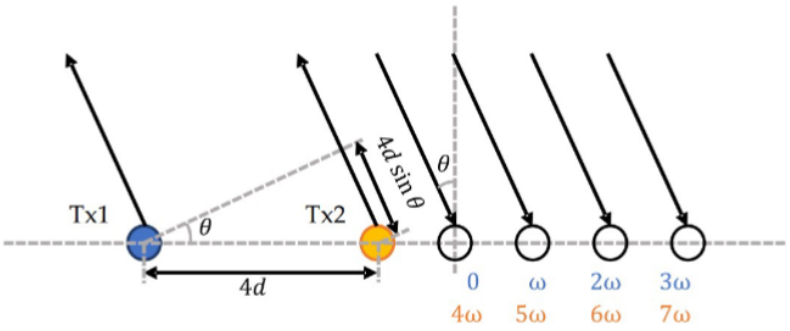
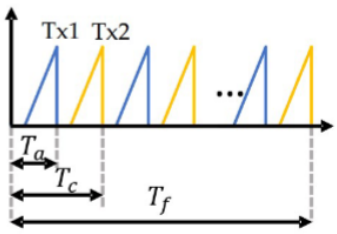


Figure 2.2: An illustration of slow-fast-time matrix on the left and Range-Doppler-map on the right.

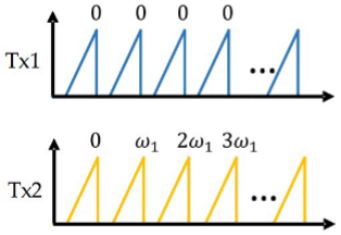
The angle of arrival is obtained through the phase shift between array channels in the antenna. While Single-Input Multiple-Output (SIMO) radar employs one transmitting (T_x) antenna and multiple receiving (R_x) antennas, Multiple-Input Multiple-Output (MIMO) radar incorporates multiple T_x and R_x antennas to create a virtual array with an increased number of channels. As visualized in Fig. 2.3, signal separation in MIMO radar is achieved using time-division multiplexing (TDM) or Doppler-division multiplexing (DDM). Furthermore, a 4D data structure containing range, azimuth, elevation, and Doppler information, is created, as described in [16], to facilitate the processing of the acquired data.



(a) Sending and receiving MIMO system.



(b) Time-division multiplexing of the received signals.



(c) Doppler-division multiplexing of the received signals.

Figure 2.3: An illustration of signal separation in MIMO radar.

Additionally, to improve the radar system accuracy and remove outliers, various techniques such as constant false alarm rate (CFAR) on the Range-Doppler maps together with filtering techniques are employed. The result of the signal processing of the radar beams is the already mentioned Fig. 2.1, the foundation of our studies.

2.2 Dataset

In this section, we will discuss the datasets used in this thesis. Since different tasks require input data in different formats, i.e., the data for SemSeg is a point cloud of the whole scene, the data for classification is the points of a cluster, and the data for sequential classification if the points of the same cluster over several consecutive time frames, there are three types of datasets used: SemSeg dataset, classification dataset, and sequential classification dataset. Each dataset is derived from the same point cloud data consisting of the same scenarios, and will be introduced later in this section.

To narrow the scope of the study we decided with Sensrad AB to focus on static environments leading to two simplistic scenarios (a) and (b), see Fig. 2.4a and 2.4b, and a simple real-world scenario (c), see Fig. 2.4c. These new scenarios prove instrumental in the evaluation of our models and allow us to study different isolated edge-case scenarios effectively.

Scenarios (a) and (b) consist of actors creating isolated challenging situations, designed to test the capabilities of our algorithms in various situations, such as walking close to each other, standing still (zero Doppler frequency), and crossing paths. While scenario (c) is a more complex real-world scenario collected at a crossroad where every road user performed unpredictable actions.

(a) Scenario 1 (artificial).

(b) Scenario 2 (artificial).

(c) Scenario 3 (real-world).

Figure 2.4: Example images of the three available static scenarios.

Scenario (a) with $7.5k$ selected frames is the simplest among the three, with minimal background action and only three actors. The scene takes place on sloped ground, confined by trees, and involves thus numerous environmental points that could be wrongly predicted. As depicted in Fig. 2.5, the scenario primarily includes pedestrians and a few bicyclists, with no cars present.

Scenario (b) with $34k$ selected frames is by far the biggest scenario and also relatively simple. It plays on flat asphalt ground with minimal ground detections. The open and unobstructed space allows for long-distance measurements. This scenario features four actors and introduces cars into the scene, adding an additional layer of complexity.
Scenario (c) with $4.5k$ selected frames represents a real-world scenario at a crossroad, with increased complexity due to the presence of a large number of objects simultaneously. Additionally, there are more environmental objects in the scene, such as traffic lights, which further contribute to the intricacy of this scenario.

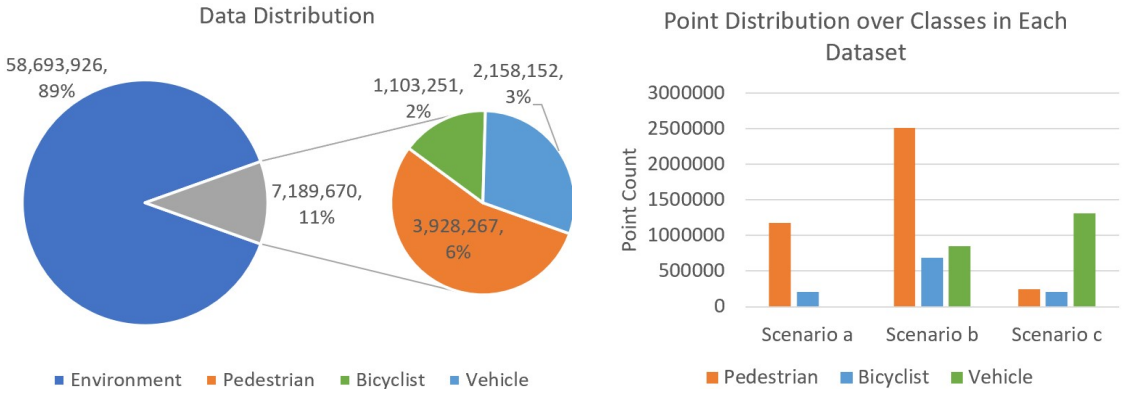


Figure 2.5: Point-wise class distribution of the selected frames.

Since we only have three well-annotated scenarios, we have chosen a validation set from each scenario. As unlabeled scenarios for quantitative testing are not yet accessible, we have utilized the validation set for both validation and testing purposes.

Finally, for three different perception tasks: semantic segmentation (SemSeg), classification, and sequential classification, we need three corresponding dataset variants by leveraging on annotations or a neural network combined with multiple conventional methods illustrated as Fig. 2.6. In general, dataset v1 represents an ideal situation that only considers annotations, whereas dataset v2 represents a more realistic situation that uses a neural network for SemSeg and conventional methods.

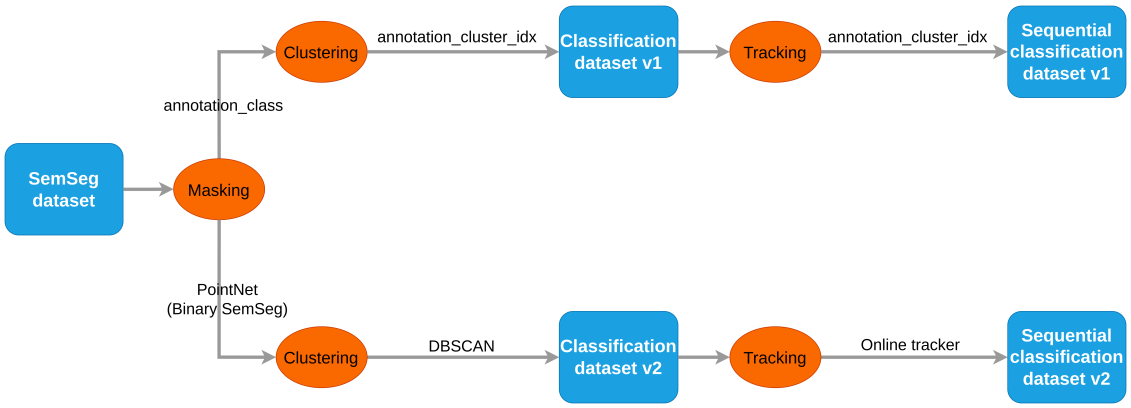


Figure 2.6: Flowchart of different dataset variants.

2.2.1 Annotation

The data is annotated by an external labeling company, leveraging an automated projection-based method for cost efficiency. This method involves mounting a camera on top of a radar system. The camera image utilizes SemSeg and offline Multi-Object Tracking (MOT) to detect, track, and classify objects within its field of view (FOV). Subsequently, radar points are projected onto the corresponding camera image and labeled according to the identified objects. As a result, the annotations include both point-wise labels and cluster indices to track each instance across different time frames.

2.2.2 SemSeg Dataset

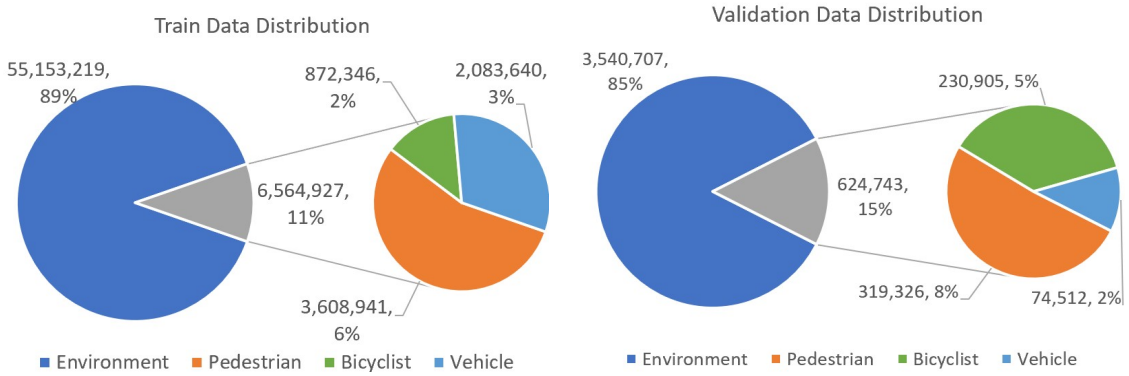
The SemSeg dataset is the unmodified point-wise labeled dataset mentioned above, which is annotated by Asymptotic AI. To get a dataset that fits our situation best, a manual selection process is applied to filter for well-annotated frames, therefore, the dataset is reduced to 46,363 frames. By selecting the most representative sequence from each scenario for validation, it leads to a split of 93% (43,277 frames) belonging to the training set, and 7% (3,086 frames) belonging to the validation set.

In Fig. 2.7, two significant discrepancies are noticeable. First, within the larger training set, approximately 90% of all points belong to the environment, leaving a mere 2% associated with the bicyclist class. This imbalance presents challenges in the development and evaluation of our models and highlights the importance of addressing such issues in our algorithms

Secondly, a disparity is observed between the training and validation sets, characterized by a relatively high representation of bicyclists and a comparably low representation of cars within the validation set. Despite this, it is important to note that the validation bags have been carefully selected to incorporate intriguing edge cases, serving as critical elements for robust validation.

Another factor to consider is the variability of points per frame, as visualized in Fig. 2.8. Throughout this thesis, we standardized the number of points to 4069 per frame across all algorithms to maintain consistency and ensure comparability. As a result, 93% of the frames were upsampled while 7% (marked in red) were downsampled, thus causing a slight reduction in the resolution of some frames. The distribution of points, illustrated in Fig. 2.7, reveals a significant imbalance.

As for the masking dataset, for the binary SemSeg task, we simply merge all non-environment labels into one object class visualized as the grey fraction in the pie charts.



(a) Number of points in training set: 61,718,146.

(b) Number of points in validation set: 4,165,450.

Figure 2.7: Point-wise statistics of SemSeg dataset v1.

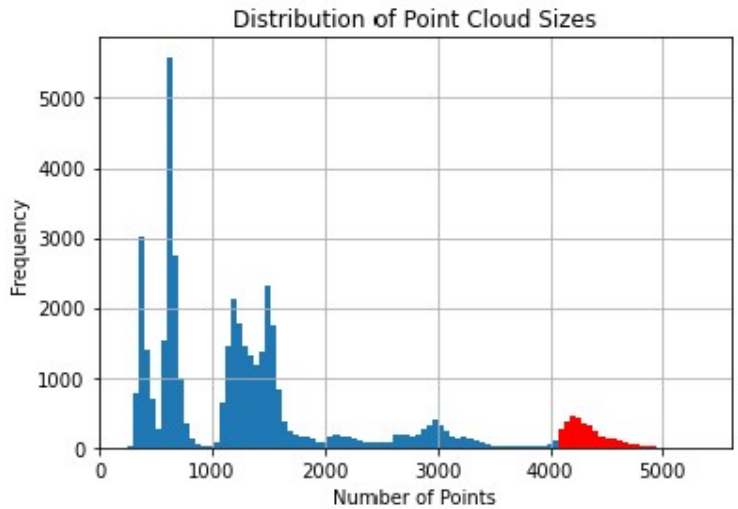


Figure 2.8: Histogram of the number of points in each frame in SemSeg Dataset v1, where the frames with more than 4096 points are marked in red.

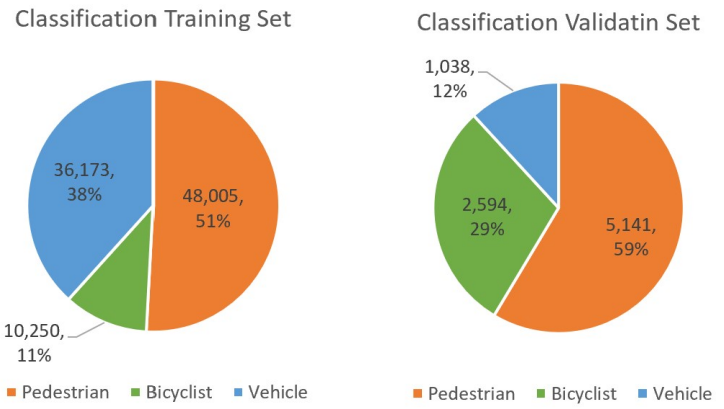
2.2.3 Classification Datasets

In contrast to the SemSeg Dataset, which comprises an entire point cloud per frame, the classification dataset only includes a single instance or object per frame, termed a cluster. These clusters are extracted from the original point cloud through two distinct methods.

The first method involves directly clustering instances based on the annotated cluster indexes. This results in relatively high-quality clusters, leading to the creation of an ideal classification dataset called classification dataset v1. The second method extracts clusters from the TSN or the perception pipeline of Sensrad AB, creating a classification dataset designated as v2.

Dataset v2 differs from v1 by introducing more realistic scenarios that include noisy clusters which are false cluster predictions that aren't associated with any specific class. It also includes instances of multi-cluster entities, where DBSCAN clusters multiple objects into one, as well as larger instances broken down into several smaller clusters. Therefore, v1 holds more academic value, demonstrating the potential of the algorithms when presented with high-quality data. On the other hand, v2 is utilized to train the classification network, serving as part of the pipeline to prepare it for data closely resembling the actual use case.

Fig. 2.9 presents a visualization of the class distribution, which corresponds to the distribution in Fig. 2.7 presented in the previous SemSeg dataset. However, instead of individual points, it is shown in clusters comprising more than six points. As previously stated, there are noticeable discrepancies in the distribution of bicyclists and vehicles between the training and validation sets. This results in a minor under-representation of vehicles and a slight over-representation of bicyclists in the performance evaluation.



(a) Number of clusters in training set: 94,428.

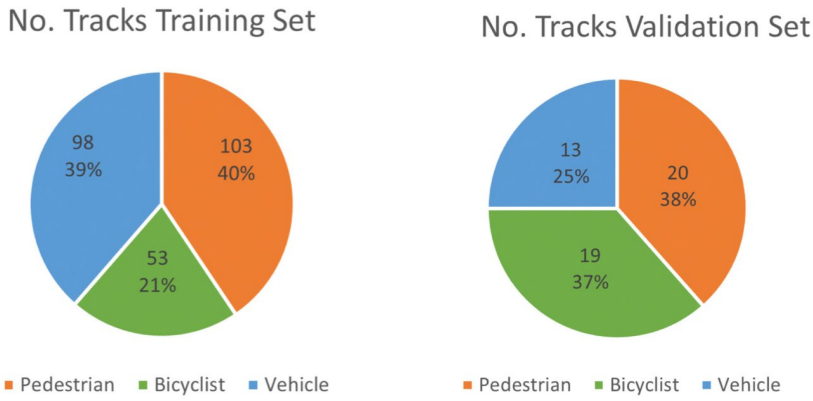
(b) Number of clusters in validation set: 8,773.

Figure 2.9: Cluster-wise statistics of classification dataset v1.

2.2.4 Sequential Classification Datasets

The sequential classification dataset has exactly the same clusters as the classification dataset, but it also has an additional track ID for each cluster to facilitate tracking across multiple frames. This dataset is crafted with the specific aim of serving sequential classifiers, as discussed in Chapter 6. Similar to the previous datasets, this one also features two versions: v1, which is created using track labels from annotations, and v2, crafted with track labels from the MOT pipeline of Sensrad AB.

In this dataset, v1 expresses higher quality compared to v2, through longer tracks. The distribution of tracks and average length for v1 is illustrated in Fig. 2.10 and Tab.2.1 respectively.



(a) Number of tracks in training set: 254.

(b) Number of tracks in validation set: 52.

Figure 2.10: Track-wise statistics of sequential classification dataset v1.

Table 2.1: Average length of sequential classification dataset v1.

	Pedestrian	**Bicyclist**	**Vehicle**
Training set	466	193	369
Validation set	257	136	79

2.2.5 Analysis

The analysis of clusters in classification dataset v1 provides insight into the general attributes of the objects. The histogram shown in Fig. 2.11 portrays the distribution of the complete training data. Every sample in the histogram corresponds to a cluster, falling under the classes of pedestrian (red), bicyclist (green), and car (blue).

In Fig. 2.11a, the **volume** for each cluster is calculated using the formula $v = \sigma_x\sigma_y\sigma_z$, where σ_i is the standard deviation of the cluster spread in dimension i.

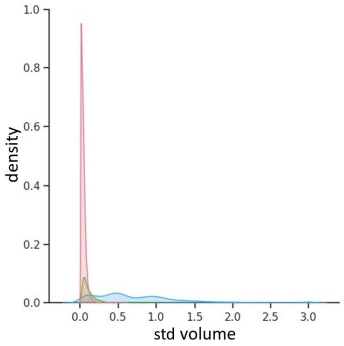
One can observe that while the volumes of pedestrians and bicyclists are relatively similar, the volume of cars is often larger, which intuitively aligns with reality. However, there are instances where cars exhibit volumes comparable to pedestrians and bicyclists. This indicates that volume is not an exclusively distinguishing factor for the car class.

In Fig. 2.11b, the **mean Doppler frequency** is computed as $\mu_d = mean(|p_d|)$, where p_d represents the Doppler value of the cluster points. This feature is highly distinguishable; pedestrians tend to occupy a specific area, while cars and bicyclists often move at faster speeds. However, the majority of cars in the dataset are stationary.

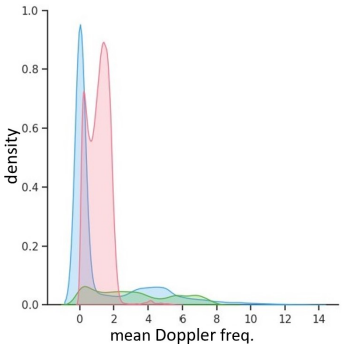
Another interesting feature is the **standard deviation of the Doppler frequency** within a cluster, as seen in Fig. 2.11c. Cars, being solid objects, exhibit lower standard deviation, whereas pedestrians with moving extremities have a higher standard deviation.

The **power** illustrated in Fig. 2.11d and Fig. 2.11e is centered for all classes. However, the standard deviation of power in Fig. 2.11e reveals that cars, being right-skewed, have greater variance than pedestrians, which are left-skewed.

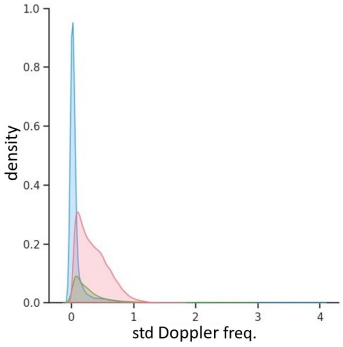
In Fig. 2.11g, the dataset exhibits a noticeable bias where background cars tend to be farther away in comparison to the scripted movements of pedestrians and bicyclists positioned directly in front of the radar system.



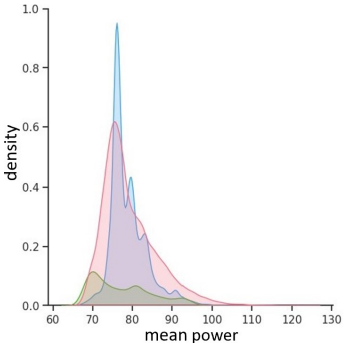
(a) cluster volume.



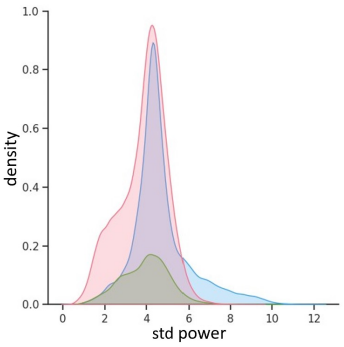
(b) mean-Doppler.



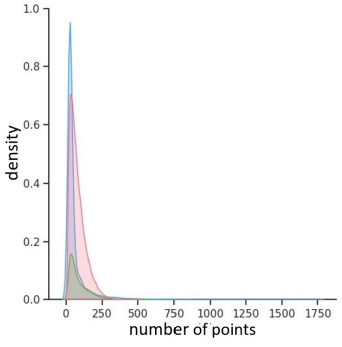
(c) std-Doppler.



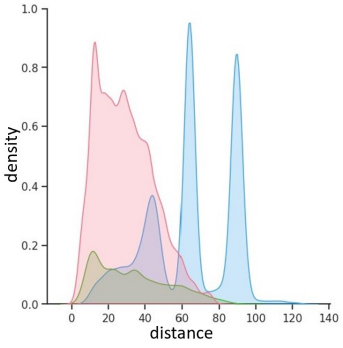
(d) mean-power.



(e) std-power.



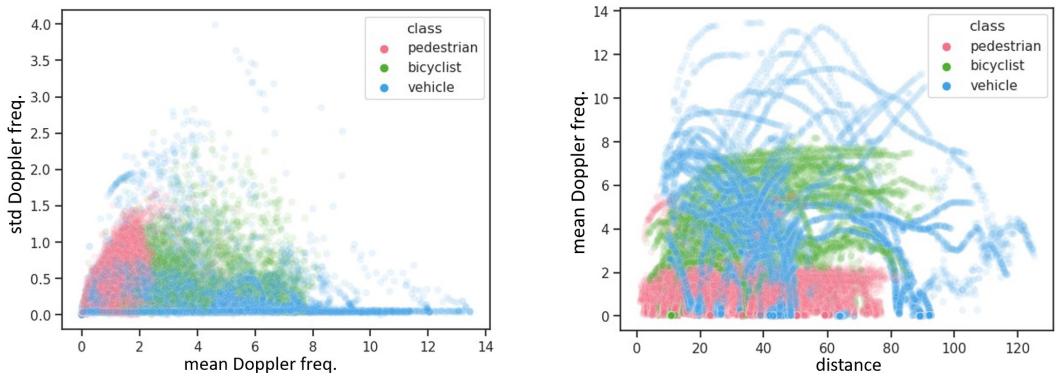
(f) Number of points in every cluster.



(g) Euclidean distance of clusters to sensor.

Figure 2.11: Kernel Density Estimate (KDE) over the classes with pedestrians in red, bicyclists in green, and vehicles in blue.

Fig. 2.12 shows a subsection of the combined attributes. One of the most compelling combinations is mean Doppler with standard deviation Doppler, as it creates a space where the classes tend to separate. Another interesting combination is mean Doppler with distance. The trajectories of cars can be seen in blue. In sequential approaches, the network could potentially leverage such patterns.



(a) std vs mean Doppler. **(b)** distance vs mean-Doppler.

Figure 2.12: The scatter plot illustrates the combination of features leads to new feature representations.

Visualizing the plots is valuable as it enables the analysis of data for patterns based on the physical meaning that the networks might learn. The plotted representations in Fig. 2.11 and Fig. 2.12 intuitively show the potential for class separation. However, neural networks must independently learn their appropriate feature spaces to separate the data and make decisions based on the raw point cloud data, which could differ from our physically derived patterns. Nonetheless, if a clear pattern were to be discovered, the network would be expected to learn it. Based on our assessment, it appears that such high-distinguishing spaces may not be present.

3
Theory

In this chapter, we delve into the various theories utilized throughout this thesis, providing a broad understanding of our investigative approach. Further elaboration on how these theories have been implemented will be explained in the following chapters.

3.1 Perception Tasks for Point Clouds

To understand the environment once a point cloud is generated by the sensor, many tasks can be done based on different purposes. In this section, we only introduce three perception tasks that are the most relevant to this thesis: classification, object detection, and SemSeg.

1. **Classification** is a smaller task compared to the other two since it does not directly interact with the whole point cloud but instead the extracted clusters of interest. It requires a model, also known as a classifier in this task, to assign predefined labels to input data instances based on their features. That is, in our case, a classifier needs to decide to which object class an input cluster belongs. Hence, the ground truth labels are the object class of each corresponding cluster, and a classifier is evaluated with precision, recall, and F1 score.
2. **Object detection** is a widespread task in the perception field as it can directly output useful information from a point cloud. It consists of two subtasks: localization and classification. The localization subtask asks a model to produce a 3-dimensional bounding box (bbox) in a point cloud for every object of interest. A bbox can be parameterized by the coordinates of its centroid, width, length, height, and optional angles according to needs. And the classification subtask is exactly the one introduced above. I.e., an object detection model, also known as an object detector, needs to further identify the object class within every bbox. So the ground truth labels contain parameters that define each bbox and the corresponding object class, and an object detector is evaluated by two types of metrics: IoU/mIoU for localization subtask, and precision, recall, and F1 score for classification subtask.
3. **SemSeg** is a similar task to object detection since it also processes the whole point cloud. However, instead of performing two subtasks at the same time, it solves a totally different task that assigns a predefined class label to every single point in a point cloud. This task involves capturing the fine-grained details and spatial relationships within a point cloud, requiring a model to

leverage both low-level point features and high-level contextual information. Therefore, the ground truth labels are the object class of every point in a point cloud, and a SemSeg model is evaluated by IoU/mIoU.

3.2 Common Structures of Neural Networks

Neural networks (NNs), a subset of machine learning, are complex mathematical models trained on data. They consist of interconnected neurons organized in layers. Each neuron performs a linear computation through its weights and bias on the input data and applies a non-linear activation function afterward. In this section, we introduce three fundamental components used in Chapter 6 for our sequential NNs: Fully Connected layers, Convolutional layers, and the Long Short-Term Memory layer/unit.

Fully Connected Layers or dense layers signifies that each neuron in this layer is connected to all the neurons in the preceding layer. Fully connected layers are often used in sequences, where sequences bigger than three are also called Multilayer Perceptrons (MLPs). The following equation

$$y_j(x) = f(\sum_{i=1}^{n} w_{ji}x_i + b_j) \tag{3.1}$$

represents a single layer that performs a weighted sum of all its inputs $\mathbf{x} \in \mathbb{R}^i$ using the weights $\mathbf{w} \in \mathbb{R}^{j \times i}$, adds a bias term $\mathbf{b} \in \mathbb{R}^j$, and then applies a non-linear activation function f to generate $\mathbf{y} \in \mathbb{R}^j$ the output, which is the input on the succeeding layer.

Fig. 3.1 visualizes the matrix multiplication of the input vector $\mathbf{x} \in \mathbb{R}^{1\times 9}$ and the learnable weights $\mathbf{w} \in \mathbb{R}^{9\times 4}$ mentioned in Eq.3.1 to generate the output vector $\mathbf{y} \in \mathbb{R}^{1\times 4}$, note that in the visualization the bias therm $\mathbf{b} \in \mathbb{R}^{1\times 4}$ is not included for simplicity but would be added in the activation function f. On the right, the corresponding neural structure is illustrated.

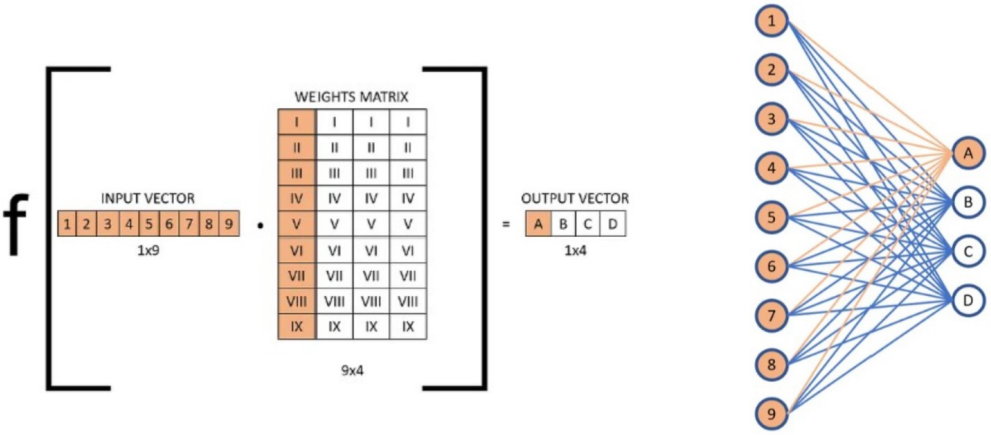


Figure 3.1: Calculation scheme and neural structure of a fully connected layer.

Convolutional Layers apply a series of filters (also known as kernels) to the input data to extract spatial features such as edges, and corners. A simple 2D convolutional layer with batch and channel size one is illustrated in Eq.3.2, where $\mathbf{x} \in \mathbb{R}^{i \times j}$ is the 2-dimensional input data and $\mathbf{w} \in \mathbb{R}^{k \times l}$ and $b \in \mathbb{R}^1$ are the learnable parameters kernel and bias. The output of the convolution is inserted into a non-linear activation function f leading to the 2-dimensional output $\mathbf{y} \in \mathbb{R}^{m \times n}$. The actual size of $\mathbf{y}$ can be calculated with Eq.3.3, where w and h is the width and height of the input, k is the kernel size, p is the zero padding, and s the step size called stride

$$y_{ij}(x) = f(\sum_k \sum_l x_{i+k,j+l} w_{kl} + b), \tag{3.2}$$

$$y_{width} = \frac{w - k_{width} + 2p}{s} + 1, \qquad y_{hight} = \frac{h - k_{hight} + 2p}{s} + 1. \tag{3.3}$$

Fig. 3.2 illustrates the formula for an input $\mathbf{x} \in \mathbb{R}^{3 \times 3}$ and kernel $\mathbf{w} \in \mathbb{R}^{2 \times 2}$ leading to the output $\mathbf{y} \in \mathbb{R}^{2 \times 2}$. Where also in this example the bias b is left out however could be added within the activation function f, stride $s = 1$, and zero padding $p = [0, 0]$. On the right side is the neural structure where it gets visible that the convolutional layer only differs from a fully connected layer by structural missing connections.

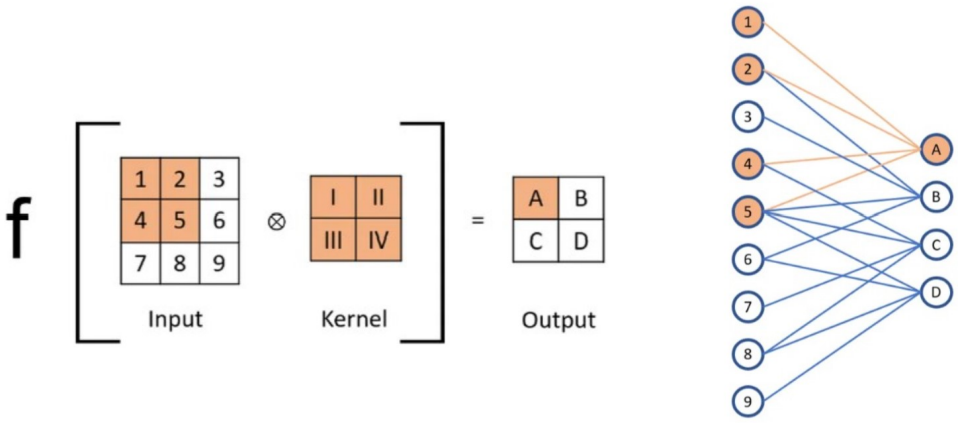
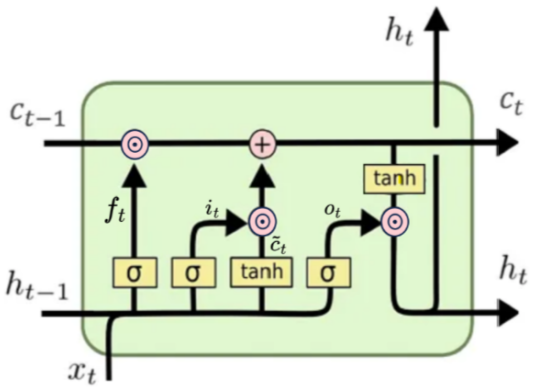


Figure 3.2: Calculation scheme and neural structure of a convolutional layer.

Long Short-Term Memory (LSTM) is a distinct variety of Recurrent Neural Network (RNN) architecture that uses a memory management system to handle temporal sequences and their dependencies. The LSTM accomplishes this through a dual memory system consisting of short-term memory h and long-term memory c. The regulation of these memories is executed by the three gates: input gate i, output gate o, and forget gate f, visualized in Fig. 3.3 as a diagram and as formulas on the right. Since the LSTM is one of the main focuses in this thesis, section 6.4, the flow within the unit is clearly explained in detail.



$$
\begin{aligned}
f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o) \\
\tilde{c}_t &= \tanh(W_c x_t + U_c h_{t-1} + b_c) \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
$$

Figure 3.3: LSTM diagram [1] and corresponding equations.

Input Vector ($x_t \in \mathbb{R}^d$) is the incoming data that the LSTM cell processes for a given timestep.

Forget Gate's Activation Vector ($f_t \in [0,1]^h$) is calculated by applying a sigmoid function to the weighted sum of the input vector x_t and the previous timestep's hidden state h_{t-1}, plus a bias term. The weights W, U and bias b are learned during training. The output f_t is a vector with values between 0 and 1, indicating how much of the cell state c_{t-1} from the previous timestep should be forgotten (with 0 indicating "completely forget" and 1 indicating "completely retain").

Input/Update Gate's Activation Vector ($i_t \in [0,1]^h$) decides how much of each new candidate value $\tilde{c}_t$ should be added to the cell state c_{t-1}. Like the forget gate f, it uses a sigmoid function applied to the weighted sum of the input vector x_t and the previous timestep's hidden state h_{t-1}, plus a bias term.

Cell Input Activation Vector($\tilde{c}_t \in [-1,1]^h$) is a vector of new candidate values that could be added to the cell state c_{t-1}. It is calculated by applying a tanh function (which outputs values between -1 and 1) to the weighted sum of the input vector x_t and the previous timestep's hidden state h_{t-1}, plus a bias term. The cell input activation vector $\tilde{c}_t$ is elementwise multiplied by the input gate's activation vector i_t to create an "update" for the cell state c_{t-1}.

Cell State Vector($c_t \in \mathbb{R}^h$) is the "memory" of the LSTM unit. It is updated by forgetting some things (according to the forget gate's activation vector f_t), and adding some new things (according to the input gate's activation vector i_t and the cell input activation vector $\tilde{c}_t$).

Output Gate's Activation Vector($o_t \in [0,1]^h$) is calculated similarly to the other gates. However, its role is to decide how much of the (newly updated) cell state c_t should be outputted as the hidden state h_t for this timestep.

Hidden State Vector($h_t \in \mathbb{R}^h$), has two arrows leaving the block, the under arrow h_t is the hidden state of the next time instance, the upper arrow h_t is the output vector of the LSTM unit. Hereby h_t is calculated by applying a tanh function to the cell state vector c_t (which pushes the values to be between -1 and 1), and then elementwise multiplying by the output gate's activation vector o_t.

Weight Matrices and Bias($W \in \mathbb{R}^{h \times d}$, $U \in \mathbb{R}^{h \times h}$, $b \in \mathbb{R}^h$) are the parameters of the LSTM unit that are learned during training. As visible in the equations, each gate has its own weight matrix and bias vector, and there's also a separate weight matrix and bias vector for calculating the cell input activation vector $\tilde{c}_t$.

3.3 Neural Networks for Point Clouds

In this section, we introduce the NNs of Chapter 4 used for semantic segmentation on radar point clouds. Point clouds in general, as described by [2], have distinct characteristics that make them challenging to work with: they are unordered, exhibit interactions among points, and have semantic invariance to specific transformations. To address these challenges, various methods have been proposed to process point clouds, which can be broadly categorized into three groups as in [17]: Projection-based, Discretization-based, and Point-based.

Projection-based methods, while relatively fast, suffer from unavoidable information loss due to the inherent nature of projection. As a result, this thesis will primarily focus on the latter two approaches, Discretization-based and Point-based methods, which have demonstrated superior performance in benchmark tests, such as the

SemanticKITTI [18] and the nuScenes [19]. In the rest of this section, we will delve into the architectures of our tested methods and discuss their advantages and limitations.

3.3.1 Point-based

Point-based networks operate directly on the point clouds, without the need to convert the data into another format (like images or voxels). This is a significant advantage because converting point cloud data into other formats can often result in the loss of information. Point-based networks usually employ shared Multi-Layer Perceptrons (MLPs) for the extraction of point-specific features and incorporate some kind of pooling layers to obtain global features.

PointNet, first introduced in 2017 by [2], is a pioneering point-based method still renowned for its simplicity and efficiency. In our thesis, we have leveraged PointNet as a critical component in our Two-Stage Network (TSN) as well as our sequential classifiers. Therefore, it is explained in more detail compared to the other architectures.

The architecture of PointNet is depicted in Fig. 3.4, and it is composed of the following key components:

1. **Input Transform:** This is a mini NN called T-Net, which aligns the input point cloud to a canonical (standardized) coordinate frame, helping to ensure that the model is invariant to transformations such as rotation and scaling.
2. **Shared MLPs:** Each point in the point cloud is processed independently through shared MLPs. This means the weights of the MLPs are the same for each point, which allows the network to handle unordered point sets.
3. **Max Pooling Layer:** After the shared MLPs, a max pooling operation is applied. This operation essentially takes the maximum value of each of the MLPs' output features across all points, creating a global feature vector that represents the entire point cloud. This feature vector is invariant to the order of input points.
4. **Classification Head:** This head classifies the entire point cloud into a category. After generating a global feature vector from the shared MLPs and max pooling layer, this vector goes through Fully Connected (FC) layers that map these global features to class probabilities.
5. **Segmentation Head:** This head classifies individual points in the point cloud. It combines local and global features for each point, and these features are passed through shared MLPs to produce class predictions for each point in the point cloud.

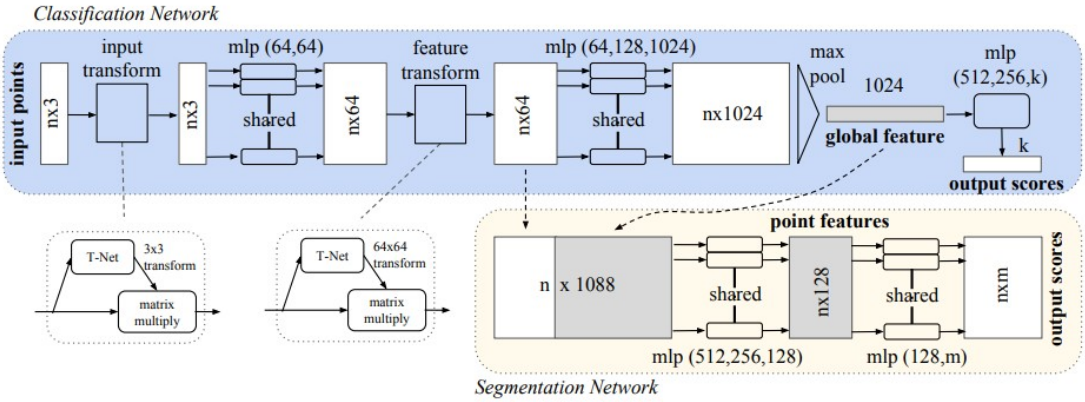


Figure 3.4: PointNet architecture for classification and SemSeg [2].

PointNet++ [20] is the successor of PointNet. The primary advancement in PointNet++ lies in its hierarchical approach, where it applies PointNet recursively on nested subsets of the input point cloud, capturing local features with increasing context. This feature extraction on multiple scales accommodates a wider range of object scales and improves performance with intricate details.

KPConv [21], short for Kernel Point Convolution, is a method designed for dealing with point-based oriented convolutions utilizing deformable kernels. The central idea is to treat each point in the cloud as an individual convolution kernel center, thus ensuring the alignment of the convolution operation with the underlying surface of the 3D structure. This approach effectively bridges the gap between irregular point clouds and regular convolutions, allowing the model to capture complex geometries and patterns more accurately.

RSConv [22], or Relation-Shape Convolution, is a method that seeks to enhance the feature learning capabilities in point cloud data by incorporating geometric relationships between points. The central concept is to establish a relationship between each point and its neighbors and use a shared Multi-Layer Perceptron (MLP) to encode these relations into features.

3.3.2 Discretization-based

Discretization-based methods treat point clouds as a special form of 3D volumetric data. In these methods, the 3D space is typically divided into a predefined grid, effectively converting the irregularly distributed point cloud into a regular volumetric representation such as voxel grids or multi-view projections. Convolutional Neural Networks (CNNs) can then be applied to learn features from these grid representations. However, these methods often face a trade-off between resolution and computational resources due to the highly increased dimensionality of 3D data.

MSSVConv [23], or Multi-Scale Sparse Voxel Convolution, is a method that specifically handles sparse point cloud data. The core idea is to divide the 3D space into

voxels at multiple scales and apply sparse convolutions on these voxelized representations to extract features.

Minkowski [24] uses the Minkowski Engine as the backbone, which is a specialized framework for sparse tensors, enabling highly efficient operations in high-dimensional spaces. The key concept is to discretize the input space into a voxel grid and handle it as a sparse tensor.

3.3.3 Combinations

Combinatorial methods encompass models which either leverage point-based and discretization-based modules or cannot be distinctly classified under either of those methodologies.

PPNet [25] short for Pose Pole Net, is a deep residual network operating point-wise within a grid using layers of transformations and local aggregation. At its heart is a straightforward local aggregation operation. In this process, PosPool solely computes the average of the features surrounding a central point, which in turn generates the features of the specific center point.

PVCNN [26] short for Point-Voxel Convolutional Neural Network, is a method designed to effectively and efficiently process 3D point cloud data. The core idea is to exploit the advantages of both point-based and voxel-based methods by dynamically allocating computations between raw point clouds (PointNet) and voxelized representations (3D CNNs). This dual-branch structure allows PVCNN to benefit from the detailed preservation of point-based methods and the computational efficiency of voxel-based methods.

3.4 Naive Bayesian Classifier

Within this thesis, the simple Naive Bayesian classifier (NBC) [27] is employed, a Bayesian probabilistic model that assigns a posterior class probability to an instance. This model calculates the probability of each class and selects the label with the Maximum A Posteriori (MAP) value $\hat{y}$ as Eq.3.8.

Considering the Bayes theorem [28] below

$$P(y_j|x_i) = \frac{P(x_i|y_j)P(y_j)}{P(x_i)}, \tag{3.4}$$

to perform calculations feasibly, we omit the joint probability of feature x_i and class y_j and adopt the "naive" assumption that the features in vector $\mathbf{x}$ are independent of each other. This assumption allows us to derive the following equation

$$
\begin{aligned}
P(\mathbf{x}|y_j)P(y_j) &= P(x_1|y_j)P(x_2|y_j)\dots P(x_p|y_j)P(y_j), \\
&= \prod_{k=1}^{p} P(x_k|y_j)P(y_j),
\end{aligned}
\tag{3.5}
$$

which simplifies the calculations. Inserting Eq.3.5 into Eq.3.4 leads to the following new expression

$$
P(y_j|x_i) = \frac{\prod_{k=1}^{p} P(x_k|y_j)P(y_j)}{P(\mathbf{x})}. \tag{3.6}
$$

Since the denominator $P(\mathbf{x})$ does not depend on the class y, and hence is only a constant scaling factor it can be removed, resulting in the below relation

$$
P(y_j|x_i) \propto \prod_{k=1}^{p} P(x_k|y_j)P(y_j). \tag{3.7}
$$

The predicted class $\hat{y}$ is the class which maximizes the probability $P(y_j|x_i)$, which is known as the MAP, and organized below

$$
\hat{y} = \underset{y_j}{\operatorname{argmax}} P(y_j|x_i) = \underset{y_j}{\operatorname{argmax}} \prod_{k=1}^{p} P(x_k|y_j)P(y_j). \tag{3.8}
$$

3.5 Doppler Masking

Doppler masking is a simple technique to extract objects from the environment. This is established by a Doppler threshold d_{trsh}, which then classifies points p_i with an absolute Doppler value higher than d_{trsh} as an object $p_i^{cls} = 1$. Consequently, the points with an absolute Doppler value $|p_i^d| < d_{trsh}$ are classified as environment $p_i^{cls} = 0$, see the equation below

$$
p_i^{cls} = \begin{cases} 1, & \text{if } |p_i^d| \geq d_{trsh} \\ 0, & \text{if } |p_i^d| < d_{trsh} \end{cases} \qquad \forall i \in \{1, 2, \dots, N\}, \tag{3.9}
$$

where N is the number of points in the point cloud. The threshold value can be determined via a linear search that evaluates the trade-off between maximizing object points and minimizing the mislabeling of environmental points as objects. This can be formulated as a simple optimization problem focusing on maximizing the Intersection over Union (IoU) for the object class over the Doppler value d as follows

$$
d_{trsh} = \underset{d}{\operatorname{argmax}}\ IoU_{obj}(d). \tag{3.10}
$$

This is an intuitive, fast, and easy-to-implement method to detect objects from the environment, which is suitable for applications where only moving objects are of interest. However, the main shortcoming is that static objects will not be detected by this conventional approach.

3.6 DBSCAN

In this section, we discuss the classical Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithms used in the thesis. DBSCAN is a clustering algorithm [29] that groups points in close proximity to one another, based on their density, while simultaneously filtering out noise from regions that lack the required density distribution. This thesis employs the classical DBSCAN in TSN.

Classical DBSCAN algorithm requires two hyper-parameters to define the desired cluster density, search radius ϵ and the minimum number of neighbors n within ϵ. The algorithm is explained with the following six steps and the corresponding figure 3.5:

1. Identify core points p_i: A point is considered as a core point p_i if it has at least n neighbors within the search radius ϵ; otherwise, it is a non-core point $\bar{p}_i$.
2. Select an unmarked core point p_i and add it to the set of core points $\mathcal{S}_j$ with an associated cluster index j, and mark this core point.
3. If another unmarked core point p_i exists within the current search area defined by radius ϵ, add it to the set $\mathcal{S}_j$ with an associated cluster index j and mark it, and repeat this step until no unmarked core points p_i exist in this search area.
4. Repeat steps 2 and 3 until all core points p_i are marked.
5. For non-core points $\bar{p}_i$ with neighbouring points p_i within the search radius ϵ, add them to the corresponding set $\mathcal{S}_j$.
6. Remaining non-core points $\bar{p}_i$ are defined as outliers $\dot{p}_i$.

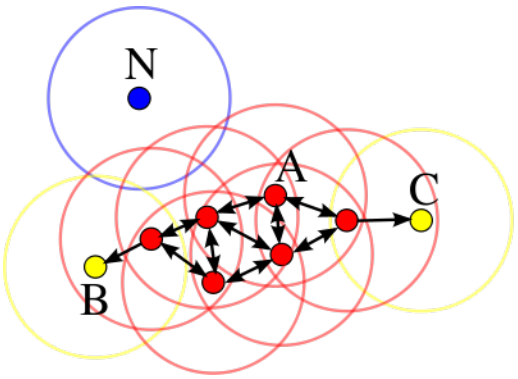


Figure 3.5: An illustration of classical DBSCAN from [3] with core points p_i in red, non-core points $\bar{p}_i$ in yellow, outliers $\dot{p}_i$ in blue, circles of radius ϵ, and the minimum neighbour points $n = 3$.

3.7 Evaluation metrics

To evaluate the performance of a model, some widely used metrics of perception tasks for point clouds, i.e., Intersection over Union (IoU), mean IoU (mIoU), preci-

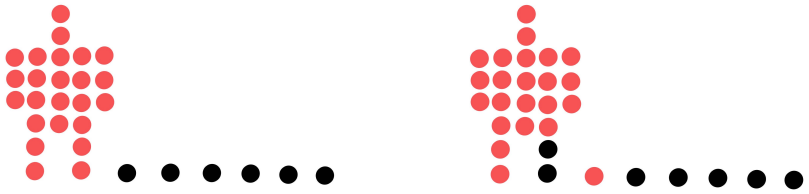
sion, recall, can F1 score will be introduced in this section.

Intersection over Union (IoU) is a fundamental evaluation metric in the field of object detection and semantic segmentation. It measures the overlap between a predicted region and a ground truth region by calculating the ratio of the intersection area to the union area of the two regions. It provides a quantitative assessment of how well an object is localized or how accurately a region is segmented. It can be expressed in a mathematical way below

$$\text{IoU} = \frac{TP}{TP + FP + FN}, \tag{3.11}$$

where TP, FP, and FN represent true positive, false positive, and false negative respectively. Apparently, IoU ranges from 0 to 1, where a value of 1 indicates a perfect overlap between the predicted and ground truth regions. An example of how to calculate IoU is illustrated in Fig. 3.6, from which IoU for pedestrian and environment can be calculated respectively below

$$\begin{aligned} \text{IoU}_{Ped} &= \frac{22}{22 + 1 + 2} = 88.0\%, \\ \text{IoU}_{Env} &= \frac{5}{5 + 2 + 1} = 62.5\%. \end{aligned} \tag{3.12}$$



(a) Ground truth. **(b)** Prediction.

Figure 3.6: An example of IoU calculation, where pink and black points represent pedestrian and environment respectively.

Mean Intersection over Union (mIoU), on the other hand, extends the concept of IoU to multiple classes by averaging the IoU values across all classes. It is commonly used to evaluate the overall performance of an object detection or semantic segmentation algorithm. In the previous example, mIoU can be calculated below

$$\text{mIoU} = \frac{\text{IoU}_{Ped} + \text{IoU}_{Env}}{2} = \frac{88.0\% + 62.5\%}{2} = 75.25\%. \tag{3.13}$$

Therefore, the mIoU metric is sensitive to both accuracy and coverage, providing a comprehensive measure of the model's ability to capture object boundaries and segment objects accurately. By utilizing IoU and mIoU, researchers and practitioners can quantitatively assess and compare the performance of different computer vision models, enabling advancements in the field and facilitating the development of more robust and accurate systems.

Precision and recall are essential metrics in object detection and classification tasks. Precision focuses on minimizing false positives, while recall aims to minimize false negatives. There is often a trade-off between precision and recall, and achieving a balance depends on the specific application. Hence, another metric, the F1 score, which takes the harmonic mean of precision and recall below

$$\text{F1 score} = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}, \tag{3.14}$$

is invented to provide a balanced evaluation. These metrics enable researchers to assess and optimize model performance in a reasonable way.

4
Comparison of Different Networks

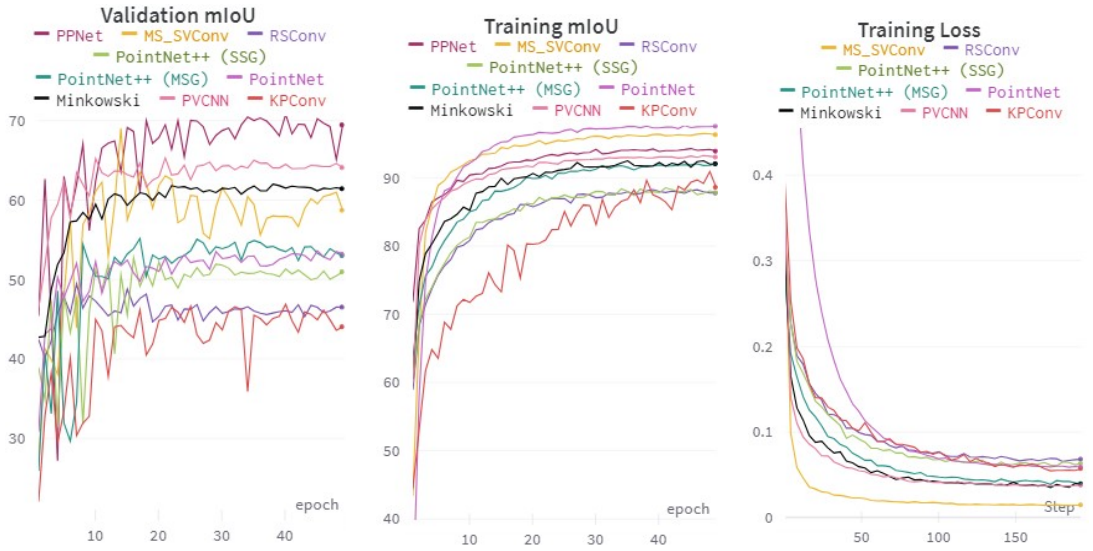
In this chapter, all the neural networks presented in Chapter 3.3 are assessed based on the criteria: prediction accuracy, model size, inference speed, and training time. The primary objective of this comparative analysis is to establish a benchmark that provides a comprehensive overview and uncovers performance trends across different architectures. This aids in making critical future decisions such as identifying which architectures are most appropriate for 4D radar data, and determining the requisite model complexity needed to address the task at hand. To mention the networks again for convenience, the point-based networks including PointNet [2], PointNet++ [20], the discretization-based networks including RSConv [22], KPConv [21], Minkowski [24], and MSSVConv [23], and the combination networks including PVCNN [26] and PPNet [25]. The extensive comparison conducted within the timeframe of a master's thesis is made possible due to the implementation of these networks in `torch-points3d` [30]. Without these, such a comprehensive comparison would have been infeasible. The details of training conditions for each network can be found in Appendix B.1.

4.1 Accuracy Evaluation

To evaluate the prediction performance of a network for the SemSeg task, the metric mIoU explained in section 3.7 is used. During the training process, every network is evaluated at every epoch on both the training and the validation dataset, leading to the results visible in Fig. 4.1, which shows the validation and training mIoU as well as the training loss. In general, the networks converge in an appropriate way, especially in view of the training mIoU. The test mIoU is in the beginning often bumpy, however, for most of the networks, smoothes over time due to the learning rate decay.

Tab.4.1 is sorted according to the converged validation mIoU. It is visible that PPNet with a mIoU = 69.48% is leading the board by a margin compared to PVCNN with a mIoU = 64.10%. The table is closed with KPConv reaching a mIoU = 44.05%. Interestingly, the validation mIoU from 44.05% to 69.48% is more spread than the training mIoU from 88.59% to 97.54%. Furthermore, the obvious discrepancy between validation and training mIoU indicates model overfitting to the training data visible in the last column mIoU_{Δ}. The tendency is that the mIoU_{Δ} is inversely proportional to the validation mIoU, showing the impact of overfitting on the model performance.

In reference to the loss depicted in Fig. 4.1c, distinct convergence rates, characterized by an elbow-shaped curve, can be observed. It is noticeable that the networks demonstrating the most pronounced elbow tend to achieve lower losses, with MSSVConv, represented by the yellow line, leading this trend. However, this observation does not entirely align with the convergence trends seen in the validation metric (mIoU) where MSSVConv is placed in the middle ranks.



(a) Validation mIoU. **(b)** Training mIoU. **(c)** Training negative log-likelihood loss.

Figure 4.1: Training and validation performance of different SemSeg networks.

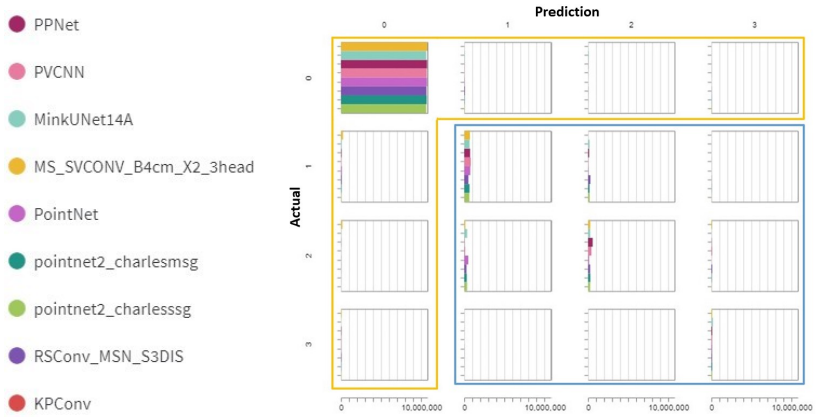
Table 4.1: Converged validation and training mIoU of different SemSeg networks as well as the difference between them in descending order of validation mIoU.

Neural network	Converged validation mIoU ↑	Converged training mIoU ↑	mIoU_Δ ↓
PPNet	**69.48%**	93.90%	**24.42%**
PVCNN	64.10%	93.02%	28.92%
Minkowski	61.44%	92.04%	30.60%
MSSVConv	58.73%	96.29%	37.56%
PointNet	53.19%	**97.54%**	44.35%
PointNet++ (MSG)	53.01%	92.03%	39.02%
PointNet++ (SSG)	50.97%	87.80%	36.83%
RSConv	46.52%	87.76%	41.24%
KPConv	44.05%	88.59%	44.54%

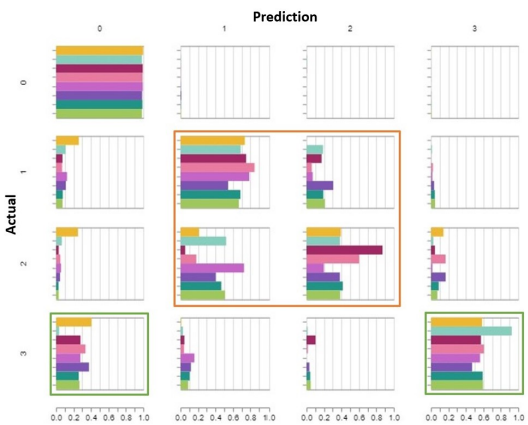
In order to analyze the predictive capabilities of the networks in more depth, three confusion matrices are illustrated in Fig. 4.2 juxtaposing the predicted labels with the true labels of the points. The subplots consist of Fig. 4.2a, the unmodified confusion matrix of all networks, Fig. 4.2b, standing for the class normalized version, and Fig. 4.2c, also class normalized but specifically focusing on the top three NNs that achieved the highest validation mIoU scores. All the matrices have the sequence represented by 'environment', 'pedestrian', 'bicyclist', and 'vehicle'. In

this configuration, the rows symbolize the true class, while the columns represent the predicted class, thus the diagonal elements stand for the correctly labeled points.

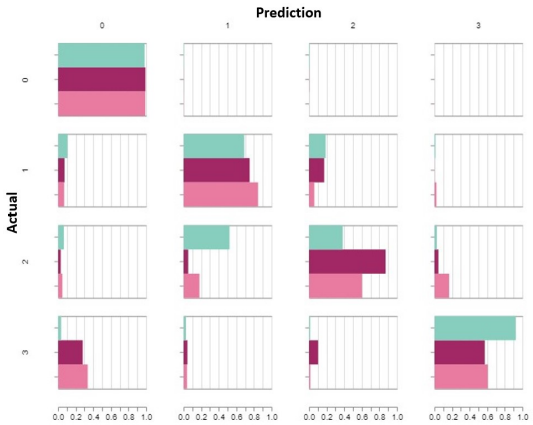
Regarding the matrix in Fig. 4.2a, it is visible that all the networks are fairly able to distinguish between environment (yellow section) and object class (blue section). However, from the matrix in Fig. 4.2b, two issues can be seen, on one hand, the general difficulty of the networks to distinguish between pedestrians and bicyclists, marked with the orange rectangle. And on the other hand, the high amount of undetected vehicles are marked with a green rectangle. Yet, it has also to be recalled that only a few instances of vehicles exist and thus it is hard to draw definitive conclusions from the few samples. Interestingly, at leas one of the aforementioned issues exist for the three top-performing networks shown in Fig. 4.2c as well, even though to a smaller extent.



(a) All NNs.



(b) All NN (normalized).



(c) Best 3 NNs (normalized).

Figure 4.2: Confusion matrices of different SemSeg networks with classes of environment (0), pedestrians (1), bicyclists (2), and vehicles (3).

To summarize this section, it can be said that the networks in general are able to detect objects fairly well with mIoU up to 70% given the class imbalance between

environment and objects. However, there is a tendency to overfit the training data. Regarding the confusion matrices, while the NNs are able to detect pedestrians and bicyclists, they have difficulties detecting vehicles. And Even though pedestrians and bicyclists are detected, the networks struggle to distinguish between them.

4.2 Resource Evaluation

For a real product that needs to run in real-time on an embedded system, like radars, the model size and speed matter a lot. On top of that, the time invested in training is also a factor to consider. To this end, all networks have been compared in model size and evaluated in terms of inference time on a GPU, NVIDIA RTX 3090, and training time on another GPU, NVIDIA GeForce GTX TITAN X, with detailed training conditions in Appendix B. Nevertheless, these three characteristics will be evaluated one by one in this section.

Model size, a quite valuable resource in a product, determines how much run-time memory is needed for a network. It is vital to choose a suitable model for some specific applications considering model size. Therefore, to get an overview of all tested models, the number of parameters in each NN is listed in TAB.4.2 in descending order. In general, point-based networks are relatively small in size, while discretization-based networks are much larger with the exception of RSConv which is even smaller than PointNet.

Table 4.2: Number of parameters of different SemSeg networks in ascending order.

Neural network	No. parameters ↓
PointNet++(SSG)	**1,398,724**
PointNet++(MSG)	1,727,468
RSConv	3,480,467
PointNet	3,536,469
MSSVConv	9,141,284
KPConv	14,080,192
PPNet	18,367,344
PVCNN	21,774,820

Inference speed is another important factor for selecting a proper model. It has less impact in offline applications, however, it is of great importance in online tasks. Thus, to simulate the real-time situation, every model is running inference with batch size 1 for 10 times in a row on GPU, NVIDIA RTX 3090. Due to the warm-up phase of a GPU, the inference speed at the first run is obviously slower than the others. Hence, we remove the slowest and the fastest results before calculating the average inference speed. Finally, the results are illustrated in Tab.4.3 in ascending order of average inference time with an input point cloud size fixed at 4096, which is a suitable value based on our dataset. Overall, point-based networks lead the board again with a huge gap. PointNet, the fastest network during inference, can reach

132.58 frames per second (fps), 7 times faster than the slowest network, MSSVConv, which can only run at 18.41 fps. On the other hand, the most accurate network, PPNet, is the second slowest model with 22.68 fps, meaning that it can not run in real-time even on a powerful GPU. According to current results, only four NNs can potentially run in real-time on an embedded product: PointNet, PointNet++, Minkowski, and PVCNN.

Table 4.3: Inference speed of different SemSeg networks on GPU (NVIDIA RTX 3090) in descending order.

Neural network	**Inference speed ↑**
PointNet	**132.58 fps**
PointNet++(SSG)	98.90 fps
PointNet++(MSG)	74.04 fps
Minkowski	68.76 fps
PVCNN	35.82 fps
KPConv	26.82 fps
PPNet	22.68 fps
RSConv	21.27 fps
MSSVConv	18.41 fps

However, discretization-based networks are generally faster than point-based networks in terms of inference speed as analyzed in [26], which is conflicted with our results. This is because of the small input size of point clouds in our application compared to other scenarios, e.g. LiDAR point clouds, with millions of input points. Therefore, it would be intriguing to investigate the decrease in inference speed of our different network architectures as the size of the input grows. The experiment is conducted by gradually increasing the number of input points for every network from 4,000 to 100,000 in increments of 1,000. Consecutively, for each network-point pairing 10 inference runs have been recorded. Regarding the computation of the average, the slowest and fastest run has been removed to exclude outliers. The tests have been conducted on our GPU, NVIDIA RTX 3090, and the results are illustrated in Fig. 4.3. PointNet family occupies the top three positions in the beginning, however, Minkowski outperforms PointNet++(MSG), PointNet++(SSG), and PointNet at around 5,000, 10,000, and 20,000 input points respectively. On top of that, Minkowski leads the board all the way until the end. It is also noticeable that only PVCNN and MS_SVConv drop almost linearly, meaning that they will probably become the top two with more input points. On the other hand, this experiment is not perfectly designed. E.g., we only duplicate existing points instead of adding new points in the scene, making the testing situation less realistic. Nevertheless, it is easy to implement within the limited time of this project and still provides valuable insights. In our application where the maximum number of input points is far less than 20,000, PointNet is still the first candidate in terms of inference speed. Even if the input size changes in the future due to more advanced radar technologies, this result can be used as a reference for selecting a proper model.

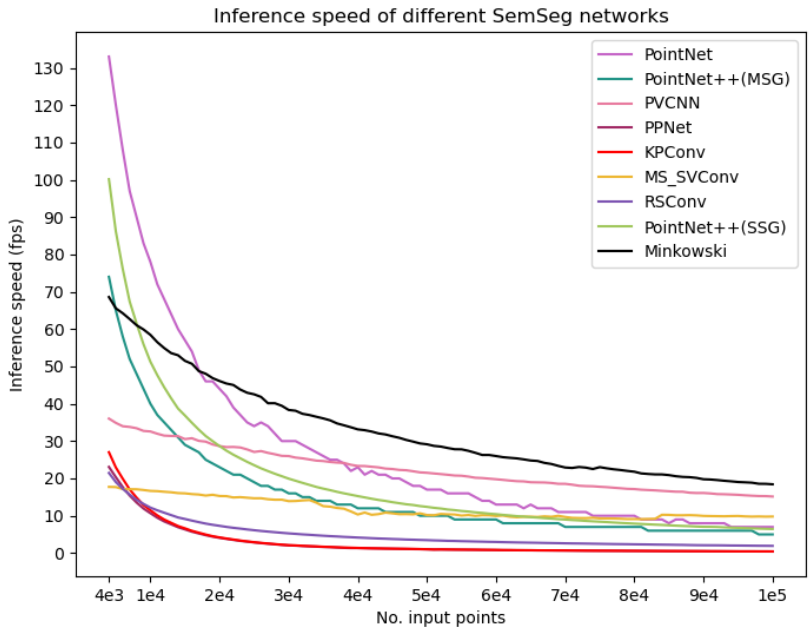


Figure 4.3: Inference speed of different SemSeg networks with different numbers of input points on GPU (NVIDIA RTX 3090).

Training time is the last factor we consider regarding the resources since it can be improved in an easier way, e.g., simply using a more powerful GPU for training, which is normally durable for commercial purposes, but still interesting to compare. Hence, all networks have been trained on the same GPU, NVIDIA GeForce GTX TITAN X, with different batch sizes but basically all used up the whole memory of 12 GB. Generally, the training time varies quite a lot from network to network. But clearly, point-based networks are located in the first half, but not the fastest ones anymore according to TAB.4.4, where Minkowski ranks at the top and only takes less than 2 hours, while the slowest and also the most accurate one, PPNet, takes more than 3 and half days.

Table 4.4: Training time of different SemSeg networks on GPU (NVIDIA GeForce GTX TITAN X) in ascending order.

Neural network	**Training time ↓**
Minkowski	**1h47m50s**
PointNet++(SSG)	4h17m54s
PVCNN	6h41m54s
PointNet++(MSG)	14h32m37s
PointNet	14h35m25s
MSSVConv	1d6h15m39s
RSConv	1d21h29m5s
KPConv	2d15h24m44s
PPNet	3d16h5m58s

4.3 Performance Analysis

The bubble plot in Fig. 4.4 concludes the aforementioned results in a single figure. It should be noted that instructions on how to interpret the plot are provided in the image description. Nevertheless, the inference speed and converged validation mIoU, are given by the x- and y-axis. While diameter, transparency and color correspond to the model size, training, and architecture type, respectively. Therefore, the ideal network would be in the upper right corner, rather small and fully transparent. Obviously, the highest-performing network is PPNet, a combination method. However, it tends to be rather slow, has a high memory demand, and necessitates extensive training time. On the other hand, PointNet is the fastest architecture. Regarding mIoU PointNet is situated in the mid-range. It requires rather few memory requirements and is also fairly fast to train.

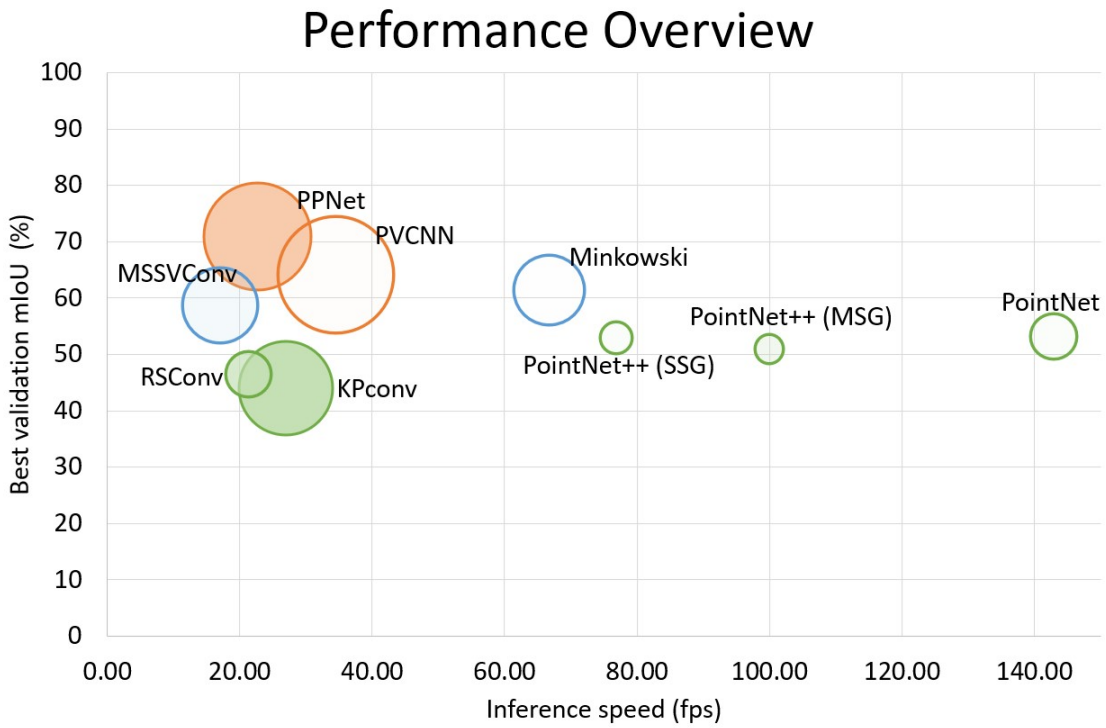


Figure 4.4: Performance overview of each neural network, where the x-axis represents the inference speed in fps, the y-axis represents the converged validation mIoU, the size of each bubble represents the model size, and the transparency of each bubble represents the training time with the fastest network corresponding to 100% transparent, the color represents the network type with point-based in green, discretization-based in blue, and combinations in orange.

According to Fig. 4.4, the network architecture types can be clustered into distinct groups, with the mIoU appearing as the primary determining factor. Combinatorial methods display superior performance, yet they come at the expense of substantial memory requirements. In contrast, point-based methods typically exhibit lower mIoU compared to other methodologies. Nevertheless, they encompass the most

efficient networks in terms of minimal memory requirements and short training periods, as exemplified by PointNet and PointNet++. It should be noted, however, that point-based methods also include RSConv and KPConv, which do not demonstrate particularly impressive performance on our dataset, being positioned in the lower-left corner of the figure. Networks relying on discretization fall within an intermediary range concerning mIoU performance, model size, and training time.

4.4 Discussion

Before further discussing the results, it is important to note that the results obtained in this study depend heavily on several factors: network configurations, which are chosen as the original ones in the library [30], learning setup, data augmentation, and dataset. Therefore, it is probable that a more optimized parameter search would yield different results.

The most notable observation of this chapter is the pronounced overfitting of the models. Higly possible due to the small variance within the training set, despite the use of data augmentation. It is also surprising that PointNet++, designed to extract local features, does not outperform its predecessor, PointNet, as usually the case. Reflecting on the evolution of these networks, it is clear that efforts to increase performance by capturing more nuanced details have resulted in increased model complexity. However, this approach appears to have led to diminished results and overfitting with our relatively simple radar data, as visible in 4.1 mIoU_{Δ}.

Therefore, it may not be the architecture of the networks at fault, but rather the level of model complexity. By reducing this complexity and thus solving the overfitting problem, we could potentially see a significant shift in prediction accuracy. Beneficial to encounter the overfitting would also be to increase the variety in the dataset. In addition, some more advanced data augmentation techniques are worth trying to mitigate the biased performance caused by the limited dataset size and imbalanced class distribution. E.g., global flipping, global rotation, and ground truth database sampling to only augment object classes in different ways.

5
Two-Stage Network

The Two-Stage Network (TSN) is a conceptual framework designed to tackle the challenging task of object classification in radar point clouds - a task with which the networks discussed in the previous chapter have had difficulties. The main approach is to decompose the segmentation task into three subsequent stages: binary SemSeg, also referred to as object masking, facilitated by a masking modul, clustering by DBSCAN, and object classification handled by a classification modul. Even though the implemented NN modules of the TSN are exclusively PointNet, alternative networks or classical methods can be employed. The choice of PointNet in this thesis is justified by its numerous advantages, such as the low computational complexity, adaptability, quick learning and testing phases, and its broad recognition within the research community as a benchmark model.

This chapter delves into the evolution of the TSN and involves three separate comparative studies. The primary comparison is between the TSN and its semantic segmentation counterpart. Additionally, there are two subsidiary comparisons within the masking and classification modules, wherein traditional methods are contrasted with neural approaches, as depicted in Fig. 5.1. Consequently, this chapter investigates the possible performance enhancements that could be achieved by breaking down SemSeg into individual, interchangeable subtasks.

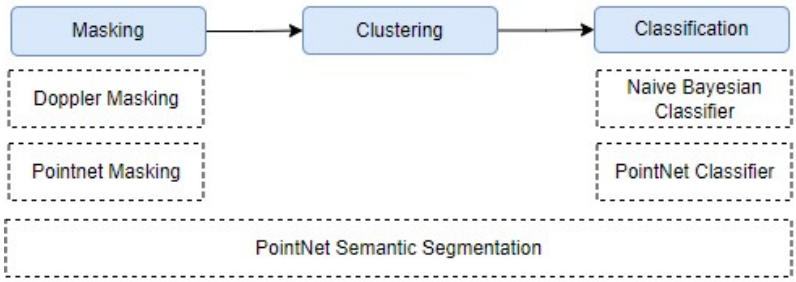


Figure 5.1: Two-Stage Network in blue with comparisons in white.

5.1 Emergence of Task Decoupling: Addressing Classification Challenges

The origin of the idea to decouple the SemSeg task emerges in Chapter 4, where we employ SemSeg networks. As mentioned, these networks experience performance is-

sues related to distinguishing between different classes due to the unnuanced nature of radar point clouds. Interestingly, the distinction between background and object is executed rather well. This leads us to consider focusing exclusively on the masking task, simplifying the challenge at hand. To validate our approach, we conduct a comparison using three different networks, each trained as multi and binary SemSeg models. Our objective is to evaluate their capabilities to differentiate only between the environment and the objects. As depicted in Fig. 5.2, the IoU for objects of each model is revealing that reducing task complexity by focusing solely on the masking problem leads to a noticeable increase in extracting objects.

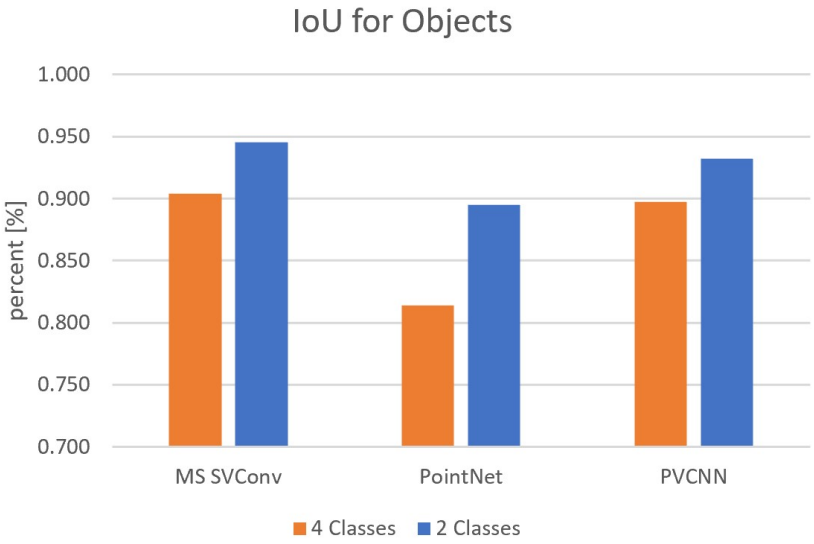


Figure 5.2: A comparison of IoU for objects between models trained on 4 classes in orange and models trained exclusively on binary classes in blue.

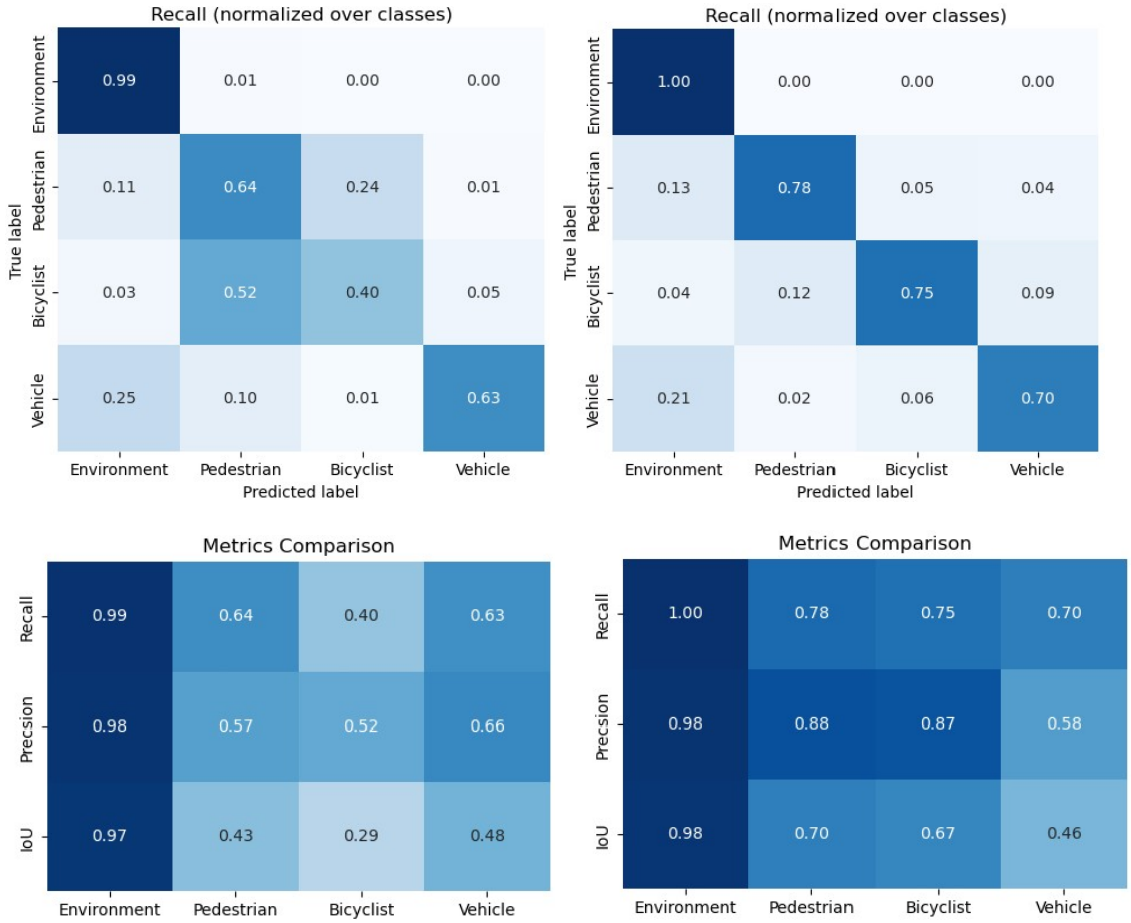
The inspiration for the TSN is stimulated by the paper [13], which achieved classification accuracy between 80-90% on radar point clouds with PointNet and even higher scores with PoinNet++ and their Radar-Transformer. This distinct higher accuracy compared to our SemSeg results sparked the concept of utilizing a two-stage approach to enhance the performance. Additionally, the decoupling of the SemSeg task into separate masking and classification tasks not only has the potential to improve performance but also facilitates a convenient integration into the perception pipeline of Sensrad AB.

5.2 Comparison: PointNet vs Two-Stage Network

In this analysis, two network architectures are compared: the aforementioned TSN implemented with PointNet versus PointNet directly as a SemSeg network. The comparison is visualized through normalized confusion matrices in Fig. 5.3, which contrast the predicted labels with ground truth labels for each point in the point cloud. The color plot below the confusion matrix provides common performance metrics for segmentation tasks. These include the Intersection over Union (IoU),

precision, and recall, whereas the recall can also be found along the diagonal of the confusion matrix.

Interestingly, the TSN resolves one of the initial issues associated with the SemSeg network visible in Fig. 5.3 - its high uncertainty in differentiating between pedestrians and bicyclists. Furthermore, by comparing the diagonal elements, the TSN increases the score across all classes. On one hand, the improved masking performance is visible in the raised 'Environment' class score. On the other hand, the scores for all object classes are increased and equally distributed. Leading to an increased recall score for all classes. In terms of precision and IoU, the TSN demonstrates a significant increase in values for pedestrians and bicyclists, however, a decrease in the IoU and the precision for the vehicle class is observed. This outcome stems from the class imbalance in the validation dataset, where cars are few in number. Consequently, even a minor percentage of incorrect pedestrian and bicycle predictions as cars results in a substantial reduction in the precision score for the car class, underscoring the effects of the class imbalance. Nevertheless, considering mIoU, which takes into account both precision and recall, the score has increased from mIoU=0.54 to mIoU = 0.7 indicating an increased overall performance.



(a) SemSeg with PointNet (mIoU = 0.54).

(b) SemSeg with TSN (mIoU = 0.7).

Figure 5.3: A comparison of SemSeg performance between PointNet and TSN on the same validation dataset.

In view of computational efficiency, the TSN falls short contrasted to the direct SemSeg approach. As shown in Tab.5.1, the TSN, which factors in DBSCAN, is four times slower than the end-to-end SemSeg network on the GPU, NVIDIA RTX 3090, measured in frames per second (fps). Therefore, while the TSN presents a solution to the prediction issue by promoting balanced class representation, its slower processing speed is a notable drawback, particularly for real-time applications or those with high computational efficiency requirements. Yet, the following section proposes a partial solution through additional flexibility to encounter the computational challenge.

Table 5.1: Inference speed of PointNet (SemSeg) and TSN on GPU (NVIDIA RTX 3090).

Neural network	**Inference speed ↑**
PointNet	**422 fps**
TSN	102 fps

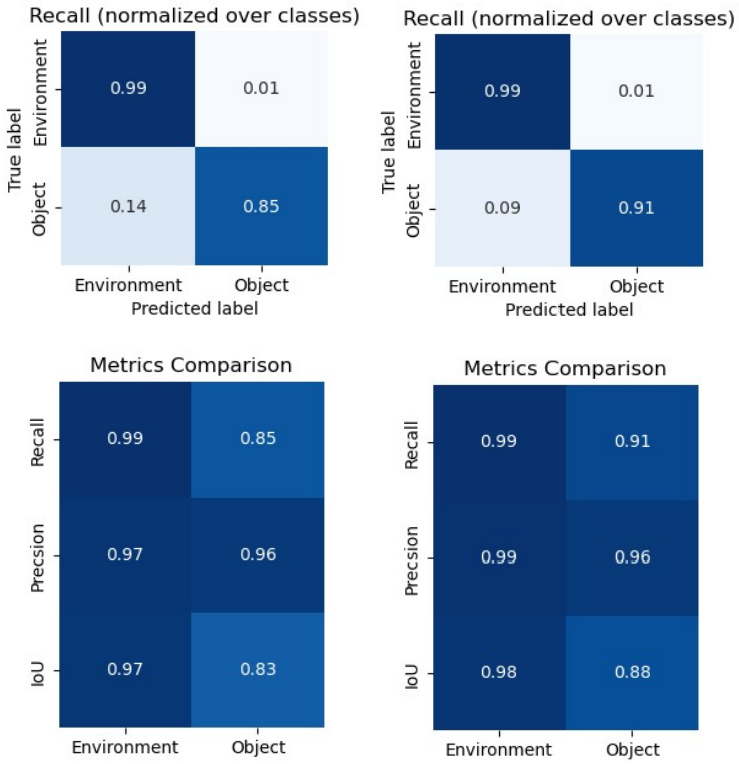
5.3 Comparison: Classical vs Neural Network Methods

In this section, we draw a comparison between the classical modules used in the perception pipeline of Sensrad AB with their neural network counterparts. It should be emphasized that all the modules are exchangeable in the TSN, making it customizable for task-specific requirements regarding accuracy and computational efficiency.

5.3.1 Doppler masking vs PointNet masking

The first module in the TSN distinguishes object points from environment points, known as object masking. This can be achieved with a classical approach, e.g., Doppler masking, a method utilized in the perception pipeline of Sensrad AB, which predicts a point as an object point if its Doppler value exceeds a predefined threshold. As an alternative, to the classical algorithms NNs can also be applied. Thus, in this section, the performance of the aforementioned classical and neural approaches is compared regarding prediction accuracy and inference speed.

The normalized confusion matrices of Doppler masking and PointNet masking on the point level are presented in Fig. 5.4. Observing the matrices, it is evident that PointNet masking exhibits superior performance regarding all values for all classes, including precision and recall and thus also IoU. However, concerning the simplicity of the Doppler masking algorithm, it performs remarkably well achieving $pre_{obj} = 0.96$. However, missing some of the object points reduces the reduced recall score to $rec_{obj} = 0.85$. This is due to the fact that points of static or slow-moving objects get not detected and will be predicted as environmental points.



(a) Doppler masking. (mIoU = 0.82)

(b) PointNet masking. (mIoU = 0.92)

Figure 5.4: A comparison of masking performance between classical and deep learning approach on the same validation dataset.

Nevertheless, considering computational efficiency, Doppler masking outperforms PointNet by an exceptional margin. As shown in Tab.5.2, Doppler masking is almost 300 times faster than PointNet masking. Meanwhile, it has to be emphasized that the Doppler masking is tested on a CPU, AMD Ryzen 9 5950X, whereas the PointNet masking is tested on a powerful GPU, NVIDIA RTX 3090. This significant difference in speed and resource requirements is a critical factor to consider, especially in applications with real-time performance and limited computational resources. However, one must bear in mind, that the fast and light Doppler masking comes with a caveat - Doppler masking is unable to detect static objects before entering the scene or never move.

Table 5.2: Inference speed of Doppler masking on CPU (AMD Ryzen 9 5950X) and PointNet masking on GPU (NVIDIA RTX 3090) in descending order.

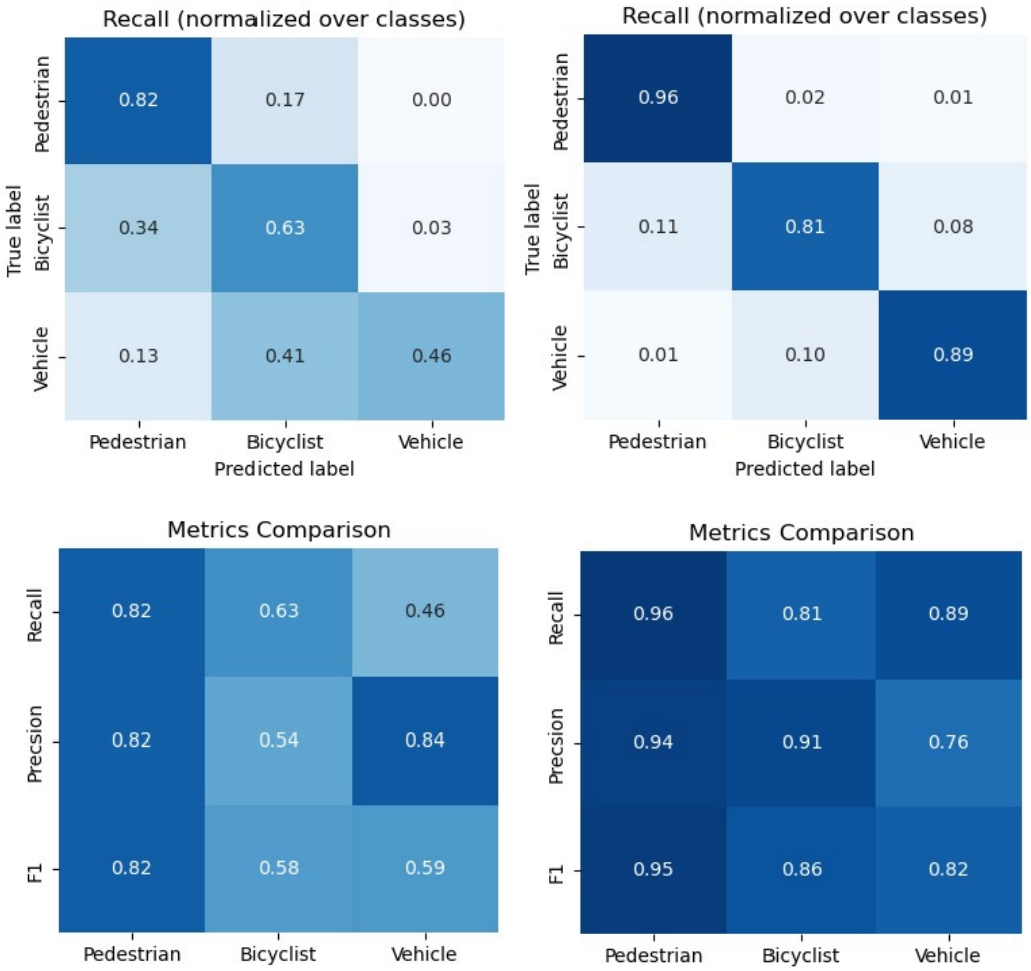
Masking method	**Inference speed ↑**
Doppler masking	**120,000 fps**
PointNet masking	408 fps

5.3.2 Naive Bayesian classifier vs PointNet classifier

The second module comparison within the TSN is between classical and neural classifiers. The classical approach investigated is the Naive Bayesian classifier (NBC), a statistical classifier based on the Bayes theorem. The neural network for comparison is a PointNet classification network. This section also compares the two aforementioned methods on prediction accuracy and speed in the same structure as before.

Figure 5.5 presents the inference results of the NBC and PointNet classifier through normalized confusion matrices. Both using the classification dataset v1, the class prediction and labels are per cluster. Right below are the precision, recall, and F1 score for the methods. Observing the confusion matrix reveals that the neural architecture-based approach yields higher evenly distributed recall scores for all three classes with $R_{PN}^{obj} = 81 - 96\%$. This validates the published results of the aforementioned paper [13] evaluating the PointNet classifier on radar data.

Observing the 'Metric Comparison' in Fig. 5.5, NBC has lower scores except for the vehicle precision $P_{NBC}^{veh} = 0.84$. This is a result of the conservative vehicle prediction of the NBC, where NBC only predicts 46% of all vehicles as vehicles, however, of the few predicted, 84% are correct, leading to the aforementioned high precision score. This is in contrast to the PointNet classifier which is able to predict approximately 90% of all vehicles correctly, however, it includes also some other classes leading to a reduced precision score of $P_{PN}^{veh} = 0.76$. Nevertheless, in view of the F1 score, which averages precision and recall, PointNet reaches higher scores than NBC, including the vehicle class.



(a) Naive Bayesian classifier with $F1_{NBC} \approx 0.63$.

(b) PointNet classifier with $F1_{PN} \approx 0.86$.

Figure 5.5: A comparison of classification performance between classical and deep learning approach on the same validation dataset.

Regarding computational efficiency in Tab.5.3, the NBC runs 8 times faster on our CPU than the PointNet classifier on our powerful GPU.

Table 5.3: Inference speed of Naive Bayesian classifier on CPU (AMD Ryzen 9 5950X) and PointNet classifier on GPU (NVIDIA RTX 3090) in descending order.

Classification method	**Inference speed ↑**
Naive Bayesian classifier	**2,600 fps**
PointNet classifier	323 fps

5.4 Object Classification: Performance Analysis

It is noteworthy to highlight the difference in classification performance between the TSN and the PointNet classifier, as illustrated in Fig. 5.6, where both confu-

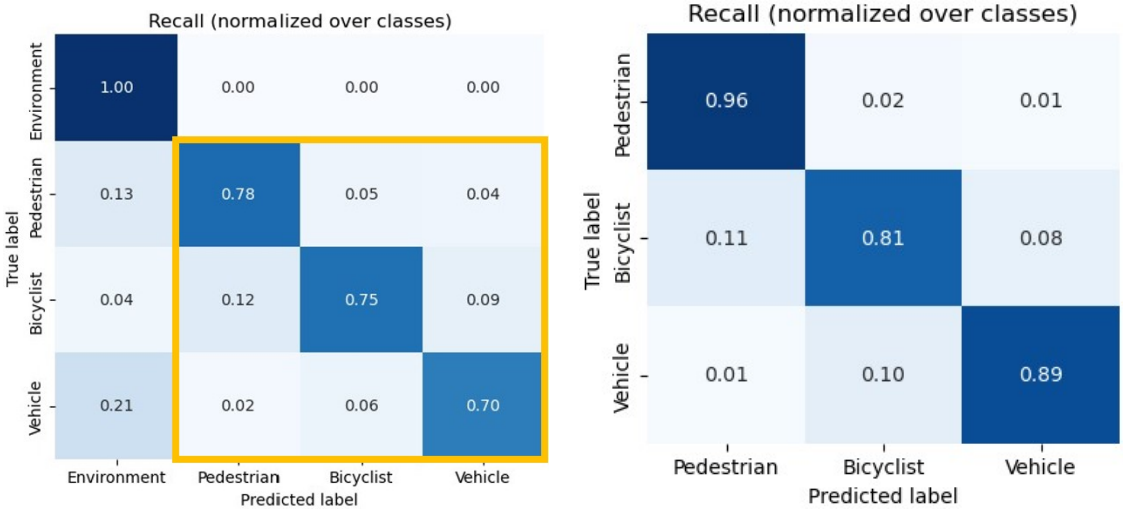
sion matrices are normalized over the classes, meaning that they are comparable even though one operates on point level and another one operates on cluster level. PointNet's higher object recall, compared to TSN (highlighted in yellow), can be attributed to three reasons: error propagation using two stages, increased problem complexity, and divergence in input data.

Error Propagation: Inevitably, the masking process introduces some degree of error, which then permeates into the classification stage.

Increased problem complexity: The classification process in TSN, as opposed to dealing with three classes, also includes an additional 'noise' class to label incorrectly masked clusters. This increases the problem complexity by incorporating an extra class.

Data Divergence: The PointNet classification network operates on ideal clusters from dataset v1, while the classifier embedded in the TSN pipeline draws its clusters from the DBSCAN module. This corresponds to dataset v2, which presents a reduced quality of cluster shapes. Further examination of the training data's influence is provided in Appendix C.1.4.

Nevertheless, as discussed in section 5.2, the concept of TSN which lowers the problem complexity, leads to a notable improvement in mIoU compared to applying the segmentation task directly.



(a) Confusion matrix of TSN classifier normalized over points with $rec^{obj}_{mean} = 0.74$.

(b) Confusion matrix of PointNet classifier normalized over clusters with $rec_{mean} = 0.89$.

Figure 5.6: A comparison of classification performance between (a) PointNet in TSN pipeline given v2 data (marked in yellow) and (b) PoinNet classifier given v1 data.

5.5 Discussion

Through the experiments and comparisons, we see that the TSN appears to fulfill its intended purpose, notably enhancing overall accuracy, especially in distinguishing between ambiguous classes. However, it is crucial to note that the TSN serves as a conceptual framework for modular architecture, incorporating both classical and NN approaches. Its performance could potentially be further improved by integrating more powerful networks such as PPNet for masking and/or classification. Additionally, as will be discussed in the forthcoming chapter, the classification accuracy of the TSN can be further improved by leveraging sequences of point clouds using a tracker and a sequential classifier. This highlights the flexibility and potential for enhancement inherent in the TSN's modular design.

When comparing classical and neural network methods, it becomes evident that NN approaches, such as PointNet, demonstrate superior accuracy and abilities compared to the corresponding classical methods such as Doppler masking and NBC. Compared to the classical method, the masking network has the ability to detect static objects and the classification network has a relatively balanced classification distribution, which validates the capability of NNs to learn complex patterns and their robustness in handling diverse data. However, this superior performance comes with a significant caveat - a relatively high computational cost. Therefore, the choice between classical and NN modules depends heavily on the specific requirements and constraints of the application in question.

6
Sequential Classifier

The next progression in enhancing classification performance is utilizing the sequences of clusters present in a track, where aggregated information from the past can aid in the current classification task. The benefits of employing sequences are two-fold:

First, the quality of observations is not consistent across all frames. There may be sporadic instances where objects are partially obscured or the measurement resolution is low. Alternatively, there might be instances where the object itself might not embody the typical class characteristics from a radar's perspective. By referencing past information about an object, we can make better-informed decisions when confronted with such situations of less informative instances.

Second, the way an object's form evolves also delivers valuable information. For instance, while cars maintain a solid form, humans and animals do not. The manner in which an object's shape changes over time can yield critical information that can only be discerned through sequential data over time.

Various strategies exist for capitalizing on sequences in point clouds, such as [31] and [32], who propose ways of accumulating points over time in diverse ways, while [33] suggests amassing information over time within the feature space.

In this chapter, we delve deeper into the potentialities of PointNet and explore how we can transform it into a sequential model to further enhance classification performance. We investigate, Feed-forward Neural Networks (FNNs), Convolutional Neural Networks (CNNs), and Long Short-Term Memory (LSTM) approaches. The use of sequential classifiers is only made possible in combination with a multi-object tracking module. The outcomes of the multi-frame classifiers are juxtaposed against the single-frame classifier discussed in subsection 5.3.2.

6.1 Sequential Architectures

The sequential NN structure that has been built consists of a feature extractor, a memory module, and a head. The feature extractor translates information from the point cloud level into a significant high-dimensional feature space. We believe it is logical to aggregate information within this space as it represents the network's own compressed data representation, from which it starts to remap the information

based on the immediate task - in our case, object classification. The features of the same track will be saved in the memory and retrieved when classifying the object of this track. In our thesis, three different heads are tested FNNs, CNNs, and LSTM to make class predictions given the stacked feature space. In this section, the feature extractor, memory module, and different heads will be explained.

Feature extractor, is a PointNet encoder, which converts a cluster into feature space. It consists of input transforms (T-Net), shared MLPs, and a max pooling layer, all of which have been introduced in subsection 3.3.1, and finally delivers a global feature vector of dimension $m = 512$ as depicted in Fig. 6.1.

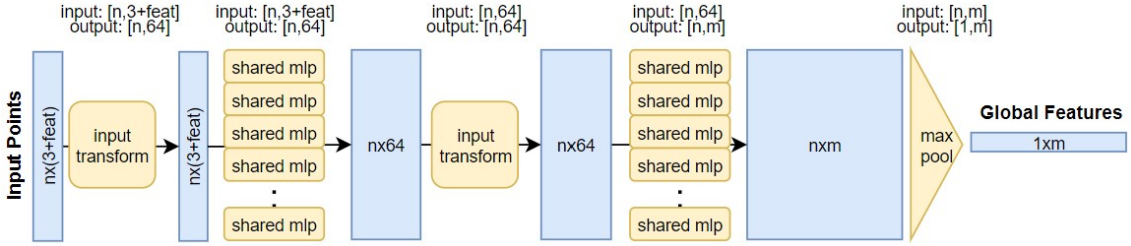


Figure 6.1: PointNet encoder architecture.

Memory module is simply a map to save the feature spaces of previously processed clusters within a track. For CNN and FNN heads, these features are subsequently stacked, generating a memory of dimension $\mathbb{R}^{T\times m}$ within the feature space for a single track, where $T = 50$ signifies the count of memorized time instances. A 'First In First Out' (FIFO) queue is then implemented, enabling the most recent information to progressively replace the oldest once a memory size of T has been completely formed. In cases where a track is newly established, the initial feature vector is replicated T times to ensure a consistent input size to the head. However, for the LSTM head, a feature vector will first be converted into a hidden state h_t and a cell state c_t for time instance t, both of dimension $\mathbb{R}^n$, where $n = 3$ represents the number of classes. Then instead of saving this feature space, only h_t and c_t will be saved.

CNN head was the first head we developed to map a temporal feature space into class probabilities. As illustrated in Fig. 6.2, we set the convolutional layer's kernel size to $k = [10, 512]$ in this experiment, indicating that the kernel operates on ten consecutive time instances concurrently. This approach allows it to detect variations in an object's feature space while maintaining complete liberty in the feature dimension m. One limitation, however, is that the $s = [10, 512]$ input values will be transformed into a singular output value. To avoid diminishing the information space too rapidly through this operation, a high channel size C_o is necessary. In the context of our experiment, we have designated the number of channels, C_o, to be 256. Using the formulas in 3.3 results in an output kernel that gets flattened to a one-dimensional tensor of size 10496. This size is manageable for subsequent fully connected (FC) layers F_i, F_h, and F_o to process and generate the final class predictions. A table of all the parameters for the CNN head can be found in Appendix Tab.D.2. As a result, this architecture together with the feature extractor leads to

a total number of about 1.2×10^7 model parameters and a computational demand of around 6.5×10^7 FLoating-point Operations (FLOPs).

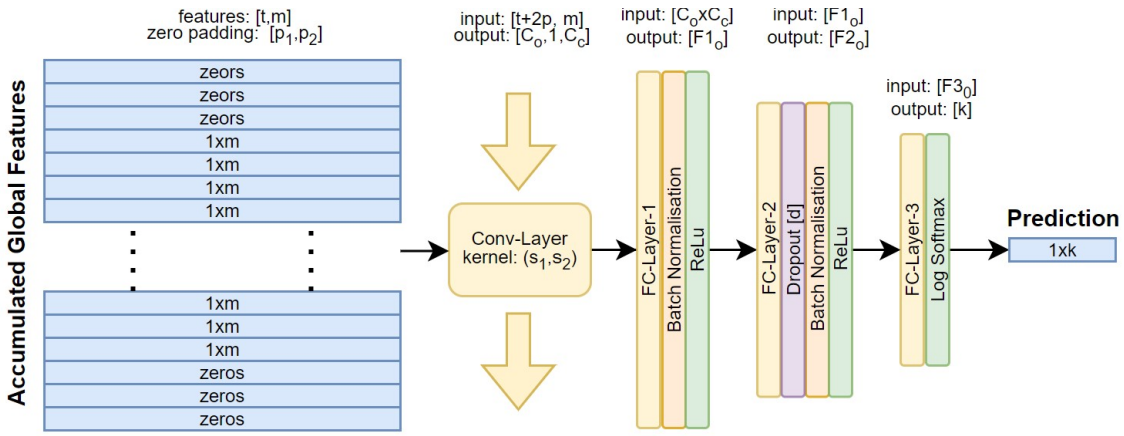


Figure 6.2: CNN head architecture.

FNN head only involves fully connected (FC) layers as shown in Fig. 6.3. Considering that a convolutional layer, employed in the previous approach, is fundamentally a constrained version of an FC layer, the goal was to allow the neural network more freedom to independently identify meaningful relations within the stacked feature space. As such, in this method, the convolutional kernel was substituted with an FC layer, and the stacked feature spaces were flattened to be fed directly. A drawback of this method compared to the CNN head is a rather high number of model parameters approximately 2.7×10^7 and consequently a reduction in computational efficiency with a demand of nearly 2.7×10^7 FLOPs. Details about the parameters can also be found in Appendix Tab.D.3.

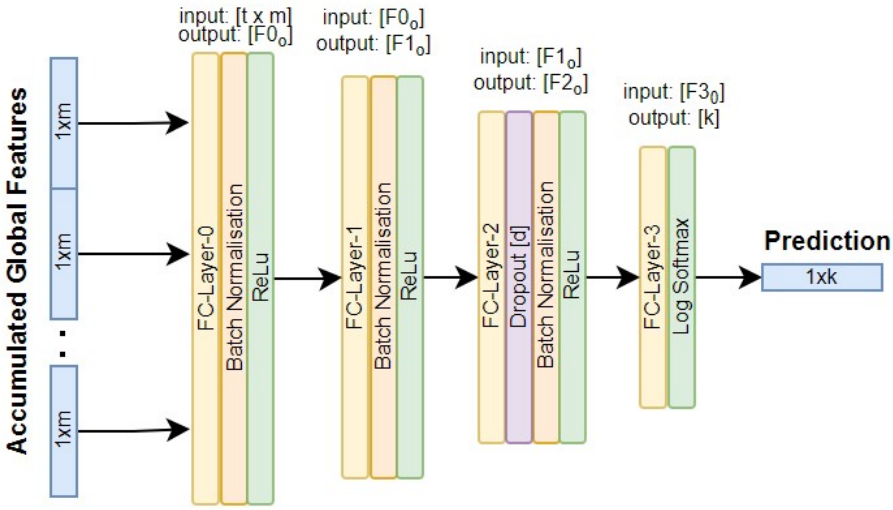


Figure 6.3: FNN head architecture.

LSTM head, for Long-, Short-Term Memory, is a more sophisticated architecture

that inhibits long- and short-term memory. Our LSTM head is constructed with a single LSTM unit $l = 1$ which transforms the feature space into the class probabilities. A major benefit of LSTM compared to the other two heads is its capacity to handle extended memorized time instances T without an increase in model size. The LSTM is the most optimized architecture, with over 30 different configurations tested using various feature sizes and FC layers, both preceding and following the LSTM layers, as well as amplifying the number of LSTM layers. However, none of these configurations surpassed the others, and most converged to the same F1 score. Thus, we utilize the most simplistic configuration capable of converging to the typical convergence range. This setup is achieved by steadily simplifying the model complexity until it could no longer overfit the training data while still maintaining the validation score. Despite its simplicity, this model demonstrates more robust training and validation behavior than its more complex LSTM counterparts. Lastly, this head has a total number of 6×10^3 model parameters and a very low FLOPs requirement of 6×10^3. Parameter details can also be found in Appendix Tab.D.4.

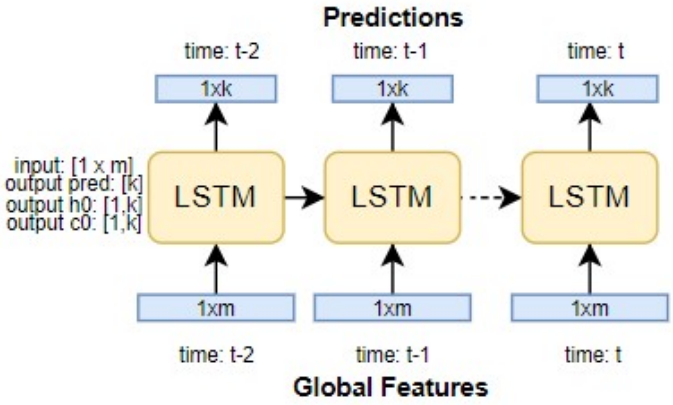


Figure 6.4: LSTM head architecture.

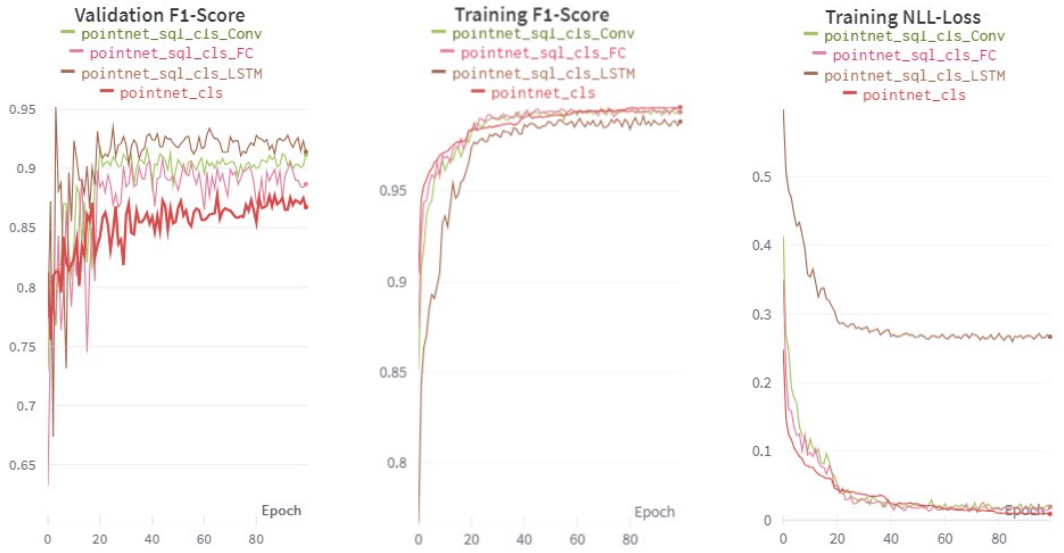
A more detailed comparison of different modules in a sequential classifier and with the PointNet classifier in terms of the number of parameters, FLOPs, and inference speed will come later in section 6.3.

6.2 Accuracy Evaluation

After training all classifiers for 50 epochs, Fig. 6.5a presents the relatively high validation scores of our three sequential classifiers - CNN-based (in green), FNN-based (in pink), and LSTM-based (in brown) - and the PointNet classifier single frame (in red), which serves as a benchmark for comparison. Although the network performance initially appears somewhat indeterminate, a pattern emerges wherein the LSTM-based classifier leads with an $F1_{LSTM}$ score of approximately 0.92 after 20 epochs. It is closely followed by the CNN with $F1_{CNN} \approx 0.90$, the FNN with $F1_{FNN} \approx 0.88$, and lastly the original single frame PointNet scoring $F1_{PN} \approx 0.86$. In reference to Fig. 6.5b and 6.5c, it can be observed that the LSTM-based classifier

is the sole model which does not fully overfit to the training dataset and therefore generalizes better on the validation dataset.

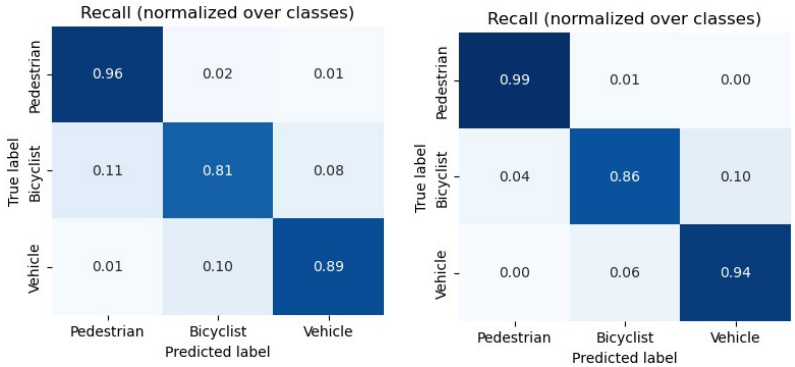
The outcome of the LSTM can be attributed to two factors. On one hand, in contrast to the CNN- and FNN-based models, which require a certain level of model complexity to reduce a large feature aggregation down to the number of classes, the LSTM-based model's complexity is independent of the number of features aggregated and hence can be built with low complexity while still having a reasonable time horizon. On the other hand, seeing greater potential in the LSTM-based network lead to a more judiciously crafted architecture. Therefore the simpler more optimized LSTM architecture generalizes better the training data and thus achieves a higher validation score.

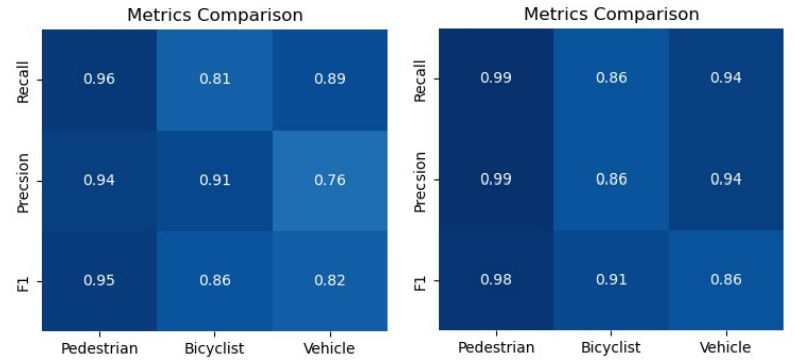


(a) Validation F1 score. **(b)** Training F1 score. **(c)** Training negative log-likelihood loss.

Figure 6.5: Training and validation performance of different classifiers.

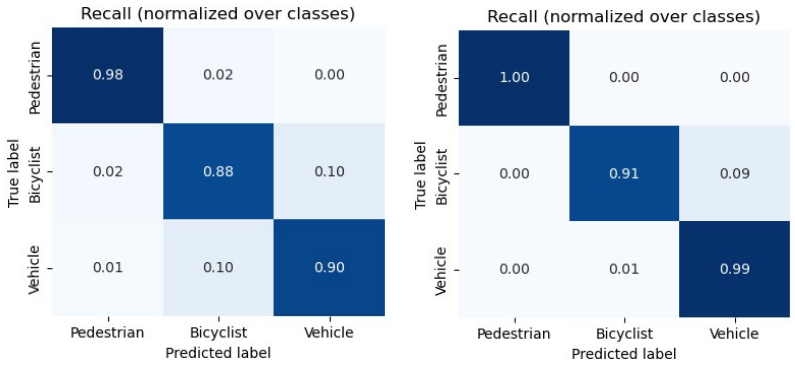
To evaluate the performance of the architectures in more detail, the confusion matrices of every classifier normalized over the class i.e., recall, are shown in Fig. 6.6. Overall, on the diagonal is visible that all classifiers have a high recall score, showing that the models are able to predict the objects' classes in a balanced manner. In the last row of the 'Metrics Comparison' the class-wise F1-score is visible which is an average of precision and recall scores of a certain class. Here is visible that the models perform the best on the pedestrian class $F1_{ped} = 95 - 100\%$, while the models have a slightly reduced score for the vehicle class with $F1_{bic} = 83 - 89\%$. Calculating the metric of interest, the mean F1-score shows that the sequential information indeed increased the performance from PointNet single frame of $F1_{PN} \approx 0.86$ to the three sequential versions FNN-based classifier $F1_{FNN} \approx 0.88$, CNN-based classifier $F1_{CNN} \approx 0.90$ and the LSTM-based classifier $F1_{LSTM} \approx 0.92$ as already mentioned above.

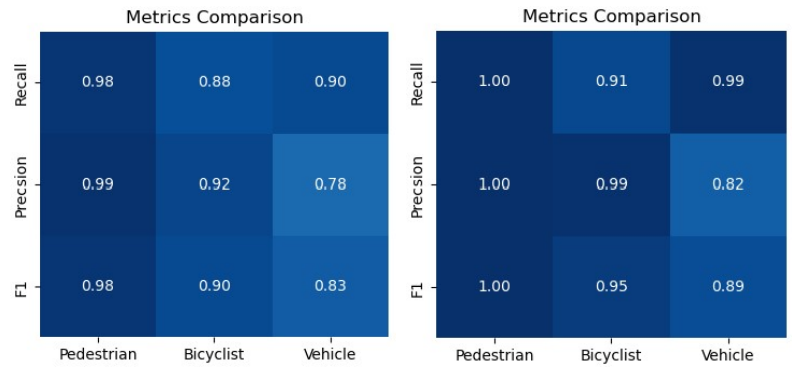




(a) PointNet classifier with $F1_{PN} \approx 0.86$.

(b) CNN-based classifier with $F1_{CNN} \approx 0.90$.





(c) FNN-based classifier with $F1_{FNN} \approx 0.88$.

(d) LSTM-based classifier with $F1_{LSTM} \approx 0.92$.

Figure 6.6: Normalized confusion matrix in the first row and Precision, recall, and F1 score per class in the second row for single frame PointNet classifier and the sequential CNN-, FNN-, and LSTM-based approaches.

6.3 Resource Evaluation

As a straightforward comparison among all the introduced heads as well as a demonstration of the feature extractor, three essential characteristics for real-world applications will again be compared: model size, inference speed, and training time. In this section, they will be demonstrated in order.

Model size, i.e., the number of parameters in a model, of every head and the PointNet encoder is listed in Tab.6.1. Observably, the LSTM head stands out as the smallest head with an incredibly small number of parameters. Another interesting point to consider is the ratio of model size between the encoder and head components. While CNN and FNN heads are much larger than the PointNet encoder, the LSTM head is 441 times smaller than the PointNet encoder, making it extremely memory efficient.

Table 6.1: Number of parameters of different modules in a sequential classifier in ascending order.

Module	No. parameters ↓
PointNet encoder	2,736,841
LSTM head	**6,204**
CNN head	12,214,787
FNN head	26,875,907

Considering all modules as a whole, there are three sequential classifiers in total with the same feature extractor but different heads. A comparison among different sequential classifiers and against the PointNet classifier for a single frame regarding the number of parameters is also made in Tab.6.2. It is exciting that the LSTM-based classifier is not only the smallest one among all sequential classifiers but also smaller than the original PointNet classifier, whereas FNN- and CNN-based classifiers are much larger than the PointNet classifier.

Table 6.2: Number of parameters of different classifiers in ascending order.

Neural network	No. parameters ↓
PointNet encoder + LSTM head	**2,743,045**
PointNet classifier	3,462,348
PointNet encoder + CNN head	14,951,628
PointNet encoder + FNN head	29,612,748

Inference speed can be measured both in theory and in practice, corresponding to FLOPs and actual inference speed on hardware respectively. As illustrated in Tab.6.3, the LSTM head is the fastest component, way better than all others. Interestingly, the PointNet encoder is now the main bottleneck compared to all heads. While the computational requirements for the encoder and head are on the same

level for the CNN- and FNN-based classifiers, the LSTM-based classifier is rather unbalanced by allocating most of its computational power to the encoder module.

Table 6.3: FLOPs and inference speed of different modules in a sequential classifier on GPU (NVIDIA RTX 3090) in descending order of inference speed.

Module	**FLOPs ↓**	**Inference speed ↑**
LSTM head	**6,228**	**11,508 fps**
FNN head	26,877,696	2,825 fps
CNN head	64,665,472	2,515 fps
PointNet encoder	97,001,728	686 fps

Again, the comparison of every classifier is also listed in Tab.6.4. Apparently, the PointNet encoder combined with the LSTM head outperforms all other classifiers including the original PointNet classifier in both FLOPs and inference speed by a good margin. Two reasons lead to this result: the smaller feature space of dimension 512 from the PointNet encoder used in sequential classifiers compared to the one in the PointNet classifier of dimension 1024, and the simple structure of the LSTM head. Note that the CNN- and FNN-based sequential classifiers also have a smaller feature space, but they are slower than the PointNet classifier. Hence, a simple head architecture is rather important to achieve a fast inference speed.

Table 6.4: FLOPs and inference speed of different classifiers on GPU (NVIDIA RTX 3090) in descending order of inference speed.

Neural network	**FLOPs ↓**	**Inference speed ↑**
PointNet encoder + LSTM head	**97,007,956**	**318.21 fps**
PointNet classifier	114,962,432	239.31 fps
PointNet encoder + CNN head	161,667,200	205.25 fps
PointNet encoder + FNN head	123,879,424	200.51 fps

Training time for each classifier is compared in Tab.6.5, where the PointNet classifier is leading the board with less than half an hour, while all sequential classifiers are quite close with nearly one and half hours. Nevertheless, even the longest training time is still very durable. So this comparison will not have any real impact on the decision of selecting a model but merely give an impression of the approximate time.

Table 6.5: Training time for different classifiers on GPU (NVIDIA RTX 3090) in ascending order.

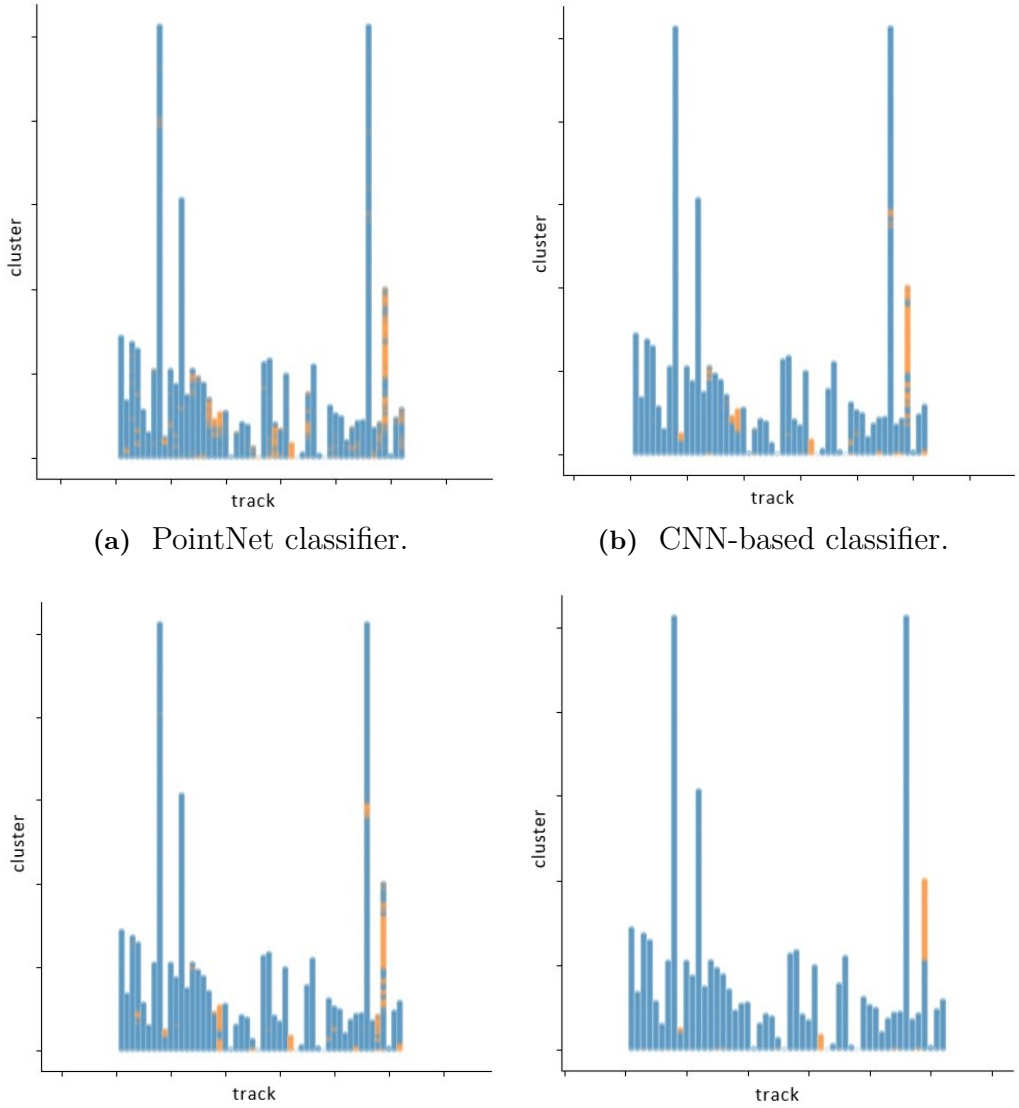
Neural network	**Training time ↓**
PointNet classifier	**27m35s**
PointNet encoder + CNN head	1h22m4s
PointNet encoder + LSTM head	1h25m32s
PointNet encoder + FNN head	1h29m36s

6.4 Performance Analysis

In this section, the characteristics of the classifiers are further investigated in regard to the distribution of misclassifications within tracks, memory confidence versus measurement trust, and an LSTM state evaluation to get more insight into the inner workings of the LSTM model.

Distribution of misclassifications within tracks for each classifier is provided in Fig. 6.7, where the x-axis represents 52 tracks and the y-axis represents the stacked predictions for every track with correct predictions marked in blue and wrong marked in orange. Although all networks demonstrate impressive results, a closer examination of the tracks reveals some differences. The PointNet classifier tends to have the highest rate of misclassifications overall, as well as the highest rate of within-track misclassifications, as evidenced by the orange-spotted track pattern. The two simple sequential networks, namely the CNN- and FNN-based classifiers, also display a degree of variance, as shown by sporadic color changes and slightly orange-tinged areas within a track. This color fluctuation within a track suggests model uncertainty as the prediction class shifts. In contrast, the LSTM-based classifier exhibits a more consistent behavior in prediction, indicated by the clear demarcation between orange and blue. Although it initially misclassifies a few clusters, it largely corrects itself subsequently.

However, the LSTM-based classifier in Fig. 6.7d shows two larger orange track splines, with which also the other networks struggle. Those two misprediction arises due to the ambiguous character of those two cases: one represents a bicycle moving at an unusually slow pace behind a pedestrian predicted as a pedestrian, and the other, a high-speed motorcycle labeled as a bicyclist and predicted as a vehicle. The misclassifications for these exceptions are understandable given their unique circumstances. The first case involves unusual dynamics, while the second presents a gray area as to whether a motorcycle should be classified as a vehicle, as predicted, or a bicyclist, as labeled.



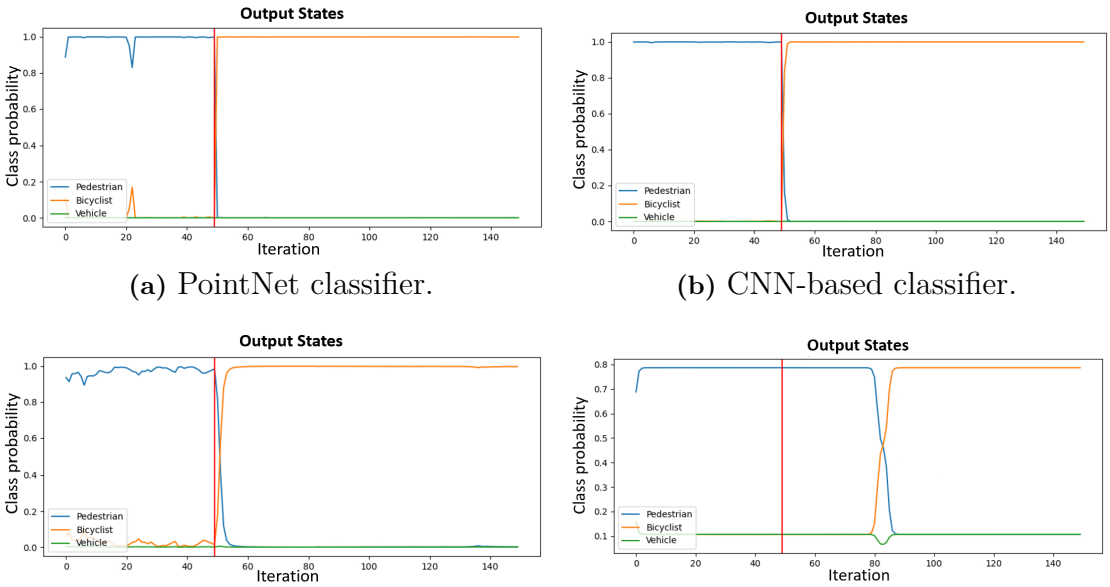
(a) PointNet classifier. **(b)** CNN-based classifier.

(c) FNN-based classifier. **(d)** LSTM-based classifier.

Figure 6.7: Prediction distribution over 52 test tracks for each classifier, where the x-axis represents different tracks, the y-axis represents the stacked predictions for every track with correct predictions marked in blue and wrong marked in orange.

Memory confidence versus measurement trust of each classifier is investigated in two tests. The first test, memory trust, evaluates how quickly the models adapt to the new measurement information. This is done by feeding 50 consecutive samples of a pedestrian track and then suddenly switching to a bicyclist sequence. In the ideal case, the models switch quickly to the new bicyclist class and show their agility. The second test, memory confidence, evaluates the memory capabilities of the models. This is done by switching after the 50 consecutive pedestrian clusters to random noise. This means that no new information will be in the measurement and hence the model should make use of the memory capabilities by holding on to the last seen class, pedestrian.

The results of the measurement trust scenario are depicted in Fig. 6.8, where the x-axis corresponds to the sequence of clusters, the y-axis corresponds to the predicted class probabilities, and the red vertical line indicates the switching point from the old to the new class. As mentioned, the optimal response would be a rather quick adjustment from pedestrian to the new bicyclist class. Observing the class probabilities in general, it can be said, that except for some initial uncertainties as denoted by the wavering line for the FNN-based classifier and a minor hiccup for the PointNet classifiers, the networks exhibit a high degree of confidence in their predictions. Regarding the class switch, as expected, the PointNet classifier performs the switch instantly, owing to its lack of memory. Interestingly, the CNN- and FNN-based classifiers, execute the switch relatively swiftly in just 2 and 5 steps respectively. This stands in stark contrast to the LSTM-based classifier, which necessitates 32 steps for the label transition, indicating its influence on prior measurements.



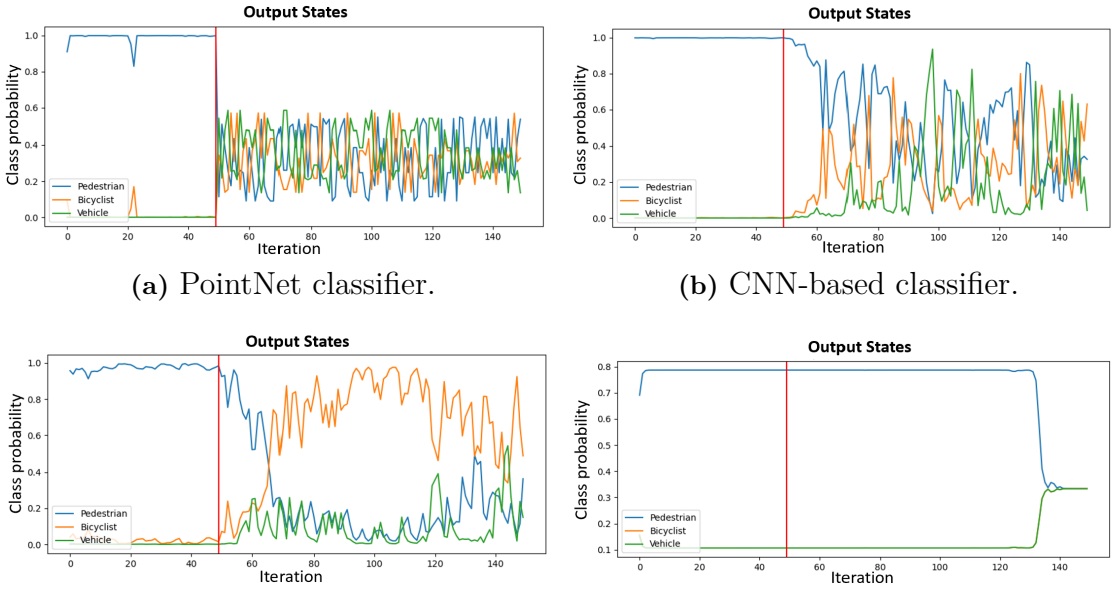
(a) PointNet classifier. **(b)** CNN-based classifier.

(c) FNN-based classifier. **(d)** LSTM-based classifier.

Figure 6.8: Predicted class probabilities for the track of 50 pedestrian samples followed by 100 bicyclist samples with the red vertical line indicating the switching point.

In the second scenario, as depicted in Fig. 6.9, the networks encounter only random noise after the red vertical line. In such a situation, it is anticipated that the networks can utilize their memory capabilities to avoid a rapid class change. It is noteworthy that the final output class is determined by the `argmax` function, which selects the label corresponding to the uppermost curve. In this experiment, the PointNet classifier immediately displays random predictions. Meanwhile, CNN- and FNN-based classifiers exhibit a noticeable decrease in their certainty, with potential curve crossover after approximately 10-12 noise samples. The LSTM-based classifier, on the other hand, switches labels after 78 samples. However, it is also worth mentioning that while CNN- and FNN-based classifiers appropriately express their

uncertainty, the LSTM-based classifier remains fairly confident in the pedestrian label, despite the fact that it is only encountering noise.



(a) PointNet classifier. **(b)** CNN-based classifier.

(c) FNN-based classifier. **(d)** LSTM-based classifier.

Figure 6.9: Predicted class probabilities for the track of 50 pedestrian samples followed by 100 noise samples with the red vertical line indicating the switching point.

LSTM states analysis investigates the LSTM unit further to gain an understanding of the inner processes when executing the two aforementioned tests. The state analysis is illustrated in Fig. 6.10 where the last row shows the already visualized predicted class probabilities. The first row represents the stacked feature space with an image plot. The short-term memory and long-term memory are visualized in rows two and three. The colors blue, orange, and green still correspond to the classes pedestrian, bicyclist, and vehicle, respectively.

First of all, when examining the feature space, a clear change in pattern is noticeable over row 50. Keep in mind that each of the 150 rows represents a different recorded cluster on the track. Interestingly, while clusters from the same class inevitably undergo shape changes from frame to frame, a recognizable class pattern still emerges in the feature space.

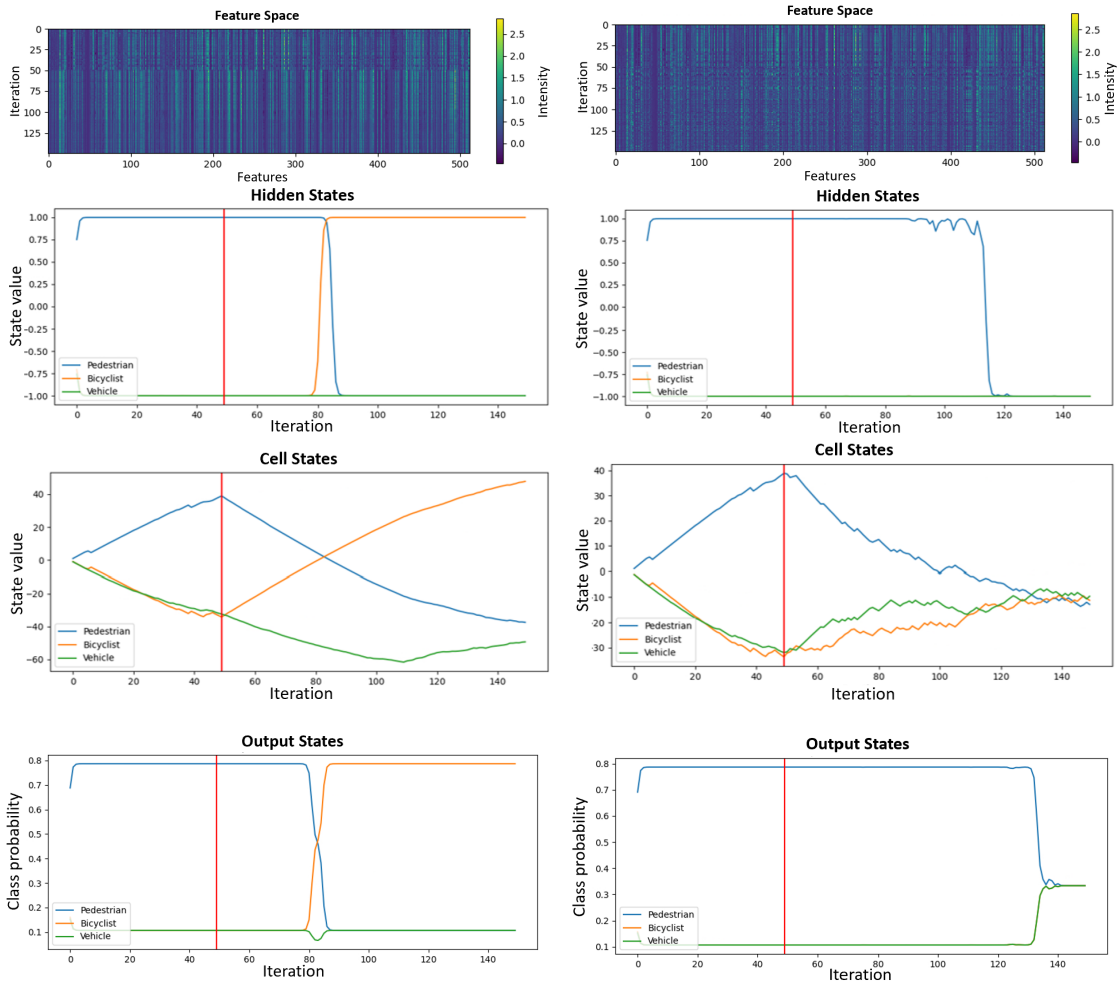
This feature space is fed to the LSTM row by row, generating the plots h_t and c_t, which represent the hidden and cell states, respectively. The cell states, acting as the long-term memory, reveal insights about the certainty of the model which were not apparent in the class probability output. In the cell states of both experiments, an immediate reaction to the class change is visible. However, the cell states behave in an integrating fashion, leading to a certain model inertia. This leads in the first experiment, after the switch, to a steady increase of the newly correct bicycle class

while the previous pedestrian class is continuously decreasing. Conversely, in the second noise experiment, all three cell states converge to the same state, indicating the model's increasing uncertainty.

In view of the two hidden state figures, in the case of our model, the hidden states, i.e. output states, represent the cell states in a binarize manner. Regarding the graphs, the hidden state $h^i = 1$ for class $i \in \{ped, bic, veh\}$ if c^i is approximately bigger than zero and $h^i = -1$ if c^i is smaller than zero. This behavior reflects the strong trust of the model in its long-term memory c_t since its hidden state, i.e. output vector, is mainly influenced by its long-term memory c_t and less by the input vector x_t generated by the measurements.

This is observed by analyzing the zero crossing of the cell states of c^i and the influence on the corresponding hidden states h^i. In the first experiment at $t \approx 80$ c_t^{bic} is increasing and crosses zero while c_t^{ped} is decreasing and crosses zero. Which leads to an immediate change of the hidden states $h_t^{bic} = 1$ $c_t^{ped} = -1$. While $h_t^{veh} = -1$ remains unchanged since $c_t^{ped} < 0$ for $\forall t$. This pattern also applies to the second experiment while all hidden states $h^i = -1$ at the point where c_t^{ped} is approximately smaller than zero.

The final output is then the class probability of the hidden states, which must sum up to one. This leads to an interesting phenomenon as visible in the second experiment. Since the model does not have confidence in any of the classes (as seen in the hidden states), the final output displays an equal probability of all three classes.



(a) Switch to another class. **(b)** Switch to random noise.

Figure 6.10: An illustration of internal workings of LSTM, where the first row represents the input data (feature space), the second and third rows show the internal states, and the last row displays the model output.

6.5 Discussion

All sequential models have surpassed the single-frame PointNet version, demonstrating a certain degree of memory capability Fig. 6.9. The LSTM version performed exceptionally well, not only in terms of accuracy but also regarding its model size and high computational speed. However, it has to be mentioned, the LSTM is the most optimized model. Enhanced tuning of the CNN and FNN models could potentially reduce the model size and, given that they overfit the training data, also likely improve validation accuracy scores. Still, it remains questionable whether the CNN and FNN could match the performance of the LSTM, which as seen in Fig. 6.10, operates quite efficiently.

Besides evaluating the models, it is also important to consider the data used. The test and validation clusters were created using the 'Sequential Classification Dataset

V1', hence the clusters are derived from annotations. However, when using the sequential network, it would be built into a two-stage architecture with a tracker, thus cluster quality would be lower. Regardless that the overall accuracy would decrease, one would anticipate that the sequential architectures would exhibit even greater advantages compared to the single-frame counterparts, as the data quality of the clusters fluctuates more across frames.

Lastly, the approach of post-processing, specifically simple label accumulation, has not been addressed in this evaluation to maintain a concise comparison. However, our tests indicate that label accumulation considerably improves the predictions and is a worthwhile method, especially since it doesn't necessitate modifications to the networks and is computationally efficient.

7

Metric and Loss Function

A metric is a ruler to evaluate the performance of a model. In the SemSeg task, the evaluation metric for a human to understand is mIoU explained in 3.7. But a metric sometimes can not reflect everything that is concerned. Thus, a modified or even new metric shall be considered if necessary. On the other hand, another ruler for the network itself is also needed, which is known as a loss function. The goal of training a neural network is actually to minimize the loss. Different loss functions shall be applied to different machine-learning tasks. For the semantic segmentation task, either Negative Log-Likelihood (NLL) loss or Cross-Entropy loss can be used. In short, these two loss functions are slightly different but interchangeable as they measure the same thing. In this chapter, the defect of IoU/mIoU will be first demonstrated, and new evaluation metrics to compensate for this defect will then be introduced. Afterward, an attempt to try new loss functions on PointNet will be explained followed by a discussion.

7.1 Defect of IoU/mIoU

Even though mIoU is a common choice of metric for the semantic segmentation task, it is not perfect for our application. E.g., as illustrated in Fig. 7.1, prediction 1 has exactly the same mIoU as prediction 2, however, the false prediction of pedestrians is different and prediction 1 is more preferred because the false prediction of pedestrians is closer to the ground truth pedestrians. In the current pipeline of Sensrad AB, all the points that are marked as an object will be fed into DBSCAN as described in 3.6, so if predicting background points as objects is not avoidable, it would be preferred to have them as close to the corresponding real objects as possible to better cluster them later on. Therefore, the defect of mIoU is the loss of distance information for wrongly predicted object points.

7.2 New Metrics

Due to the defect of mIoU explained above, some new metrics are needed to measure how badly those wrongly predicted object points are located in a point cloud. It is valuable information for us to understand how difficult it is for a clustering algorithm, e.g., DBSCAN, to correctly cluster different instances. To this end, two similar but slightly different metrics will be introduced in the rest of this chapter.

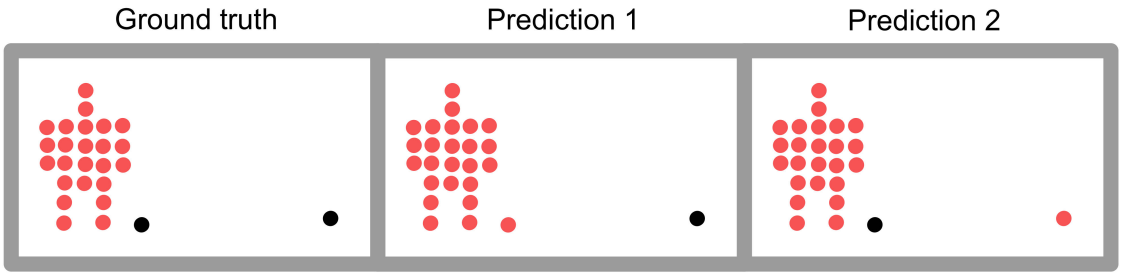


Figure 7.1: An example of ground truth and different predictions of the same point cloud, where pink represents pedestrian and black represents environment.

7.2.1 Average Euclidean Distance to the Closest Corresponding Ground Truth (AEDC)

To measure how far in Euclidean space a wrongly predicted point of some class is on average to its corresponding ground truth point of the same class, the new metric, Average Euclidean Distance to the closest Corresponding ground truth (AEDC), is invented. As the name indicates, it measures the average Euclidean distance from every point that is predicted as some object but actually background to its closest corresponding ground truth point of the same class is calculated. The condition for a point to be taken into consideration, i.e., predicted as some object class c but actually a background, is denoted as ψ as follows

$$\psi : \quad p_c = c \ \wedge \ l_c = 1 \ \wedge \ c \neq 1, \tag{7.1}$$

where p_c is the predicted class, and l_c is the ground truth label class. However, it is possible that the wrongly predicted class does not have its corresponding ground truth point in this point cloud. In this case, the maximum range of this point cloud will be taken instead. Finally, the mathematical expression for this Average Euclidean Distance to the closest Corresponding ground truth (AEDC) of a class c can be derived below

$$\text{AEDC}_c = \frac{\sum_{i=1}^{n_\psi} d(P_\psi^i, P_{l_c=c}^i)}{n_\psi}, \tag{7.2}$$

where n_ψ is the number of points that fulfill the condition ψ, P_ψ^i is the i^{th} point that fulfills the condition ψ, $P_{l_c=c}^i$ is a point with ground truth label c and the closest to P_ψ^i in Euclidean space, and $d(P_\psi^i, P_{l_c=c}^i)$ is the Euclidean distance between these two points.

With this new metric, it is easy to describe how dispersed those false object points of each class are. But to have a holistic view of all classes, it is natural to measure the mean value over all classes, i.e., mean AEDC (mAEDC), shown below

$$\text{mAEDC} = \frac{\sum_{c=2}^{N_c} \text{AEDC}_c}{N_c - 1}, \tag{7.3}$$

where N_c is the number of classes.

7.2.2 Average Euclidean Distance to the Closest Object Ground Truth (AEDO)

Though the new metric above, mAEDC, looks fairly reasonable, it makes no sense to look for the corresponding ground truth points of the same class as wrongly predicted if this metric is invented to understand how hard it is to cluster instances as the clustering algorithm does not care about class. Thus, a slightly different but more suitable metric, Average Euclidean Distance to the closest Object ground truth (AEDO), that seeks ground truth points of any object class instead of the corresponding class, is introduced. Similar to mAEDC, if no ground truth points of any object class exist in the point cloud, the maximum range value will be taken instead. The mathematical expression is structured as

$$\mathrm{AEDO}_c = \frac{\sum_{i=1}^{n_\psi} d(P_\psi^i, P_{l_c \neq 1}^i)}{n_\psi}, \tag{7.4}$$

where $P_{l_c \neq 1}^i$ is a point with a ground truth label not equal to the background, i.e., could be any object class, and the closest to P_ψ^i in Euclidean space, and all the rest notations have exactly the same meaning as in Eq.7.2.

Therefore, the mean value of AEDO over all classes, mAEDO, is organized similarly to mAEDC as follows

$$\mathrm{mAEDO} = \frac{\sum_{c=2}^{N_c} \mathrm{AEDO}_c}{N_c - 1}. \tag{7.5}$$

Note that the new metrics, both mAEDC and mAEDO, however, are not designed to replace mIoU, but to compensate for its loss of distance information.

7.3 New Loss Functions

Since new metrics have been created, it is worth trying to train neural networks with better performance on mAEDC and mAEDO. In the formulas of mAEDC in Eq.7.3 and mAEDO in Eq.7.5, the maximum range value is taken if no corresponding ground truth point exists in the point cloud, however, it would make the loss incredibly large in this case. Besides, there should be a maximum range for these metrics since DBSCAN can not cluster points together once the distance is beyond some threshold. Thus, `sigmoid` function ranging between 0 and 0.5 after being shifted downwards by 0.5 is a good choice.

The original `sigmoid` function reaches its upper limit once the variable is raised up to 10, meaning that if a distance of 10 meters is as bad as any distance longer than 10. So a scaling factor α is needed to adjust this distance threshold. Finally, the weights for each class w_c are also applied to this loss function due to the imbalance of the dataset. Hence, the mAEDC and mAEDO loss functions derived from their corresponding metric expressions are structured respectively as below

$$\text{mAEDC loss} = \frac{1}{N_c - 1} \cdot \sum_{c=2}^{N_c} \left(\frac{w_c \cdot \sum_{i=1}^{n_\psi} (\texttt{sigmoid}(\alpha \cdot d(P_\psi^i, P_{l_c=c}^i)) - 0.5)}{n_\psi} \right), \quad (7.6)$$

$$\text{mAEDO loss} = \frac{1}{N_c - 1} \cdot \sum_{c=2}^{N_c} \left(\frac{w_c \cdot \sum_{i=1}^{n_\psi} (\texttt{sigmoid}(\alpha \cdot d(P_\psi^i, P_{l_c \neq 1}^i)) - 0.5)}{n_\psi} \right), \quad (7.7)$$

where all the symbols represent the same meaning as in the previous section 7.2, and α is selected as 10, meaning that the maximum distance we care is one meter and any farther distance shall be treated as bad as one meter for DBSCAN to cluster.

The total loss function now consists of mAEDC/mAEDO loss and NLL/Cross-Entropy loss. To verify if it works, the simple and fast network, PointNet, is the one that has been experimented with for the SemSeg task. So this network is again trained three times, once with mAEDC loss, once with mAEDO loss, and once without mAEDC loss nor mAEDO loss.

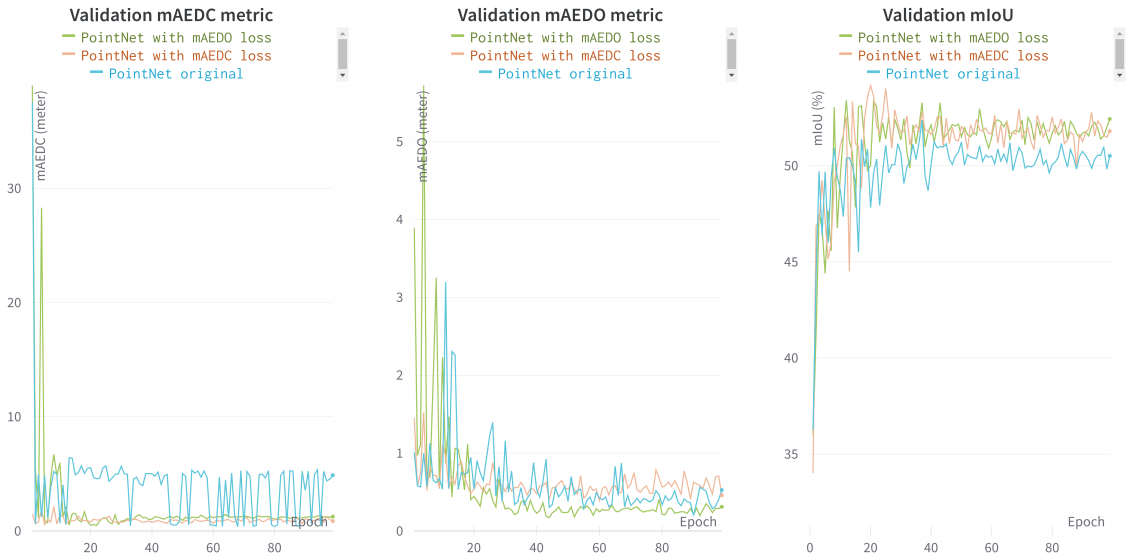
7.4 Evaluation

To evaluate the effect of the mAEDC and mAEDO loss functions, PointNet with mAEDC and NLL loss functions, PointNet with mAEDO and NLL loss functions, and the original PointNet with only NLL loss function are trained for 100 epochs, and both mAEDC and mAEDO metrics, as well as mIoU, have been calculated on the validation set after every training epoch.

The validation mAEDC metric is depicted in Fig. 7.2a, from which we can see that the original PointNet starts with a fairly large mAEDC but decreases dramatically. However, in the end, it still struggles between a low level and a high level (~ 4.8), thus never converges. In contrast, PointNet with mAEDO loss is much better and can stabilize at a very low level (~ 1.2). Lastly, PointNet with mAEDC loss is obviously the best one, stabilized at the lowest level (~ 0.8), slightly better than the one with mAEDO loss. Hence, it proves that this loss function is working in the way we expect.

The validation metric of more interest, mAEDO, is shown in Fig. 7.2b, where something similar and different can both be observed in the meantime. The same as above, PointNet with mAEDO loss performs the best, converging at (~ 0.3) with no surprise. However, the original PointNet (~ 0.5) outperforms PointNet with mAEDC loss (~ 0.6) by a tiny margin. This is a bit unexpected but still can be explained. While the mAEDC loss function prefers to push a false positive point closer to its corresponding ground truth point, it could, unfortunately, push it farther away from its closest ground truth point of any object.

On the other hand, one of the hypotheses of employing an extra loss function is that the mIoU is still on the same level since it is the most concerned metric. As illustrated in Fig. 7.2c, all three variants of PointNet are indeed on the same level. Not only that, both PointNet networks with extra loss even reach a slightly higher mIoU than the original one by $\sim 1.5\%$. But we do not encourage to over-explain this small difference due to the randomness during training even with exactly the same configuration for the same network. Nevertheless, it proves that mAEDC and mAEDO loss functions do no harm to mIoU performance.



(a) Validation mAEDC. **(b)** Validation mAEDO. **(c)** Validation mIoU.

Figure 7.2: Validation mAEDC, mAEDO, and mIoU of three PointNet variants.

7.5 Discussion

To sum up this chapter, we have proposed two metrics and corresponding loss functions: mAEDC and mAEDO. The metrics help us better understand how far those false positives are from their corresponding or any object ground truths on average. They are invented to compensate for the distance information loss of mIoU, but not to replace mIoU. On the contrary, both metrics can be considered and perhaps weighted differently for different applications.

In our tests, a model without an extra distance loss function can reach a fairly good result on the mAEDO metric, but never stabilizes on the mAEDC metric. More excitingly, both the mAEDC and mAEDO loss functions improve their corresponding metric as expected without a sacrifice of mIoU. However, while the mAEDO loss also improves in the mAEDC metric, the mAEDC loss decreases in the mAEDO metric. In conclusion, it is the best choice to implement the mAEDO loss function in a SemSeg model. Nevertheless, more experiments need to be repeated.

8

Conclusion

This master's thesis explored how neural networks can identify and classify objects in 4D radar point cloud data. Considering that our dataset has been labeled on a point-wise basis, and recognizing the limited research conducted on semantic segmentation within the 4D-imaging radar data domain, we directed our focus towards exploring semantic segmentation networks for this task.

We examined eight different neural networks for semantic segmentation on 4D radar point cloud data. Despite tendencies to overfit, perhaps because the radar data was less detailed than the point cloud data these models were initially developed for, the networks achieved solid mean Intersection over Union (mIoU) scores. They were particularly good at differentiating between objects and background, but struggled more with distinguishing between different classes.

Our experiment found that reducing problem complexity from four to two classes improved the ability to distinguish objects from the background. By considering source [13], which claimed high classification scores on radar data, we investigated the concept of task decoupling using a two-stage network (TSN). This approach enhanced both our main objective of class distinction and furthermore object-background separation, thus significantly improving the overall accuracy. However, it came with an increased computational cost.

To offset this cost, we suggest the trade of combining the TSN with more computationally efficient classical methods, which could be especially beneficial in embedded systems, despite their lower prediction accuracy. This can be easily achieved due to the modular characteristics of TSN.

To further enhance the object class scores, we designed and tested three different sequential network approaches. Among them, the combination of a PointNet encoder and an LSTM (Long Short-Term Memory) head offered the most stable results and was also the smallest and fastest, making it a suitable candidate for future inclusion into the existing pipeline of Sensrad AB.

The primary limitation of this thesis was the limited variety within our dataset. Despite having a sufficient number of frames, there were only four static scenarios, which allowed the segmentation models to overfit the dataset.

Lastly, the two proposed metrics: mAEDC and mAEDO, help us better understand

how false positive points are distributed in a point cloud as compensation to mIoU. Moreover, their corresponding loss functions work as expected without decreasing mIoU, and the mAEDO loss can even improve the mAEDC metric at the same time. This finding indicates that employing the mAEDO loss function should be considered for a SemSeg network.

For future research:
Increasing the size and variety of the dataset could validate the results of our performance tests. It would be interesting to see how the performance ranking of the eight networks changes with more varied data, which would lead to less overfitting. Ultimately, employing a dynamic dataset could resolve this issue and provide deeper insights into the characteristics of the tested neural networks.

Given the growing interest in end-to-end network research, it might be worth investigating whether the object detection task can be handled all at once. For high 3D object detection accuracy transformer models could be investigated, while for computational efficiency discretization methods like PointPillars [34] could be assessed.

If pipeline control is a requirement, for safety-critical considerations, for instance, further research could be done on combinations of classical and neural network methods. Currently, the DBSCAN is a bottleneck and could be replaced with a neural network approach or an instance segmentation network, which predicts not only the class but also a cluster index.

Bibliography

[1] Christopher Olah, "Understanding lstm networks," 2015, [Online; accessed 28-Mai-2023]. [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs

[2] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," 2017.

[3] Wikipedia, "Dbscan — Wikipedia, the free encyclopedia," 2023, [Online; accessed 25-Mai-2023]. [Online]. Available: https://en.wikipedia.org/wiki/DBSCAN

[4] G. Horváth and G. Erdős, "Point cloud based robot cell calibration," *CIRP Annals*, vol. 66, no. 1, pp. 145–148, 2017.

[5] S. D. Pendleton, H. Andersen, X. Du, X. Shen, M. Meghjani, Y. H. Eng, D. Rus, and M. H. Ang Jr, "Perception, planning, control, and coordination for autonomous vehicles," *Machines*, vol. 5, no. 1, p. 6, 2017.

[6] J. Lin, P. Diekmann, C.-E. Framing, R. Zweigel, and D. Abel, "Maritime environment perception based on deep learning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 9, pp. 15 487–15 497, 2022.

[7] L. Piwek, D. A. Ellis, S. Andrews, and A. Joinson, "The rise of consumer health wearables: promises and barriers," *PLoS medicine*, vol. 13, no. 2, p. e1001953, 2016.

[8] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[9] M. Meyer and G. Kuschk, "Automotive radar dataset for deep learning based 3d object detection," in *2019 16th european radar conference (EuRAD)*. IEEE, 2019, pp. 129–132.

[10] A. Palffy, E. Pool, S. Baratam, J. F. Kooij, and D. M. Gavrila, "Multi-class road user detection with 3+ 1d radar in the view-of-delft dataset," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4961–4968, 2022.

[11] J. Rebut, A. Ouaknine, W. Malik, and P. Pérez, "Raw high-definition radar for multi-task learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 17 021–17 030.

[12] L. Zheng, Z. Ma, X. Zhu, B. Tan, S. Li, K. Long, W. Sun, S. Chen, L. Zhang, M. Wan *et al.*, "Tj4dradset: A 4d radar dataset for autonomous driving," *arXiv preprint arXiv:2204.13483*, 2022.

[13] J. Bai, L. Zheng, S. Li, B. Tan, S. Chen, and L. Huang, "Radar transformer: An object classification network based on 4d mmw imaging radar," *Sensors*, vol. 21, no. 11, p. 3854, 2021.

[14] F. Ding, Z. Pan, Y. Deng, J. Deng, and C. X. Lu, "Self-supervised scene flow estimation with 4-d automotive radar," *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 8233–8240, 2022.

[15] Y. Zhou, L. Liu, H. Zhao, M. López-Benítez, L. Yu, and Y. Yue, "Towards deep radar perception for autonomous driving: Datasets, methods, and challenges," *Sensors*, vol. 22, no. 11, p. 4208, 2022.

[16] Y. Zhao and H. Liu, "Overview on mimo radar," *Shuju Caiji Yu Chuli/Journal of Data Acquisition and Processing*, vol. 33, pp. 389–399, 05 2018.

[17] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun, "Deep learning for 3d point clouds: A survey," 2019. [Online]. Available: https://arxiv.org/abs/1912.12033

[18] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, C. Stachniss, and J. Gall, "Semantickitti: A dataset for semantic scene understanding of lidar sequences," 2019.

[19] W. K. Fong, R. Mohan, J. V. Hurtado, L. Zhou, H. Caesar, O. Beijbom, and A. Valada, "Panoptic nuscenes: A large-scale benchmark for lidar panoptic segmentation and tracking," 2021.

[20] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," *Advances in neural information processing systems*, vol. 30, 2017.

[21] H. Thomas, C. R. Qi, J.-E. Deschaud, B. Marcotegui, F. Goulette, and L. J. Guibas, "Kpconv: Flexible and deformable convolution for point clouds," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 6411–6420.

[22] Y. Liu, B. Fan, S. Xiang, and C. Pan, "Relation-shape convolutional neural network for point cloud analysis," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 8895–8904.

[23] S. Horache, J.-E. Deschaud, and F. Goulette, "3d point cloud registration with multi-scale architecture and unsupervised transfer learning," in *2021 International Conference on 3D Vision (3DV)*. IEEE, 2021, pp. 1351–1361.

[24] C. B. Choy, J. Gwak, and S. Savarese, "4d spatio-temporal convnets: Minkowski convolutional neural networks," *CoRR*, vol. abs/1904.08755, 2019. [Online]. Available: http://arxiv.org/abs/1904.08755

[25] Z. Liu, H. Hu, Y. Cao, Z. Zhang, and X. Tong, "A closer look at local aggregation operators in point cloud analysis," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXIII 16*. Springer, 2020, pp. 326–342.

[26] Z. Liu, H. Tang, Y. Lin, and S. Han, "Point-voxel cnn for efficient 3d deep learning," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[27] D. Berrar, "Bayes' theorem and naive bayes classifier," *Encyclopedia of Bioinformatics and Computational Biology: ABC of Bioinformatics*, vol. 403, p. 412, 2018.

[28] T. Bayes, "An essay towards solving a problem in the doctrine of chances," *Philosophical Transactions of the Royal Society of London*, vol. 53, pp. 370–418, 1763.

[29] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[30] T. Chaton, C. Nicolas, S. Horache, and L. Landrieu, "Torch-points3d: A modular multi-task frameworkfor reproducible deep learning on 3d point clouds," 2020. [Online]. Available: https://github.com/nicolas-chaulet/torch-points3d

[31] X. Liu, M. Yan, and J. Bohg, "Meteornet: Deep learning on dynamic 3d point cloud sequences," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 9246–9255.

[32] O. Schumann, J. Lombacher, M. Hahn, C. Wöhler, and J. Dickmann, "Scene understanding with automotive radar," *IEEE Transactions on Intelligent Vehicles*, vol. 5, no. 2, pp. 188–203, 2019.

[33] F. Nobis, F. Fent, J. Betz, and M. Lienkamp, "Kernel point convolution lstm networks for radar point cloud segmentation," *Applied Sciences*, vol. 11, no. 6, p. 2599, 2021.

[34] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "Pointpillars: Fast encoders for object detection from point clouds," *CoRR*, vol. abs/1812.05784, 2018. [Online]. Available: http://arxiv.org/abs/1812.05784

A

Appendix 1

This appendix includes all the explanations mentioned in Chapter 2.

A.1 Technical Specification of the Used 4D Radar

The detailed technical information of the 4D-imaging radar that we use in this project is listed in Tab.A.1, which is collected from the official website of Sensrad AB at `www.sensrad.com`. The radar chip-set, version Everest RPU 1.1, is developed by Arbe Robotics Ltd, details of which can be found on their official website: `www.arberobotics.com`.

Table A.1: Specification of the used 4D-imaging radar.

Parameter	Value
Frequency	$76 \sim 81$ GHz
Antennas	48×48 MIMO configuration
Virtual channels	2304
Frame rate	> 15 fps
Number of simultaneously tracked objects	> 500
Detection range	$0.2 \sim 400$ m
Target radial velocity	-35 m/sec $\sim +70$ m/sec
FOV (azimuth)	$\pm 50°$
FOV (elevation)	$30°$
Resolution (range)	$0.1 \sim 0.75$ m
Resolution (azimuth)	$1.25°$
Resolution (elevation)	$1.7°$

B
Appendix 2

This appendix includes all the explanations mentioned in Chapter 4.

B.1 Training Conditions for SemSeg Networks

In order to facilitate a reproducible comparison, we have maintained the neural networks in their original form as provided in the library `torch-points3d`. Further, the networks underwent training under identical conditions such as dataset, features, augmentation, normalization technique, class weights, epoch count, and GPU resource utilization. The dataset used is the SemSeg dataset v1 explained in section 2.2. Where the six features: Cartesian coordinates, range to the radar, power, and Doppler, have been fed to the networks. The argumentation consists of rotations around the z-axis. Regarding the feature normalization, Z-score normalization Eq. B.1 is utilized. Hereby, the parameters μ_i and σ_i are determined using the training set, ensuring frame-independent and shape-conserving normalization in the validation and test case.

$$\text{Z-score normalization:} \quad \frac{\mathbf{x} - \mu_{\mathbf{x}}}{\sigma_{\mathbf{x}}}. \tag{B.1}$$

Due to the imbalance of the dataset, different weights for each class are necessary. Since this is a semantic segmentation task, the class weights are calculated according to the number of points of each class as follows

$$w_c = \sqrt{\frac{1}{N_c} \cdot \frac{\sum_{c=1}^{N_c} n_c}{n_c}}, \tag{B.2}$$

where w_c is the weight for the class c, N_c is the number of classes, and n_c is the total number of points of this class among all training samples. As a result, the class with a larger number of training samples will be assigned a smaller weight.

Afterward, every network has been trained for 50 epochs on the same GPU, NVIDIA GeForce GTX TITAN X, with the GPU memory of 12 GB fully used, resulting in different batch sizes varying from 2 to 64, see Tab.B.1.

Table B.1: Batch size of each SemSeg network.

Neural network	**Batch size**
Minkowski	**64**
PointNet	32
PointNet++(SSG)	32
MSSVConv	32
PVCNN	32
KPConv	24
PointNet++(MSG)	16
RSConv	4
PPNet	2

C
Appendix 3

This appendix includes all the explanations mentioned in Chapter 5.

C.1 Training Conditions for TSNs

Table C.1: Training Conditions for the networks.

Model	Dataset	epochs	optimizer	lr	dec
PointNet (SemSeg)	SemSeg V1	50	ADAM	1×10^{-3}	0.7 per 20 ep
PointNet (Masking)	SemSeg V1	50	ADAM	1×10^{-3}	0.7 per 20 ep
PointNet (Cls)	Classification V1	50	ADAM	1×10^{-3}	0.7 per 20 ep
TSN (Masking)	SemSeg V1	50	ADAM	1×10^{-3}	0.7 per 20 ep
TSN (Cls)	Classification V2	50	ADAM	1×10^{-3}	0.7 per 20 ep

C.1.1 Data Augmentation

The **classification data** underwent a process that included a dropout rate of 0.4, rotation around the z-axis, and translation. On the other hand, the **semantic segmentation data** is subjected solely to rotation around the z-axis.

C.1.2 Normalization

The normalization method utilized is the Z-score normalization, with μ_x and σ_x computed over the test set. In an effort to avoid distorting the object shapes, the spatial values - x, y, and z - are shifted by μ_x, μ_y, and μ_z respectively. However, they are all divided by a single σ_d, representing the standard deviation of the Euclidean distance of all points to the mean point cloud center of the test set- μ_x, μ_y, and μ_z. This approach ensures the preservation of object shapes by scaling them down uniformly during normalization. For other features, specifically range power and Doppler, individual μ_x and σ_x values are utilized, given their independence from each other.

$$\text{Z-score normalization:} \quad \frac{\mathbf{x} - \mu_{\mathbf{x}}}{\sigma_{\mathbf{x}}}, \tag{C.1}$$

C.1.3 Equal Sampling vs Class Weights

It's worth mentioning that the initial **classification network** is designed with weights to tackle label imbalance. Unfortunately, the model didn't learn effectively, resulting in misdirected predictions split between pedestrians and bicyclists. However, significant improvements are seen in the validation performance when equal sampling is introduced. This approach ensured that the network received an equal number of samples from all classes, leading to the results we presented.

C.1.4 Comparison: cluster dataset vs sequential cluster dataset

As mentioned in subsection 2.2.4, both cluster dataset and sequential cluster dataset can be used to train classification networks that only consume one cluster at a time. Even though both datasets are leveraging the same annotation explained in subsection 2.2.1, there is a small difference in that the sequential cluster dataset, which is extracted from the pipeline using DBSCAN and MOT, results in similar cluster shapes as in the pipeline than the cluster dataset. The shape difference comes from the imperfect performance of DBSCAN and MOT so that one ground truth cluster can be divided into two smaller clusters and the other way around. This is actually a vital argument since TSN will finally be integrated as used in the pipeline, it is preferable to train the classification network on a dataset as close to the real situation as possible.

To verify this idea, the same classification network, PointNet, has been trained twice, once trained on the cluster dataset, and once trained on the sequential cluster dataset, but both are validated on the sequential cluster validation dataset. Fig.C.1 illustrates that PointNet trained on the sequential cluster dataset outperforms PointNet trained on the cluster dataset by a large margin on the sequential cluster validation dataset, which proves that our idea is correct and the shape difference caused by DBSCAN and MOT is non-negligible.

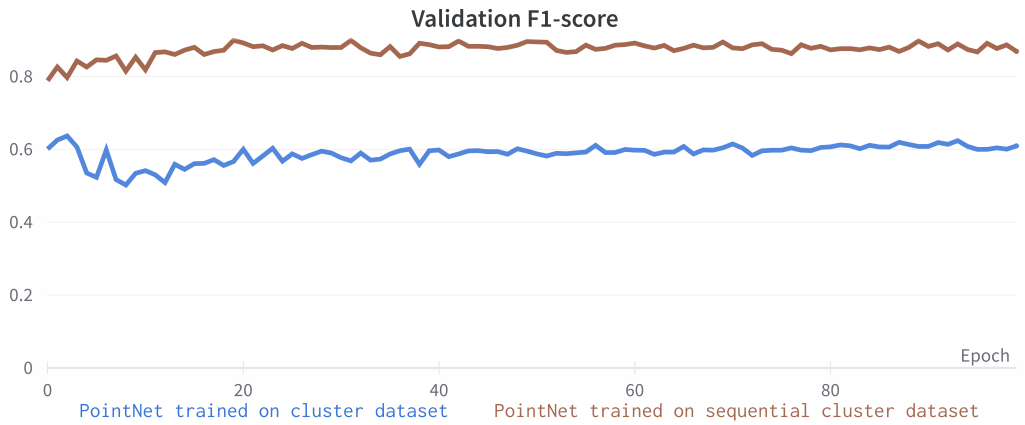


Figure C.1: F1-score of PointNet trained on cluster dataset and sequential cluster dataset on the sequential cluster validation dataset.

D

Appendix 4

This appendix includes all the explanations mentioned in Chapter 6.

D.1 Training Conditions for Sequential Classifiers

Table D.1: Training Conditions for sequential classifiers.

Network	Dataset	Epochs	Optimizer	lr	decay	Dropout rate
All networks	Cls Seq V1	50	ADAM	5×10^{-4}	0.1 per 20 ep	0.4

The same **Data Augmentation** techniques employed in the TSN are explained in section C.1.1.

The equal **Sampling** technique, as mentioned in section C.1.3, proved beneficial when applied in the TSN, and therefore, it is utilized in the sequential training pipeline as well. Tracks longer than 30 are randomly truncated to a length of 30, while shorter ones are taken as they are. As mentioned, equal sampling is employed to select tracks from each class with the same probability. Of note, once a track ends, the hidden states (h_0, c_0) or the accumulated feature states are reset for the LSTM and the FNN, CNN respectively before a new track starts.

D.2 Parameters for Different Heads

Table D.2: Parameters for the CNN head architecture.

Parameter	Meaning	Value
n	number of classes	3
m	feature size	512
T	number of memorized time instances	50
s	stride	1
p	zero padding size	[0,0]
k	kernel size	[10,512]
C_c	convolutional layer output channel size	256
F_i	input FC layer size	[10496, 1024]
F_h	hidden FC layer size	[1024, 128]
F_o	output FC layer size	[128, n]
d_h	dropout rate on hidden FC layer	0.4

Table D.3: Parameters for the FNN head architecture.

Parameter	Meaning	Value
n	number of classes	3
m	feature size	512
T	number of memorized time instances	50
F_i	input FC layer size	[$t \times m$, 1024]
F_{h1}	1^{st} hidden FC layer size	[1024, 512]
F_{h2}	2^{nd} hidden FC layer size	[512, 128]
F_o	output FC layer size	[128, n]
d_{h2}	dropout rate on 2^{nd} hidden FC layer	0.4

Table D.4: Parameters for the LSTM head architecture.

Parameter	Meaning	Value
n	number of classes	3
m	feature size	512
l	number of recurrent layers	1
H	hidden state size	3
C	cell state size	3

D.3 Noise Clusters in Sequential Classifiers

The challenge of generating noise information from scratch, that is neither biased towards a particular class nor deviating from the standard cluster distribution, rendered us hesitant to attempt it. Instead, we utilized the validation set to identify the

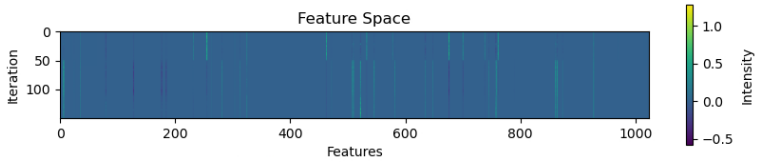
most ambiguous clusters. We employed PointNet to generate a prediction for each cluster, namely class three probabilities and then calculated the Euclidean distance between these probabilities and a hypothetically undefined cluster, where all probabilities, denoted by p_u, equate to $\frac{1}{K}$, with K representing the number of classes. The formula to calculate the distance is given as

$$d = \sqrt{\sum_{i=1}^{k}(p_i - \frac{1}{k})^2}.$$

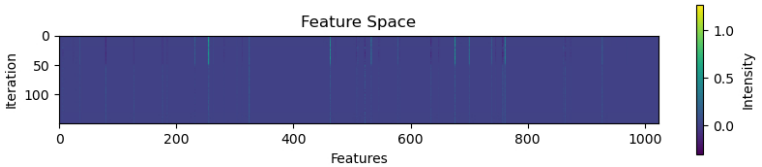
Then the 15 closed clusters to the theoretical undefined cluster are chosen to be randomly selected for the tests.

D.4 Feature Spaces in Sequential Classifiers

Feature space of the four sequential networks given a track of 150 clusters. The track switches labels from pedestrian to (a) bicyclist or (b) noise after the first 50 iterations (y-axis). Take note that the minimum and maximum values fluctuate across different plots. While this may complicate the direct comparison between figures, it enhances the color resolution within each individual image, thereby highlighting patterns more effectively.

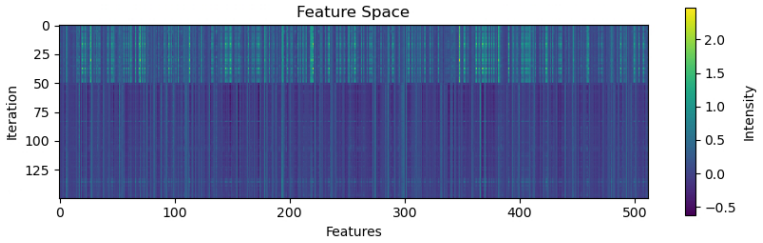


(a) Switch to Class.

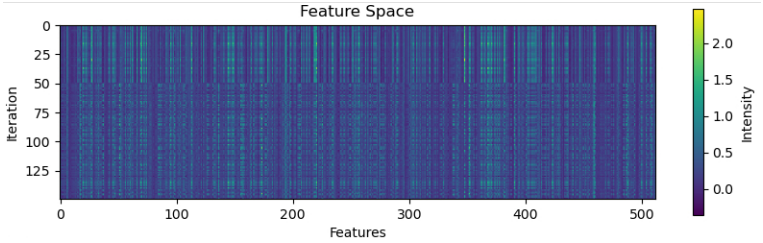


(b) Switch to Noise.

Figure D.1: Sequence of 150 clusters, encoder size 1024 of original PointNet.

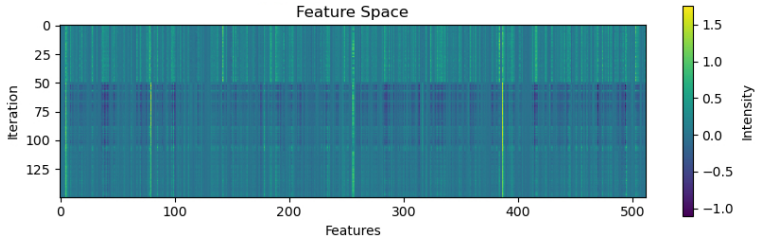


(a) Switch to Class.

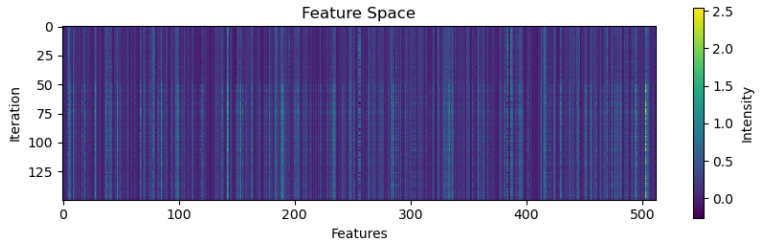


(b) Switch to Noise.

Figure D.2: Sequence of 150 clusters, encoder size 512 of FNN.

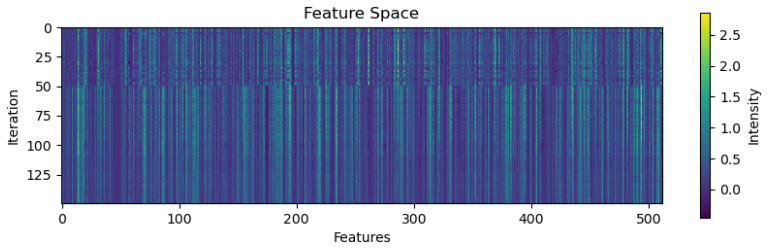


(a) Switch to Class.

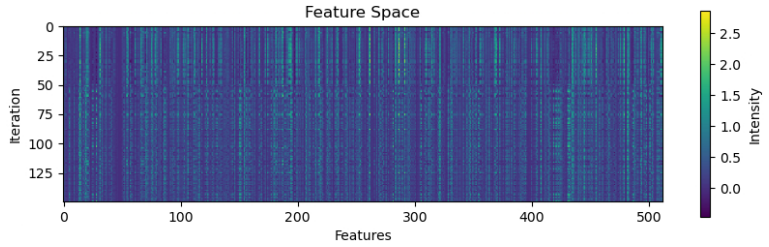


(b) Switch to Noise.

Figure D.3: Sequence of 150 clusters, encoder size 512 of CNN.



(a) Switch to Class



(b) Switch to Noise.

Figure D.4: Sequence of 150 clusters, encoder size 512 of LSTM.

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY