Radar Detection Using Deep Learning

Leonardo Carrera, Ziliang Xiong

Master's thesis
2022:E65

Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

Master's Programme in Machine Learning, Systems and Control

Radar Detection Using Deep Learning

Radardetektering med djup maskininlärning

by

Leonardo David Carrera le7763ca-s@student.lu.se
Ziliang Xiong zi8602xi-s@student.lu.se

Abstract This thesis aims to reproduce and improve a paper about dynamic road user detection on 2D bird's-eye-view radar point cloud in the context of autonomous driving. We choose RadarScenes, a recent large public dataset, to train and test deep neural networks. We adopt the two best approaches, the image-based object detector with grid mappings approach and the semantic segmentation-based clustering approach. YOLO v3 and Pointnet++ are the deep networks for the two approaches, respectively. We implement an radar-based version of DBSCAN to extract instance clusters (objects). For both approaches, various preprocessing techniques are implemented, such as velocity skew function, upsampling and data augmentations, including rotation and flipping. We also adapt the evaluation metrics, IOU, mAP, and F1-score for point clusters so that both approaches' output can be comparable. The reproduction of both approaches achieved comparable performance as in the original paper, which indicates the image-based detector overwhelmed the semantic segmentation-based clustering approach. We also managed to improve the metrics by adapting clever variations in the DBSCAN pipeline. Besides, we implemented the ablation study for the YOLO approach and found horizontal flipping the point cloud as the optimal data augmentation operation. We implemented the ablation study for the PointNet/DBSCAN pipeline as well and found that randomly jittering the points considering the radial velocity of the radar reflections output the best model, and in under specific cases, it improved it. We also investigated the effect of time accumulation on APs of all the classes. We found that low AP of the pedestrian class is the performance bottleneck, and simply accumulating a longer period cannot significantly improve it.

FMAM02
Master's Thesis (30 credits ECTS)
August, 2022
Supervisor: Mikael Nilsson
Examiner: Kalle Åström

Acknowledgements

Ziliang Xiong, I would like to express my deepest gratitude to my academic supervisor, Mikael Nilsson, for his precious patience and feedback. He also participated frequently in the supervision meeting, especially at the time when we were faced with poor outcomes. His suggestion on visualization of ground truth bounding boxes in grid mappings helped me embrace a breakthrough in the performance. I also could not have undertaken the journey without the daily help from my industrial supervisors, Maryam Fatemi and Erik Werner. Their debug sessions saved me from struggles with massive codes and helped clear my mind. Additionally, I am extremely grateful to my examiner, Kalle Astrom, with whom we put a satisfactory end to this thesis. I would also thank Zenseact for funding this thesis.

Special thanks to my thesis teammate, Leonardo David, who shared half of my pressure and helped me through hundreds of discussions. Thanks should also go to Fabian Sörensson, Johannes Kübel, Julian Brandes. They were generous with their time for helping me with tons of merge conflicts on Gerrit. I also enjoyed playing ping pong on the second floor and fika on the balcony. I'm also grateful to Nicolas Scheiner, the author of the paper we tried to reproduce. He patiently offered a lot of insights into handling details.

Lastly, I'd like to acknowledge my parents for generously supporting me during my master's study. Especially my mother's encouragement comforted me during the hardest period. Her advice on getting along with colleagues was always wise. I'd also like to recognize Jianan Liu. The discussion of model performance with him was encouraging and inspiring. I would be remiss in not mentioning my ex-girlfriend, who was my very reason to move to Gothenburg for this thesis. I'm sorry that it didn't work out in the end.

Leonardo Carrera, I wish to show my gratitude and appreciation to everyone who was involved and helped in completing this master thesis and during my master's program journey. First, to my parents, Jorge and Yolanda, my sister Daniela, and my girlfriend Lucía, who gave me the strength, love, and wisdom to endure and succeed in this challenge. To all my friends who made this journey easier with their joy and advice. To my professors who guided and enlightened me with their teachings.

Special thanks to Ziliang Xiong, Bo Bernhardsson, Mikael Nilsson, Kalle Åström, Anders Robertsson, Myriam Fatemi, and many others not included here.

Contents

1 Introduction **11**
1.1 Research Problem . . . 12
1.2 Background . . . 13
1.2.1 Radar . . . 13
1.2.2 Frequency Modulated Continuous Wave (FMCW) radar . . . 14
1.2.3 Automotive Radar Properties . . . 16
1.2.4 Radar Point Clouds . . . 17
1.3 Outline of the Thesis . . . 18

2 Theory **20**
2.1 Multilayer perceptrons . . . 20
2.1.1 Convolutional Neural Networks (CNN) . . . 22
2.1.2 Training . . . 23
2.2 Classical Detection and Instance Segmentation . . . 24
2.2.1 Object Detection . . . 24
2.2.2 Sliding Window Detection . . . 25
2.2.3 Semantic and Instance Segmentation . . . 26
2.3 Clustering Analysis . . . 26
2.4 YOLO . . . 26
2.5 Deep Learning Segmentation and Clustering on Point Clouds . . . 29
2.5.1 PointNet . . . 29
2.5.2 PointNet++ . . . 31
2.6 DBSCAN Clustering Algorithm . . . 32
2.7 Evaluation Metrics . . . 33
2.7.1 Point-wise IOU . . . 34
2.7.2 Average Precision per class (AP) and Mean Average Precision (mAP) . . . 34
2.7.3 Harmonic Mean of Precision and Recall (F1 score) . . . 36

3 Dataset **37**
3.1 A Comparison . . . 37
3.2 RadarScenes Introduction . . . 39
3.2.1 Sensor Settings . . . 39
3.2.2 Data Structure . . . 40
3.2.3 Annotations . . . 40
3.3 Dataset Analysis . . . 42
3.4 Snippet Extraction . . . 42
3.4.1 Ego-motion Compensation . . . 45

3.4.2 Train, Validation, Test Sets Split 46

4 Methods 47
4.1 Image Object Detection Network: YOLO with Grid Maps 47
4.1.1 Bounding Box Extraction 48
4.1.2 Grid Mapping 51
4.1.3 Doppler Velocity Skew Function 51
4.1.4 Blurry Filter 51
4.1.5 Data Augmentation 51
4.2 Instance Segmentation Approach: Semantic Segmentation Network (PointNet++) and Radar Clustering (DBSCAN) 54
4.2.1 Radar Point Cloud Extraction 54
4.2.2 Upsampling and Clipping of Dataset 57
4.2.3 Point Cloud Preprocessing for Training the Semantic Network 58
4.2.4 PointNet++ Semantic Segmentation Architecture 60
4.2.5 Point Cloud Filtering for Clustering 61
4.2.6 Class-Sensitive Filtering and Clustering 63

5 Results and Discussion 66
5.1 YOLO v3 Training and Evaluation 66
5.1.1 Ablation Study 66
5.2 Semantic Segmentation Network (PointNet++) Training and Evaluation 67
5.2.1 Semantic Segmentation Network (PointNet++) Baseline Model 68
5.2.2 Inference and Clustering 73
5.3 Comparison of Two Approaches 73
5.4 Discussion 73
5.4.1 Weight Initialization 75
5.4.2 DBSCAN Settings 76

6 Conclusion and Future Research 79
6.1 Conclusion 79
6.2 Future Research 80

List of Figures

1.1 Simple diagram of the FMCW radar where the transmitter (T_X) and receiver (R_X) antennas are shown. The chirp generator is shown and connected to the transmitter antenna, and the Analog-to-Digital Converter (ADC) processor is shown. 14

1.2 Both: Representations of the chirp. Left: Amplitude vs. time plot of transmitted signal. Right: Frequency vs. time plot for both, transmitted (red) and received (blue dashed) signal. T_c is the the time of transmitted signal, τ is the difference between transmitted and received signal, n is the number of chirp up to N (one scan). 15

1.3 Position and orientation of the three coordinate systems. A black dashed line describes the trajectory of the vehicle. a) gives an overview of the three systems. b) The resulting vectors are formed considering the yaw angle γ and the azimuth angle $\phi_{\text{sensor},1}$ of the *first* sensor. . . 17

1.4 Representation of a 2D radar point cloud with its extra features not plotted. Labels per point included in this plot. 17

2.1 An example of a simple feedforward neural network called a multilayer perceptron. There are three fully connected layers and one "hidden" layer. The rightmost units could represent probabilities for two classes or be real numbers regressing two quantitative outputs. 21

2.2 Typical convolutional neural network layer. Here, the convolutional network is viewed as a small number of relatively complex layers, and each layer is populated with many *stages*. In this terminology, there is a one-to-one mapping between kernel tensors and network layers. . 22

2.3 Typical behavior in training and test data set loss errors, respectively. It is seen that after the red line, the gap increases, indicating overfitting. The opposite is underfitting, and it is located before the red line. 24

2.4 YOLO object detection example (Source: MTheiler). 25

2.5 An example image from the Plant seedling dataset [15]. Red rectangles are the sliding windows. 25

2.6 Semantic segmentation and instance segmentation performed over radar point clouds. 27

2.7 YOLO v3 architecture: Darknet 53 as the backbone feature extractor and three prediction heads for various scales. 28

2.8 PointNet pipeline. The input and feature transformations are shown as well as the max pooling operation for getting features. Two branches are shown: segmentation and classification, being segmentation an extension of the classification network. MLP are multi-layer perceptrons and the numbers, their layer sizes. The image is taken from the paper of Qi et al. [32]©IEEE 2018, Fig.2. 30

2.9 Symmetric function of max pooling in PointNet. 31

2.10 Input transform in PointNet.From [32] ©IEEE 2017, Fig.2. 31

2.11 (a) Multi-Scale Grouping (MSG), (b) Multi-resolution Grouping (MRG). Image from paper of Qi et al. [33]IEEE 2018, Fig.3. 32

2.12 The PointNet++ hierarchical feature learning architecture description. Both applications (segmentation and classification) are shown in the bifurcation. The example shown for 2D points. Image from the paper of Qi et al. [33] ©IEEE 2018, Fig.2. 33

2.13 Representation of a cluster formation using DBSCAN. Here, $minPts = 3$ with a certain ϵ. Red points are core points since they meet both requisites. Yellow points only meet one of the criteria; therefore, they are density-reachable points. The blue point does not meet any directive, and it is considered an outlier or noise. The double direction arrow indicates that the points share both conditions successfully, while the one direction arrow shows that only one of the conditions was met. 34

2.14 Point-wise IoU visualization. It depicts a predicted bounding box to illustrate the object cluster that is containing the predicted object. . 35

3.1 The car coordinate(cc) is marked orange. All the sensor measurements are relative to it. Four radar sensors are mounted at the front face close to the lights. The documentary camera is mounted behind the windscreen(white). The FOV (±60°)of each radar sensor is marked in different colors. From [39]©IEEE, Fig.3. 39

3.2 Up Left : Urban driving; Up Middle: Cross section; Up Right: Highway; Bottom Left:T section; Bottom Middle: Crowded area; Bottom Right: Motorbike 41

3.3 Histogram for the number labeled points in each frame. Most frames contains less than 10 labelled points, which is quite sparse and requires accumulation over time to improve the density. 42

3.4 Histogram for the number of objects in each frame. Most frames contains less than 3 objects. 43

3.5 Left:Histogram for time intervals between two succession sensor scans. The frequency decreases as the interval length increases, which shows the fours sensors' cycles are not synchronized. Right: Histogram for time intervals between two succession sensor scans in sensor NO.3. All the time interval locates in a narrow band, which verifies that the working cycles lasts around 60 ms. 43

3.6 Distribution of number of instances of each class 44

3.7 Snippet Extraction: 1. accumulate frames over time to reach 500 milliseconds; 2. Clip the snippet to remove outliers; 3. Check if there is no target in the snippet. If so, go to the next snippet; 4. Remove the clusters that has no volume, i.e., clusters contain only one or two points. 45
3.8 Left: The class distribution on the test set; Right: The class distribution on the combined set of the training set and test set. 46

4.1 Baseline pipelines for the two studied object detection models using RadarScenes data set. 48
4.2 Sequence 109 start index 0 number of future frames 28; the left is the whole snippet, the right one is enlarged axis aligned bounding boxes . 49
4.3 Oriented Bounding Boxes: The snippet from the first 28 frames in sequence 137. 50
4.4 An example for grid mapping: a snippet from sequence 109 frame 816 to frame 845. (a)reference image;(b)point cloud with oriented bounding boxes. Three objects are visible;(c)Amplitude map. There are three clusters marked as light yellow corresponding to three objects; (d) Maximum Doppler map, the correspondence is more obvious; (e)Minimum Doppler map, the correspondence is less obvious. All three objects are moving away from the test vehicle, therefore max Doppler map shows the clearest correspondences. 52
4.5 Left: Histogram of the radial velocity distribution; Right: Fourth order polynomial skew function. 53
4.6 The propagation scheme for blurry filter from [8] IEEE, 2021, Fig. 6. 53
4.7 Pipeline for the object detection system using the algorithms of DBSCAN and PointNet++. 55
4.8 A snippet with the original number of points (b), after clipping (a), and after random upsampling (c) procedures. Note that the clipping is done over static class points in (a) preserving the non-static points. Also, only for this example one can see that the length of the snippet is not 4096 points 58
4.9 A snippet with the original number of points (a) and after data augmentation (b). Note the increasing number of points in dynamic classes and reduced by the same number in the static class. Although the jittering is not as visible as expected in the plots, the favorable effects of the training are remarkable. 61
4.10 Structure of the semantic segmentation network. The segmented line indicates skip connections from which extracted features from MSG modules are passed to the FP modules. Improved from [36] and [37]. 62
4.11 Original snippet (a), pruned snippet after filtering (b). Notice the number of static points dropped. In this example, $\eta_{v_r} = 0.0022$. . . . 63
4.12 Original snippet with ground truth clusters (LEFT) and the same snippet with predicted labels and clusters (RIGHT). In this case, one car and two pedestrians clusters are shown along with the noise (static) points. 65

5.1 YOLO approach's performance on the validation set with various data augmentation operation 67

5.2 mAP and AP of each class on the test set changes over the accumulation time. 68
5.3 Visualization ground truth bounding boxes vs. detection bouding boxes 69
5.4 Confusion matrix (absolute) for the baseline model in the test data set 71
5.5 Confusion matrix (relative) for the baseline model in the test data set 72
5.6 Loss error in the unseen data for our semantic network baseline model. 72
5.7 IoU in the unseen data for our semantic network baseline model. . . . 72
5.8 Inference (predicted labels and clustering) results on two typical traffic scenarios of two main approaches with ground truth clusters as the references. Ground truth are in the left hand side (a and c) and the corresponding predicted ones at the right hand side (b and d). The top two are the same snippet (a and b) and the two in the bottom are both the same (c and d). Note the effect of the filter in the predicted ones, also notice the little differences from a to b where there are more clusters than in the original plot. 74

List of Tables

1.1 Example of 2D radar data points and features gathered from a small number of frames and from the RadarScenes dataset 18

3.1 Overview of popular public radar data sets 38

4.1 Estimated number of points in percentages across the entire dataset from RadarScenes 59

5.1 Metrics of AP per class 73
5.2 Result scores for all methods on the test set 77
5.3 Number of points in a cluster of each class 78
5.4 RCS of points in a cluster of each class 78

1 Introduction

Deep Learning technology has become more prevalent in recent years for diverse tasks that require predictions, both in industry and at home. This technology has had a such impact because of the capability of accurate and quick interpretations of complex patterns over collected data sets. Thus, more and more successful design implementations based on deep learning are emerging in new fields to address such tasks.

The field of assisted/autonomous driving has been taking great advantage of deep learning frameworks since they have solved critical challenges: To efficiently perceive the road environment in which the car is located through sensor systems, make decisions about the interpretations of the environment detection system, control the mechanisms of the vehicle based on the findings as mentioned earlier with the aim of safe and efficient navigation, and finally, feedback the detection and decision system with new data from the environment.

More and more deep learning algorithms for object detection have been developed and perfected with remarkable outcomes [45], [29]. Such performance has allowed these models to be considered in detection tasks where real-time process, fine accuracy, and reliability are crucial.

According to the best of our knowledge, autonomous driving and Advanced Driver Assistance Systems (ADAS) are the most well-known and proficient fields where deep learning has had a tremendous impact. Numerous representative works combine automotive sensory systems with deep learning techniques [1]. i.e., combined perception-and-detection tasks of road elements have been successfully explored and implemented using automotive sensory hardware. These works have allowed an evolution in the deep learning algorithms oriented to autonomous driving. We are encouraged then to challenge the latest deep learning models with one type of sensor: the automotive radar.

It is worth mentioning that there are extensive and well-documented works on detection systems. Most of them have combined video cameras, LIDAR sensors, and radar sensors to record and interpret a vehicle's navigation environment with great accuracy and robustness [4], [9], [25]. Moreover, there are works where radar is considered a key element but not essential [43]. Instead, it has been used to provide robustness and redundancy to object detection systems based on cameras and LIDAR sensors. This lack of interest is primarily due to the radar's challenge in gathering and decoding its information and the clear advantage the data retrieved from video cameras or LIDAR sensors have over the radar. Thus, radar object detection remains a niche with unexplored challenging potential.

At the time this work is being deployed, there are no thoroughly investigated standalone deep learning models for moving object detection tasks that rely exclusively on radar data. Considering that the radar is a commonly adopted sensor in the automotive industry and its information has not been fully exploited by machine/deep learning techniques, investigating methods for performing radar detection with deep learning seems like a promising research direction.

In this work, we explore deep learning methods applied solely to radar data; thus, we intend to offer insights and solid foundations to the state-of-the-art radar object detection field. We are formally focusing on studying two prolific sets of models based on deep learning frameworks for later evaluation on radar processed radar data sets.

1.1 Research Problem

Radar sensors have become standard equipment in almost every motor vehicle for their unique performance and reasonable costs. They are mainly used in assisted driving and safety awareness on roads, i.e., non-categorical detections and distance ranging tasks [7]. These tasks include road-lane keeping, braking/parking assistance, and cruiser assistance. Radar sensor systems have also been used in cars as side components to add robustness to other sensory mechanisms. Radar readings gather unique, relevant information about the environment when appropriate preprocessing techniques are used.

The radar can detect the relative position of objects and infer properties from such objects located around the sensor. The radar can gather such properties from the surroundings due to its ability to measure radial distances, relative angles with respect to some radar's north (azimuth), radial velocities, and RCS (Radar Cross-Section) values [40]. The spectrum of properties and the robustness against adverse weather conditions constitute an advantage over other sensors [36]. Also, in [38], it is explained how Doppler signatures in radar data are used to filter out moving objects from the static ones, alleviating the inherent data sparsity of moving (dynamic) objects. Radar data (static and dynamic) is cumulative over frames, and moving objects (dynamic) require shorter time frames to be reflected and gathered, typically in the order of a few hundred milliseconds [36]. This property gives solid measurements so the detection algorithms can take advantage of it. Thus, all these properties make the radar a good candidate to be considered as input in driverless tasks powered by machine learning algorithms.

In contrast, radar has been less studied and developed along with detection algorithms. The lack of related works is because the processing of raw data is slightly more difficult to interpret and decode than in other sensory systems, i.e., angular sparsity and lack of altitude information.

However, radar systems can be complemented with deep learning algorithms with good enough results, as in [36] and complete the current radar system features (basic pure detection and ranging) semantic-environment-understanding systems.

Our research problem is unraveled in a structured, measurable, and more understandable manner as follows,

- As for many automated driving functions nowadays, a highly accurate perception of the vehicle environment is crucial. Therefore, a robust object detection framework is the goal.

- The study is limited to the usage of conventional automotive radar sensors, i.e., high-resolution radars with advanced features like elevation or polarimetric information are not considered. Moreover, non-specialized computers have been considered, i.e., the study needs to be capable of being conducted and implemented by making use of standard personal computer capabilities.

- Then, the majority of automobile radar research has focused only on categorization or instance detection. For multi-class object recognition tasks on dynamic road users, there are fewer research documents accessible.

- The problem is considered a 2D bird-eye-view point cloud only, with the sole input of raw radar data through the algorithms.

- Finally, this work's focus is entirely aimed at object detections, i.e., the localization and classification of moving road users (MRU) and vulnerable road users (VRU) such as pedestrians.

1.2 Background

1.2.1 Radar

Radar is a common type of detection device, the history of which dates back to the early twentieth century [11]. An emitter actively emits electromagnetic waves, and a receptor collects the waves that have bounced back from surfaces in form of reflections. If these electromagnetic waves are well decoded, they can determine the range, angle, and velocity of objects by analyzing the reflected signals. A typical radar system usually contains a transmitter, a transmitting antenna, a receiving antenna, a receiver, and a processor, as is shown in Figure 1.1.

The radar systems offer attractive advantages compared to other similar sensors. They include robustness to adverse weather, like snow and heavy fog, fully functioning in darkness, and the ability to cover long-distance ranges. It might be the only sensor that can detect occluded objects and provide speed information. These features are complementary to the weakness of the camera and Lidar, which makes it an arguably necessary sensor for AVs. On the other hand, it also has disadvantages due to hardware limitations. It suffers from multi-path reflections, low angular

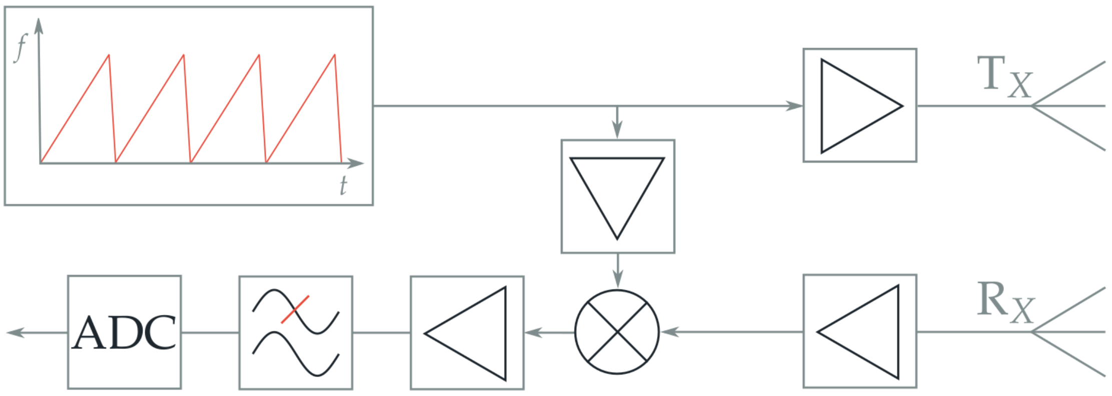


Figure 1.1: Simple diagram of the FMCW radar where the transmitter (T_X) and receiver (R_X) antennas are shown. The chirp generator is shown and connected to the transmitter antenna, and the Analog-to-Digital Converter (ADC) processor is shown.

resolution, and a lack of height information (except for 4D imaging radars where this information is available with high quality) [7].

1.2.2 Frequency Modulated Continuous Wave (FMCW) radar

The working frequency and emission patterns of a radar vary from device to device. In 2015, the World Radio communication Conference decided on a bandwidth of 77.5-78.0GHz in radio localization applications [21]. Therefore, the most widely used vehicle radar transmits millimeter waves (MMW). MMW technology has narrower wave beams than a microwave, which makes its high resolution on small objects. Furthermore, the hardware is also smaller and more portable. Such radar is called MMW radar directly.

MMW radars have been widely used in ADAS systems. One popular emission pattern is frequency modulated continuous wave (FMCW). As the name suggests, FMCW radar transmits MMW with linearly increasing frequency that lasts for a particular duration [22]. Such a signal is called a chirp, as is seen in Figure 1.2. A certain number of chirps that are transmitted evenly spaced in the time domain are a chirp frame. The range of the detected object is determined by the time delay between the Tx chirp and the Rx chirp.

Range, azimuth, and radial velocity are restored by three independent fast Fourier transforms. They are all in the polar coordinate with the radar as the origin. In addition to the motion states, FMCW radar also measures the radar cross-section (RCS) of the object (measured in m^2 and often reported in a logarithmic scale as dBsm). RCS shows how strong the reflection of an object is, which is determined by several factors: the material of the target, the size of the target, incident angle and the reflection angle, and some other properties of the material. The RCS property does not depend on the distance or strength of the emitter. Thus, one can use the RCS as the electromagnetic signature of the object. Such property is resourceful in a classification algorithm; for instance, in the current project, pedestrians and vehicles class labels have different surface materials and, therefore, different RCS

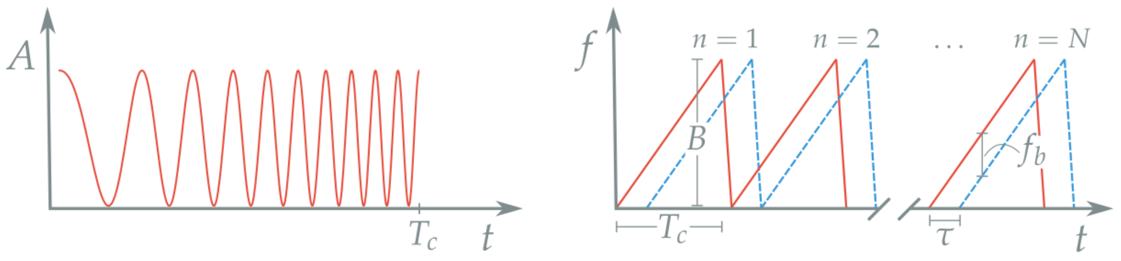


Figure 1.2: Both: Representations of the chirp. Left: Amplitude vs. time plot of transmitted signal. Right: Frequency vs. time plot for both, transmitted (red) and received (blue dashed) signal. T_c is the the time of transmitted signal, τ is the difference between transmitted and received signal, n is the number of chirp up to N (one scan).

signatures. The neural network will extract features depending on the RCS value the object has.

- **Radar Signal Processing**. The FMCW technique measures **range**, **azimuth angle** and **radial velocity** simultaneously on a phase level. The radar itself gathers these radar features through chirps. Formally, let a chirp n be the reflection which includes: the frequency f of the reflection, the time of transmitted signal T_c, and the difference in time τ between the transmitted signal and the received one, as shown in Figure 1.2. A number N of chirps makes one *scan*. Acknowledge that a *scan* is defined as one complete measured cycle where the three properties as mentioned earlier are measured for the objects within the field of view of the sensor.

- **Range and Doppler Estimation**. A matrix of $M \times N$ is considered, where M represents all the measurements in time positions of a sampled chirp in a certain time frame t, and N the number of chirps able to be created from the radar reflections. Then, through *Fast Fourier Transforms (FFT)*, the same matrix becomes the readings of range (in M rows) and Doppler (in N columns).

- **Azimuth Angle Estimation**. The feature that fully defines the direction of the Doppler velocity is the *azimuth angle*, which is estimated using rotating antennas (or arrays of antennas in vehicles instead). This angle is relative to the sensor's coordinate system. The phase differences at multiple Rx signals are used to restore the azimuth.

- **Target Extraction**. There are algorithms such as the *Constant False Alarm Rate (CFAR)* (and its derivations) that are used to identify peaks in the signal that truly belong to objects. Often, an actual measurement of a peak could represent either noise or an actual moving object, and that is the reason why the *false alarm* term is taken for describing it. This algorithm and variations in the automotive field usually extract targets with the following information:

  1. range r (in m)
  2. azimuth angle ϕ (in rad)
  3. Doppler velocity v_r (in m/s)
  4. RCS (radar cross section) value σ (id dBsm)

1.2.3 Automotive Radar Properties

There are conventions on the automotive radar data adapted to the studied deep learning algorithms along this work. They are not studied in detail in the Theory chapter since they are not within the scope of this thesis, but a general introduction of such conventions is necessary to understand the terminology and the effect they have on the implemented algorithms. Here, only the relevant ones reappear in this work and are described briefly. Further applicability and usage of these definitions and features are described in Dataset and Methods chapters also.

- **Coordinate Systems**. The coordinate system used for this project is Cartesian (following the ISO 8855 standard). It is important to remember, though, that the data from each sensor is reported according to each sensor's polar coordinate system. Furthermore, appropriate translation and rotation matrices are applied to the already transformed polar magnitudes (using range r and azimuth angle ϕ) into Cartesian. After the corresponding transformations, the position of the vehicle and radar with respect to the space's objects is finally given by 3 Cartesian coordinate systems as shown in Figure 1.3. The ones used in this thesis are the vehicle-fixed car coordinate system (cc) and the sensor coordinate system (sc); the global coordinate system (gc) is not considered since only the relative positions with respect to the car and sensor were needed.

- **Ego-motion Compensation**. As mentioned before, the Doppler effect is used to estimate the radial velocity of the objects with respect to the sensor. In this case, the vehicle is not stationary with respect to the surroundings of the object (and therefore, neither is the attached sensor). Thus, the Doppler value has to be compensated for the ego-motion. The Doppler over ground (or *ego-motion compensated Doppler velocity*) $\hat{v}_r$ is calculated subtracting the radial velocity of static objects v_r^{static} from the measured Doppler velocity v_r. Thus, $\hat{v}_r = v_r - v_r^{\text{static}}$.

- **Clutter** This term is used to describe measurements with undesired information that could lead to false positives or overlay. The objects themselves produce the clutter as the radar collects the waves. There are three categories for which a reflection is cataloged as clutter.

  1. **Mirrored Objects**. They correspond to the objects that bounce back electromagnetic waves due to they act as electromagnetic mirrors. Objects like metallic fences or doors can show a different range or angles in the radar point cloud.
  2. **Ambiguities**. The radars are susceptible to ambiguities such as output wrong values of azimuth angle ϕ resulting in wrong positioning of the points later.
  3. **Noise**. As with any other electronic sensor, the radar measurements contain false detections or *noise*. The data is filtered and processed to characterize the noise and generate reliable datasets without this noise. This characterization is particularly crucial for the radar since the readings are noisier and appear more frequently than in cameras or Lidars.

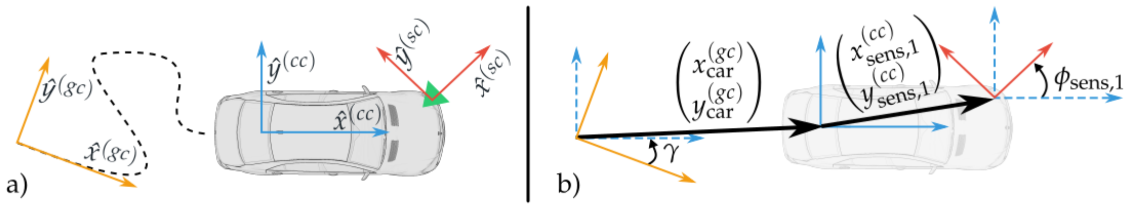

$\hat{y}^{(gc)}$
$\hat{x}^{(gc)}$
$\hat{y}^{(cc)}$
$\hat{x}^{(cc)}$
$\hat{y}^{(sc)}$
$\hat{x}^{(sc)}$
a)
b)
$\begin{pmatrix} x_{\text{car}}^{(gc)} \\ y_{\text{car}}^{(gc)} \end{pmatrix}$
$\begin{pmatrix} x_{\text{sens},1}^{(cc)} \\ y_{\text{sens},1}^{(cc)} \end{pmatrix}$
γ
$\phi_{\text{sens},1}$

Figure 1.3: Position and orientation of the three coordinate systems. A black dashed line describes the trajectory of the vehicle. a) gives an overview of the three systems. b) The resulting vectors are formed considering the yaw angle γ and the azimuth angle $\phi_{sensor,1}$ of the first sensor.

These terms can be deepened in [40] for better understanding and for the curious reader.

1.2.4 Radar Point Clouds

In the broadest sense, a point cloud is a set of points with n dimensions distributed in a settled format. Usually, 2D and 3D point cloud representations are plotted and the rest of dimensions are usable as extra features. The point cloud is the basic unit used for the two object detection models studied in this work.

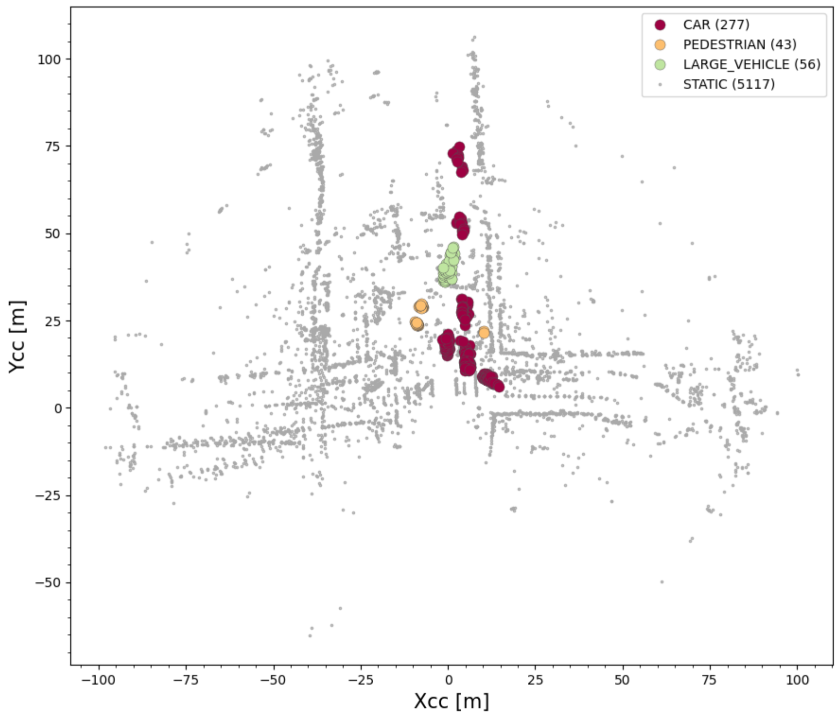


Figure 1.4: Representation of a 2D radar point cloud with its extra features not plotted. Labels per point included in this plot.

Thus, as in a point cloud, the radar point cloud is simply an adaptation of the format where each point is encoded with the radar features (mentioned in 1.2.2) across space and time. i.e., a selected time frame will populate a radar point cloud from a particular dataset. An example of a couple of frames gathered from a radar dataset is shown in Table 1.1 and the whole 2D representation of one entire point cloud in Figure 1.4.

Naturally, there are differences between how the studied models will process the enclosed information available from the given multi-dimensional points in the point cloud, but, in the end, this will be the format from which each model will gather its input needs.

Table 1.1: Example of 2D radar data points and features gathered from a small number of frames and from the RadarScenes dataset

Frame	x [m]	y [m]	$\hat{v}_r$ [m/s]	RCS [dBsm]	r [m]
0	0.7247	-2.9067	-7.3799	5.5794	3.5734
1	0.6688	-3.7162	-6.4936	-7.4465	4.1290
2	3.7217	-14.3565	0.0421	-19.5192	13.4836
3	3.6685	-17.8650	0.0602	-14.5649	16.9920
4	6.3278	-11.3200	0.0228	0.2069	10.7815
5	6.3725	-13.6340	-0.02237	3.1928	13.0454

1.3 Outline of the Thesis

This section displays an overview of the entire report structure and how it is outlined. It also shows how to move across the chapters.

Chapter 2 describes a detailed theory and key concepts about the pre-requisites that support the two different models and their corresponding methods. It also introduces theory about the evaluation metrics supported and shared for both studied models.

Chapter 3 starts by comparing different initially considered datasets and their properties. Next, an analysis of radar properties and the data structure and annotations are made. Finally, it explains how the data is formatted and will be transferred to each model in the form of snippets.

Chapter 4 is where the methods are assembled for the corresponding models and the followed methodology used for each. It contains their pipelines sustained by mathematical principles and references from the Theory chapter. It also describes the specific filters and adaptations for correctly managing the data across the models.

Chapter 5 presents the results from each model in given conditions. The sets of experiments and their results are described chiefly in tables and analyzed according to the appropriate metrics.

Lastly, chapter 6 is a summary of the insights gathered from the results in chapter

5. The initial conclusions, practical implications of this work on the real world, and how these implications can be advantageous for future works.

2

Theory

In this chapter, we first introduce some basic concepts in deep learning as a necessary prerequisite for the subsequent sections. Then we describe the algorithms' theoretical foundations, i.e., YOLO v3, PointNet (and PointNet++), and DBSCAN, that we will implement. Finally, we will introduce the evaluation metrics that measure the performance of our deep neural networks on point clouds, namely pointwise-IOU, mAP and F1 score.

2.1 Multilayer perceptrons

As one of the Deep feedforward networks, *multilayer perceptrons (MLPs)* are the foundations of deep learning models. The purpose of a feedforward network is to approximate the outcome to a desired function f^*, [18].

To understand the feedforward network, let f^* be a classifier function such that $y = f^*(\boldsymbol{x})$ maps an input $\boldsymbol{x}$ to a category y. Then, a feedforward network will define a mapping such that $\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ are the learned parameters which result in the best approximation of f^*.

The term feedforward is named after the sequential flow the input information is exposed to. i.e., the input information $\boldsymbol{x}$ goes through the intermediate computations that approximate f to the output $\boldsymbol{y}$, which is the approximated (predicted) value(s). The term *feedforward* indicates that there are no feedback connections among the layers of the network. Otherwise, the term would be *recurrent neural network*, which exists but it is out of the scope of this thesis.

The term network is given to feedforward networks because they are represented as interconnected connections of many different functions. The common representation is a directed acyclic graph with defined interconnections between functions. A simple example to illustrate a network is $f(\boldsymbol{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\boldsymbol{x})))$, which is a chain structure. Thus, $f^{(1)}$ is the *first layer*, $f^{(2)}$ is the *second layer*, $f^{(3)}$ is the *third layer*, and so on if there are more terms. Additionally, the term *deep* is defined by the depth or the number of layers the network has. This depth will define (among other parameters) how good the function approximation of the original function will be. The model's parameters are referred to as weights and biases, where the biases are intercept terms that add constant values to each unit for each connection, and the weights indicate the degree to which the value of a specific unit influences the values of related units.

Approximate the outcome to a desired function f^* implies that the input $\boldsymbol{x}$ must produce a value close to y. Thus, the behavior of the output y and the input data $\boldsymbol{x}$ are given, i.e., that f^* is not given, but y and $\boldsymbol{x}$ are. However, this does not specify the behavior of what the intermediate layers (functions) should do. Instead, the learning algorithm decides the operation of each intermediate layer in order to achieve the approximation stated. Then, the definition of *hidden layers* goes to every intermediate layer that is not specified by the input $\boldsymbol{x}$ nor output y.

Each hidden layer necessitates the selection of an *activation function* to compute each n hidden layer values functions $f^{(n)}$. Thus, a hidden unit in the hidden layer will contain an activation function that will dictate how well the data approximates the mapping from the input to the output. The role of each unit in the hidden layer resembles a neuron because each one of them will use data to *learn* from the data provided through the activation function. The term *neural* is also acknowledged because these units are fully connected between layers and hidden layers and can be seen in Figure 2.1.

In the present work context, both studied models have in their most basic structures one or several feedforward networks. The way each model decides to interconnect its neurons is described and analyzed later in the Methods chapter. The interest in this chapter is that both perform a multi-classification task.

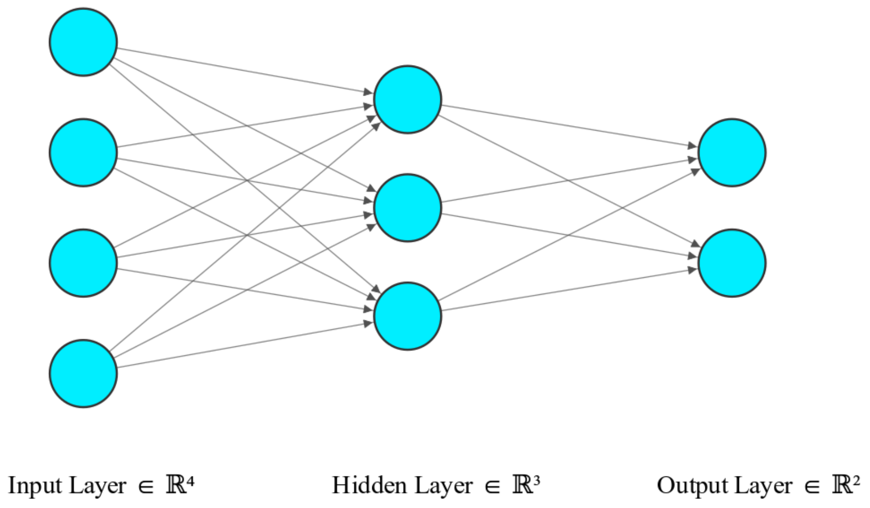

Input Layer $\in \mathbb{R}^4$
Hidden Layer $\in \mathbb{R}^3$
Output Layer $\in \mathbb{R}^2$

Figure 2.1: An example of a simple feedforward neural network called a multilayer perceptron. There are three fully connected layers and one "hidden" layer. The rightmost units could represent probabilities for two classes or be real numbers regressing two quantitative outputs.

2.1.1 Convolutional Neural Networks (CNN)

The *convolutional neural networks (CNNs)* are one specialized kind of neural network for processing data with a known topology that resembles a grid. An example is the image data, which can be thought of as a 2D grid of pixels, [18].

The term *convolutional* is given since the network employs a mathematical operation called convolution. Goodfellow et al. [18], state: Convolution networks are neural networks that use the convolution operation, i.e., the integral of the product of two functions (one is reversed and shifted). The convolution is extrapolated for matrices in at least one of their layers.

The convolution in neural networks has three important advantages that improve a Machine learning model. These are sparse interactions, parameter sharing, and equivariant representations. The convolution can work with inputs of any variable size. One can deepen on these terms in [18].

There are three stages in a typical layer of a convolutional network; see Figure 2.2. In the first stage, several convolutions are performed in parallel in the layer to produce a set of linear activation functions. In the second stage, each linear activation is run through a nonlinear activation function. This stage is sometimes called the detector stage. In the third stage, a pooling function modifies the output of the layer further.

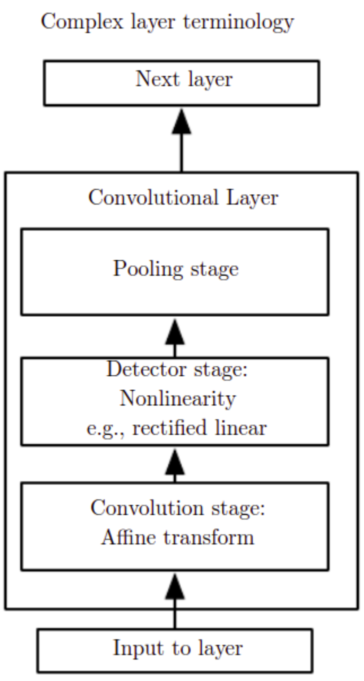


Figure 2.2: Typical convolutional neural network layer. Here, the convolutional network is viewed as a small number of relatively complex layers, and each layer is populated with many stages. In this terminology, there is a one-to-one mapping between kernel tensors and network layers.

A pooling function replaces the network's output at a certain layer location with a statistic compilation of the nearby outputs. E.g., the *max pooling* operation reports the maximum output within a neighborhood. In all cases, pooling helps make the representation approximately invariant to small input translations. The term invariance to translation means that if a translation in the input by a small amount will not change the values of most of the pooled outputs.

2.1.2 Training

The training goal of the learning algorithms is that from a given data set containing features associated with labels (targets), such algorithms can learn the approximated function from that data set. Furthermore, the challenge is extended by evaluating the trained network model on new unseen input data. This ability is called *generalization.*

Since typically the training data set is accessible, the output can be compared against that training data and generate an error measurement. This error is called *training error*, and now, part of the performance can be measured. Then, a second error is calculated, called *generalization error* or *test error* where the performance is better measured.

Normally, the whole data set for a specific problem is given. In order to generate the test (not seen by the algorithm) set and training (used by the algorithm to learn) set to calculate the errors mentioned above, a *data-generating process* is needed. Here, *independent and identically distributed (i.i.d.)* assumptions are taken. These assumptions state that the samples from both data sets are independent (not correlated) from each other and that both should be identically distributed. In real life, achieving such assumptions is difficult; however, they do not necessarily need to meet these conditions to their fullest. A general knowledge of the shape of the data set, followed by a splitting taking into account these criteria, is just enough.

Thus, the two factors that determine how well a machine learning algorithm performs are:

- Reduce the training error as much as possible.
- Reduce the gap between the training and test set errors.

The consequence of poor performance is not getting enough low loss error on the training data. This is called *underfitting.* The opposite case occurs when the gap between the training and test errors is too large. This is called *overfitting.* The ability to modify the model behavior in the underfit/overfit spectrum is called *capacity.* Thus, models with low capacity will have issues fitting the training data set. Models with high capacity can memorize the training data set properties, which are not really when used in the test data set. All these concepts can be seen in Figure 2.3.

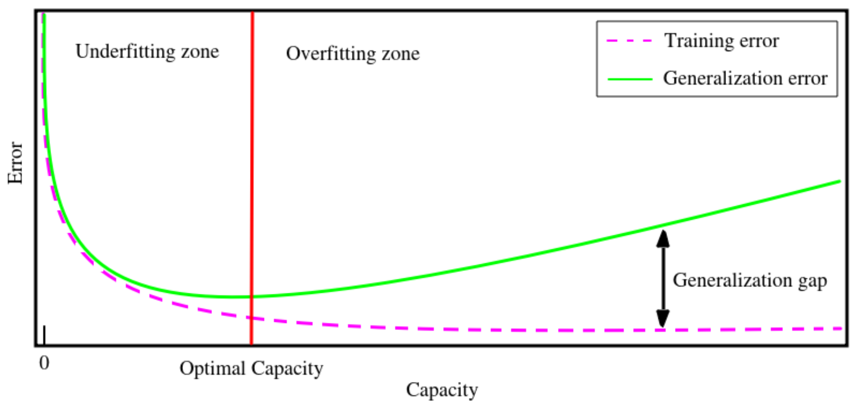


Figure 2.3: Typical behavior in training and test data set loss errors, respectively. It is seen that after the red line, the gap increases, indicating overfitting. The opposite is underfitting, and it is located before the red line.

2.2 Classical Detection and Instance Segmentation

Instance segmentation and object detection refer to the task of segregating objects out of visual environments and is an important area of computer vision and machine learning research.

The next subsections contain descriptions of the ways the detection/segmentation tasks are pursued by each network architecture and what are the bases upon which these were devised, sustained, and built.

Here, only the neural network approaches are defined. Cluster formation (and more specifically, the DBSCAN algorithm [10]) is, in some sense, an instance segmentation task. Nevertheless, since it is not a trainable neural network but an algorithm, its foundations do not fit into the following definitions and are described later in this chapter. It is mentioned to avoid confusion because it complements and enhances the output of the semantic segmentation network in this project.

2.2.1 Object Detection

Object detection is a computer vision technology that attempts to localize objects of a specific class, making use of the inherent features of the objects, e.g., the task of face identification, where facial features are sought (distance between eyes, eyes' color, etc.) in images. Usually, localization means finding the bounding boxes of the target objects. Specifically, in this work (**first model, YOLO**), the object detection task is accomplished by a neural network that retrieves localized bounding boxes and associated class probabilities per object found within the search space (image), see Figure 2.4.

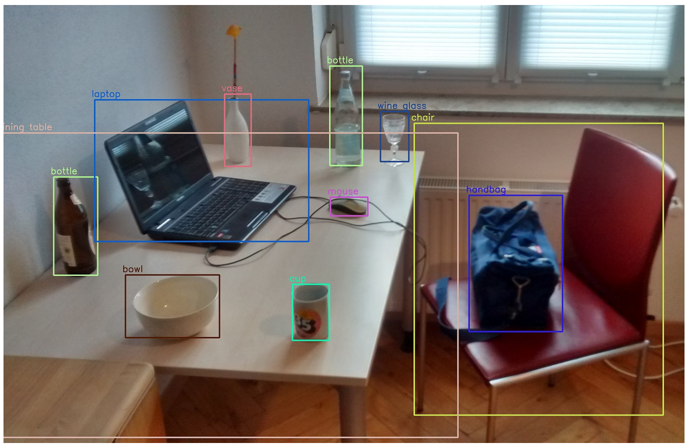


Figure 2.4: YOLO object detection example (Source: MTheiler).

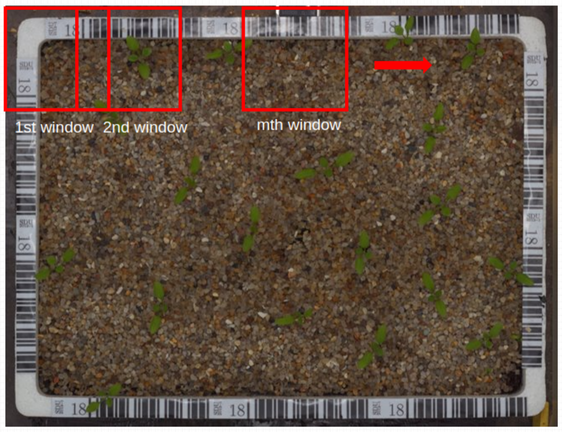


Figure 2.5: An example image from the Plant seedling dataset [15]. Red rectangles are the sliding windows.

2.2.2 Sliding Window Detection

Given an image classifier, e.g., a CNN model in 2.1, a natural question is what we should do to localize interesting objects in the image. Sliding window, in a straightforward way, played an important role in object detection. In a computer vision context, a sliding window is a rectangle region with a fixed size that slides across an image. We apply the image classifier to each such region to tell if there are interesting objects. Figure 2.5 illustrates such a process. Combined with image pyramids, i.e., shrinking the image into a sequence of scales, we can detect objects in images at various scales and locations.

On the other hand, the disadvantages of sliding window detection are also obvious: the image classifier has to infer the same number of times as the number of sliding windows. The quantity of windows proliferates when there are objects on different scales. Thus, the time complexity becomes unacceptable. The recently-merged algorithms mentioned in 2.4 offer a decent way to reduce the time complexity.

2.2.3 Semantic and Instance Segmentation

Segmentation of image points is splitting a set of points into multiple image segments of the same category, also known as regions or objects (in digital image processing and computer vision, this corresponds to splitting into sets of pixels). Segmentation aims to make a picture more intelligible and more straightforward to examine by simplifying and changing its representation. Segmentation is often used to locate objects and boundaries from points within the space the process is acting on, [42].

This process is usually divided into *semantic segmentation* (Figure 2.6 (a)) and *instance segmentation* (Figure 2.6 (b)), where semantic segmentation is the technology which detects for each point, the object category (label) it belongs to, and *all* the object categories must be known to the model. In contrast, instance segmentation performs a deeper characterization than semantic segmentation by partitioning two objects (instances) with the same label; this means that the algorithm computes certain surrounding correlations between objects and differentiates them. Thus, semantic segmentation does not distinguish different instances of the same category, while instance segmentation can identify them individually.

Since there are many methods for performing instance segmentation over data, a simple approach turned out to be efficient enough to segregate the different instances of objects in this work (**second model, PointNet++ + DBSCAN**). Thereby, a semantic segmentation (based on neural networks) followed by a clustering algorithm will be the followed path in order to sort out instances of objects in this project.

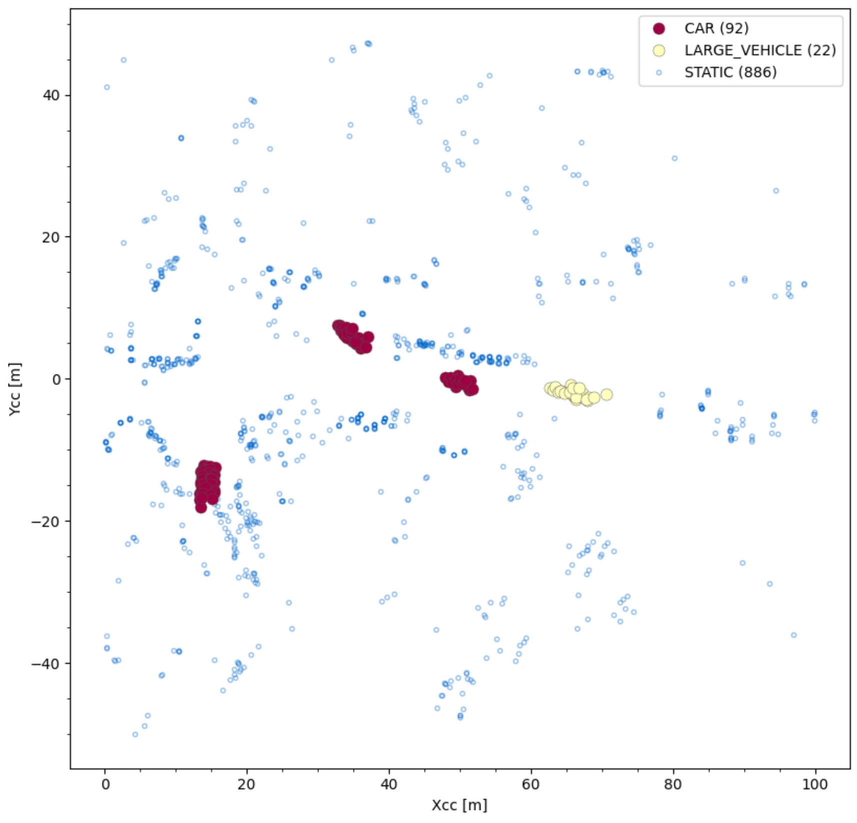
2.3 Clustering Analysis

The term *clustering* includes all the tasks of grouping a set of objects with similar properties or ruled by some criteria. Thus, each group of objects found in a search space by some cluster analysis is defined as a *cluster* of objects. In this search space, there can be several clusters or none depending on the criteria selected, i.e., small distances within cluster members, dense areas of the data space, intervals, or particular statistical distributions.

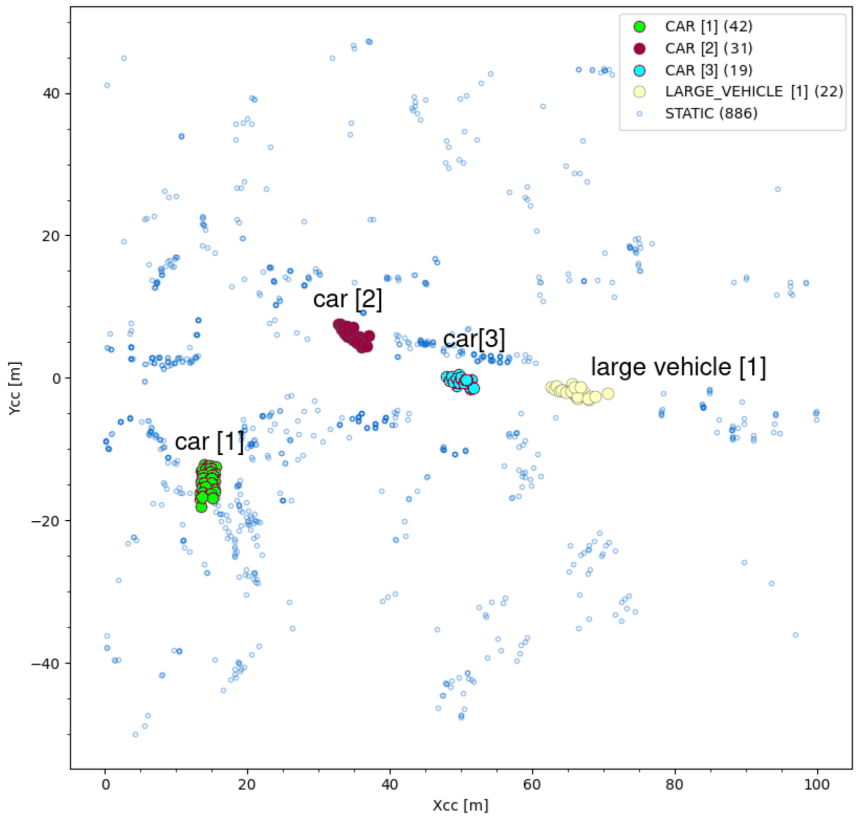
It belongs not to one type of algorithm but to a general task to be solved. Moreover, it can be achieved by different algorithms that can understand the concept of clustering differently and the way they group the data into clusters. In section 2.6 a popular algorithm for clustering (and the one used in this thesis) is defined and described.

2.4 YOLO

As a state-of-the-art representative in a one-stage detector, YOLO [34] is famous for its end-to-end architecture and short inference time. Unlike two-stage detectors that rely on region proposals, YOLO treats the detection as a regression problem and solves localization and classification problems simultaneously. It dexterously utilizes the nature of convolution layers to divide the image into $S \times S$



(a) *Semantic Segmentation*



(b) *Instance Segmentation*

Figure 2.6: Semantic segmentation and instance segmentation performed over radar point clouds.

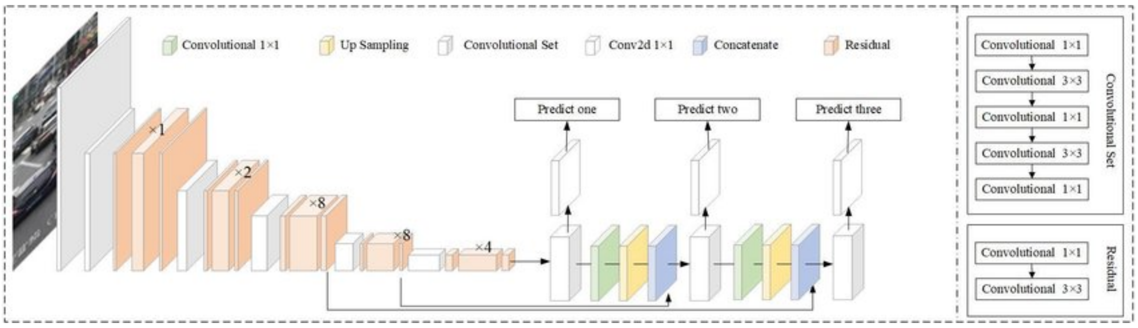


Figure 2.7: YOLO v3 architecture: Darknet 53 as the backbone feature extractor and three prediction heads for various scales.

grid and regress on each grid cell. The output is B bounding boxes, confidence for those boxes, and C class probabilities. Each bounding box is in the format of $\langle x, y, width, height, class, confidence\rangle$, where x, y define the center of the box, *width, height* are the size of the box, *class* shows the category and *confidence* shows the percentage confidence that there is an object in the predicted bounding box.

Figure 2.7 illustrates the architecture of YOLO v3. YOLO v3 utilizes Darknet-53 as the backbone feature extractor. Darknet-53 is a convolution network with 53 layers and residual connections, which resemble ResNet [20] structure. The input images' size has to be an integer multiple of 32. To handle multi-scale detection, the input image is down-sampled 8, 16, and 32 times to generate features at three different scales. Each scale corresponds to a prediction head that outputs bounding boxes.

YOLO v3 uses anchors to help the computation in the prediction heads, i.e., the boxes are output as the offset from the anchor box sizes. There are three anchors in each regression head, thus nine in total, which are pre-defined by K-means clustering on all the bounding boxes in the training set.

To train YOLO v3, the loss function contains three components, namely an object loss, a classification loss and a localization loss. C_i is the ground truth class in cell i and $\hat{C}_i$ is the predicted class labels. The loss is formally expressed as

$$\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{I}_{ij}^{obj}\left[(x_i-\hat{x}_i)^2+(y_i-\hat{y}_i)^2\right] \tag{2.1}$$

$$+\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{I}_{ij}^{obj}\left[(\sqrt{w_i}-\sqrt{\hat{w}_i})^2+(\sqrt{h_i}-\sqrt{\hat{h}_i})^2\right] \tag{2.2}$$

$$+\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{I}_{ij}^{obj}(C_i-\hat{C}_i)^2 \tag{2.3}$$

$$+\lambda_{noobj}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{I}_{ij}^{nobj}(C_i-\hat{C}_i)^2 \tag{2.4}$$

$$+\sum_{i=0}^{S^2}\mathbb{I}_{i}^{obj}\sum_{c\in classes}(p_i(c)-\hat{p}_i(c))^2, \tag{2.5}$$

Where 2.1 and combined are the localization loss; 2.3 and 2.4 combined are the classification loss; 2.5 is the object loss. The loss is a weighted sum of sum-squared errors. $\lambda_{coord} = 5$ and $\lambda_{noobj} = .5$ are weights that force the network to prioritize localization loss and also avoid 2.4 overpowering the gradient because most of the grid cells do not contain any objects. $\mathbb{I}_i^{obj}$ denotes if object appears in cell i and $\mathbb{I}_{ij}^{obj}$ denotes that the jth bounding box predictor in cell i is in charge for that prediction. $\mathbb{I}_{ij}^{nobj}$ denotes that the jth bounding box predictor in cell i is not in charge for that prediction, i.e., the jth bounding box predictor does not have the biggest IOU with the ground truth. The indicator functions intend to only punish classification error if there is an object in the grid cell and punish only bounding box error if the predictor is responsible for the ground truth box, i.e., has the highest IOU with the ground truth among all B predictions.

2.5 Deep Learning Segmentation and Clustering on Point Clouds

In order to understand the fundamentals of instance segmentation using semantic segmentation and a clustering algorithm, three algorithms are studied: PointNet, PointNet++, and DBSCAN.

With the neural networks named *PointNet* [32] and *PointNet++* [33], Qi et al. introduced novel architectures capable of operating on raw point clouds. In this project, the *semantic segmentation* network presented in PointNet++ (and therefore, also PointNet) is used.

The cluster formation algorithm proposed by M. Ester et al. [10] called *DBSCAN* is selected since it performs over any set of points in a defined space (in this case, a point cloud set) given. The result is clusters defined in such a set of points.

2.5.1 PointNet

The point cloud format tends to be difficult to process for neural networks because they usually only accept structured and even data formats, e.g., in YOLO, the format unit is an image with even pixels. Thereby, a neural network capable of dealing with this raw format is advantageous. Thus, PointNet tackles the problem by consuming unordered point sets as inputs directly into the architecture and producing outcomes regarding recognition tasks, including object classification, part segmentation, and semantic segmentation, i.e., either classification scores or categorical classification.

Formally, PointNet is a unified neural feedforward network architecture that accepts point clouds as input and produces class labels for the full input or per point segment/part labels for each point in the input. The network learns a set of optimization functions/criteria from areas in the point cloud and encodes them into features. The network's last fully connected layers combine these optimum learned values into a global descriptor (max pooling) [32] for either classification or segmentation. This pipeline is shown in Figure 2.8.

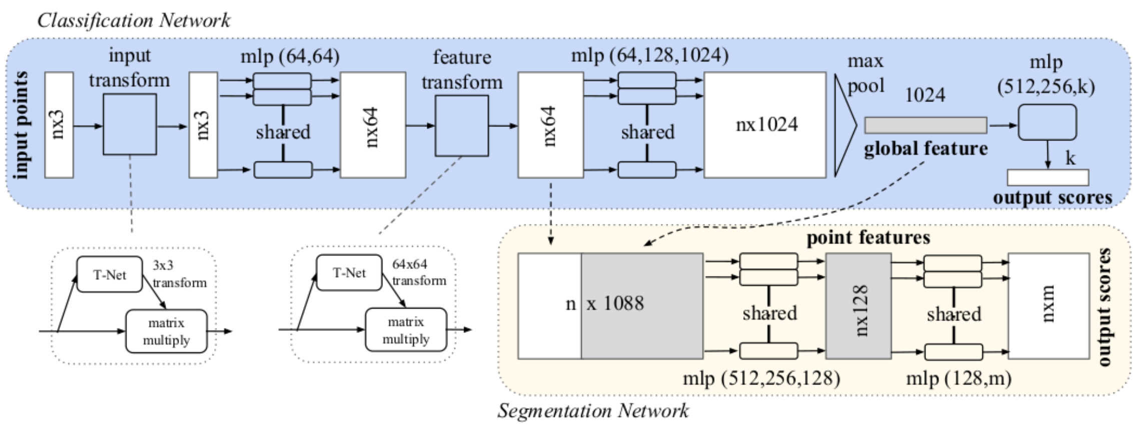


Figure 2.8: PointNet pipeline. The input and feature transformations are shown as well as the max pooling operation for getting features. Two branches are shown: segmentation and classification, being segmentation an extension of the classification network. MLP are multi-layer perceptrons and the numbers, their layer sizes. The image is taken from the paper of Qi et al. [32]©IEEE 2018, Fig.2.

There are three key elements that for which PointNet is able to perform:

1. Point clouds have an unstructured format, and any neural network that is fed with N point sets needs to be invariant to $N!$ permutations on the input set feeding order.

    - This is achieved by using symmetric functions (specifically, *max pooling*, Figure 2.9). Such symmetry for a value given dependent on n input arguments, is the same regardless of the order of the arguments, e.g., $f(x_1, x_2) = f(x_2, x_1)$. Consequently, a global feature vector is produced, which can capture an aggregate signature of the n input points. The procedure works by stacking feature vectors to form a matrix; with the rows as the feature vectors and the columns as their values, a picking-out action is performed to retrieve the maximum value of each column. Thus, a vector with the maximum values is generated, which supposedly contains all the information about the point cloud, i.e., an $n \times m$ input matrix is reduced to a vector of length m after max pooling (Equation 2.6).

2. Points in a points cloud belong to a space with distance metrics, i.e., they are not isolated, and neighbors groups meaningful subsets. This model has this behavior because it works by capturing local structures from neighbor points interactions. The aforementioned learned global descriptor (vector) (learned in 1.) is produced without errors if the point set does not contain isolated elements.

3. The learned representation of the point set should be invariant to certain geometric transformations, i.e., the global point cloud category should not be modified by rotations/translations of any kind.

    - PointNet solves the invariance by applying an affine transformation to the input coordinates. The output is a transformation matrix dependent on

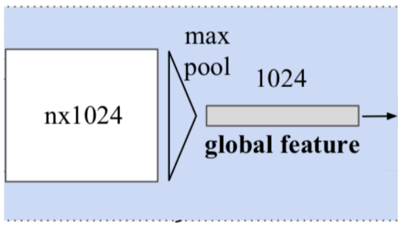


Figure 2.9: Symmetric function of max pooling in PointNet.

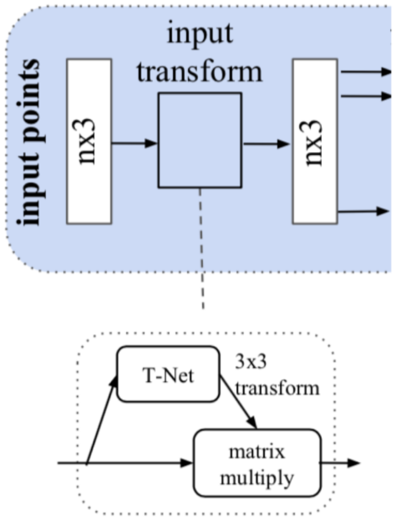


Figure 2.10: Input transform in PointNet.From [32] ©IEEE 2017, Fig.2.

the input. This small transformation network (often called T-net) works in the same way as PointNet (with point independent feature transformation, max pooling for feature aggregation, and finally fully connected layers) by getting an independent feature transformation matrix of 3×3. The matrix size comes after a computation on each of the n input points that are represented as a vector; these input points are mapped to the embedding spaces independently. Applying a geometric transformation amounts to a matrix multiplying each point with a transformation matrix, see Figure 2.10.

$$\text{Feature\_vector} = \max_i(\boldsymbol{X} = (x_{ij}) \in \mathbb{R}^{n \times m}) \quad , \quad \text{Feature\_vector} \in \mathbb{R}^m. \tag{2.6}$$

2.5.2 PointNet++

PointNet++ uses the same principle of max pooling for extracting features as in PointNet. However, it differs in the sense that PointNet++ provides the ability to capture local features at different scales. This feature is achieved by applying a max pooling function to the data in a hierarchical fashion and in a number of sets of abstractions. Thus, the network can extract finer details in densely sampled data regions.

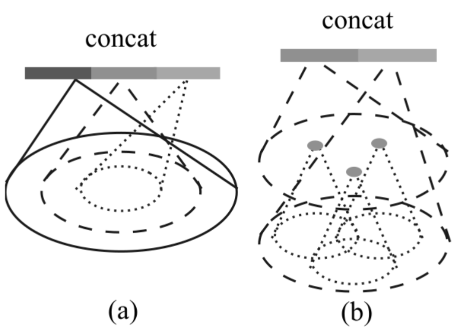


Figure 2.11: (a) Multi-Scale Grouping (MSG), (b) Multi-resolution Grouping (MRG). Image from paper of Qi et al. [33]IEEE 2018, Fig.3.

For each set of abstraction layer, three tasks are performed:

- Sampling: Selects a subset of centroids using a *iterative farthest point sampling (FPS)* technique. Another less efficient method is random sampling.
- Grouping: Selects a group of points for each centroid (KNN or ball-query methods applied here). Since the points are sampled from a metric space in the previous step, the neighborhood of a point will be defined by a metric distance.
- PointNet: Run PointNet (feature extraction) on each group found.

The local patterns extraction at multiple scales (according to local point densities) in each set of abstraction is achieved by any of the two types of the proposed density adaptive layers: *Multi-Scale Grouping (MSG)* (Figure 2.11a) and *Multi-resolution Grouping (MRG)* (Figure 2.11b). In MSG, a PointNet feature extraction layer is applied to each scale selected, then these features (at different scales) are concatenated to form a single multi-scale feature. In MRG, the PointNet feature extraction layer is applied first to all the points in the lowest region (right concat vector in Figure 2.11b) and then to the features in the subsequent sub-regions (left concat vector in Figure 2.11b).

The complete process for two sets of abstractions is depicted in Figure 2.12. The pipeline followed for the current project is segmentation. After interpolating the feature vectors obtained from the sets of abstractions, the outcome is the predicted class label for each point fed into the network. The interpolation is achieved by a *Feature Propagation* layer where a particular feature $f(x)$ in a layer corresponds to the inverse distance weighted average of the k nearest neighbors' features in the previous layers. Next, the interpolated features are skip-link concatenated to the respective set abstraction feature.

2.6 DBSCAN Clustering Algorithm

The general clustering algorithm *DBSCAN (Density-Based Spatial Clustering of Applications with Noise)* proposed by Ester et al. [10] is a density-based clustering non-parametric algorithm, i.e., In a point cloud $P \in \mathbb{R}^n$, the algorithm can group

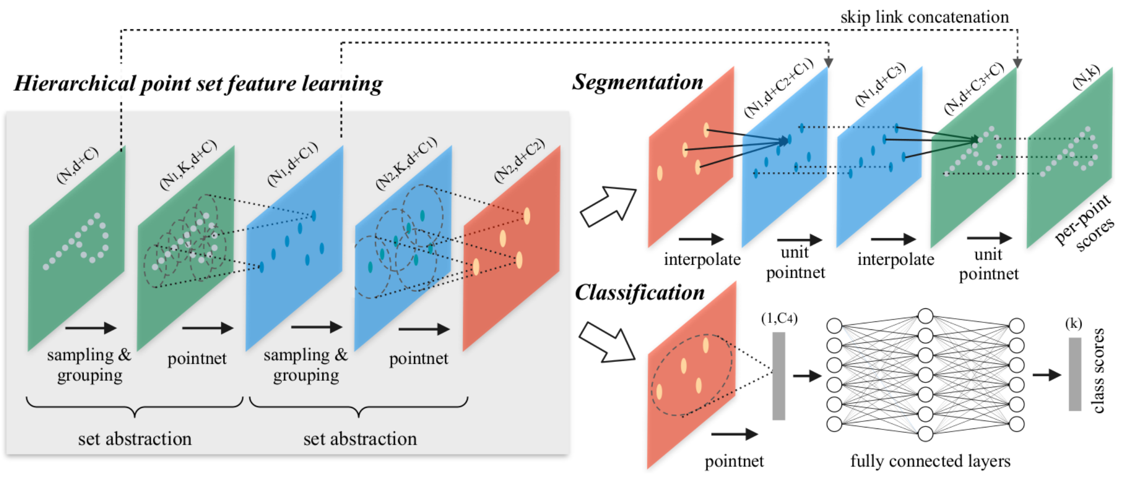


Figure 2.12: The PointNet++ hierarchical feature learning architecture description. Both applications (segmentation and classification) are shown in the bifurcation. The example shown for 2D points. Image from the paper of Qi et al. [33] ©IEEE 2018, Fig.2.

the points in regions where they are densely near to each other and let outliers lie alone in regions where the density is low, as shown in Figure 2.13.

The parameters that govern the algorithm are the minimum number of points $minPts$ and the neighborhood radius for each point ϵ. Each point is classified as: *core point*, *density-reachable point*, or *outlier (noise)*. The algorithm decides the outcome for each point and cluster them according to these guidelines:

- A point p is a core point if a minimum of points defined by $minPts$ are within the radial distance ϵ, including itself.
- A point 1 is density-reacheable from core point p if q is within radial distance ϵ.
- The points that are not reachable within the distance ϵ from other points are cataloged as outliers or noise. They are also cataloged as the same if they do not meet the condition of $minPts$. Note that it does not suffice to share radius ϵ between points but to contain the point within this radius.

The algorithm executes these computations for the entire set of points one by one until every point is classified and, therefore, clustered. Each cluster meets the following two properties:

- All the points in the cluster are density connected among them.
- A point is part of the cluster when is density-reachable without being a core point.

2.7 Evaluation Metrics

For evaluating both models, the selected common metrics are defined in the next sections: Point-wise IoU, Average Precision (AP), Mean Average Precision (mAP), and the *harmonic mean of precision and recall* better known as F1 score.

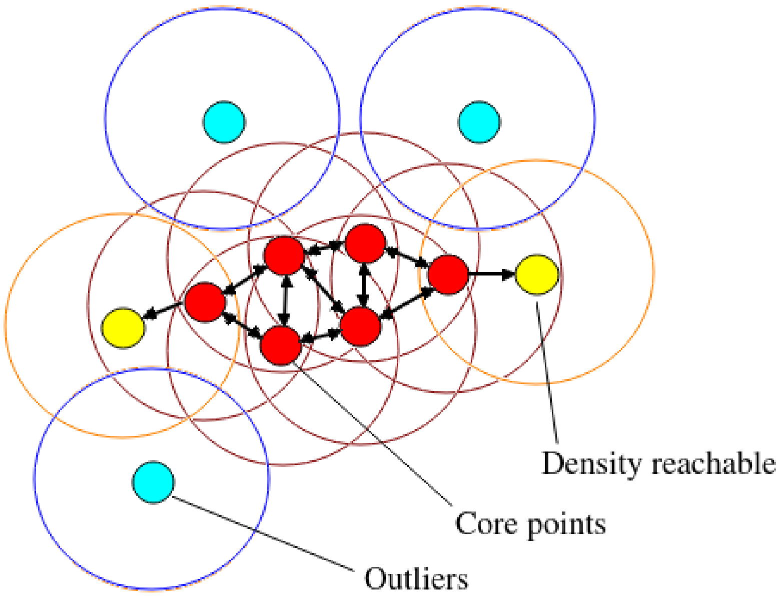


Figure 2.13: Representation of a cluster formation using DBSCAN. Here, $minPts = 3$ with a certain ϵ. Red points are core points since they meet both requisites. Yellow points only meet one of the criteria; therefore, they are density-reachable points. The blue point does not meet any directive, and it is considered an outlier or noise. The double direction arrow indicates that the points share both conditions successfully, while the one direction arrow shows that only one of the conditions was met.

2.7.1 Point-wise IOU

The point-wise IoU metric is defined by the area ratio formed by the intersecting points over the union points of two clusters [46]; these clusters are the predicted cluster and the ground truth cluster, respectively, (Equation 2.7), i.e., true positives points over the sum of true positives points, ground truth points, and false positive points, visualized in Figure 2.14. Thus, it measures the performance of the model by comparing the output of the model against the corresponding known target. One should not confuse the semantic segmentation IoU metric since the one used here takes into consideration the instance metric of each object (cluster or bounding box) and not only the labels per point.

$$IoU = \frac{|\text{predicted points} \cap \text{true points}|}{|\text{predicted points} \cup \text{true points}|}. \tag{2.7}$$

An object instance is defined as matched if a prediction has an IOU greater or equal than some threshold.

2.7.2 Average Precision per class (AP) and Mean Average Precision (mAP)

The mean average precision (mAP) and Average Precision (AP) per class are popular metrics used to measure the performance of models doing object detection tasks. The first task prior calculating AP or mAP is to compute the *precision* and the *recall* metrics.

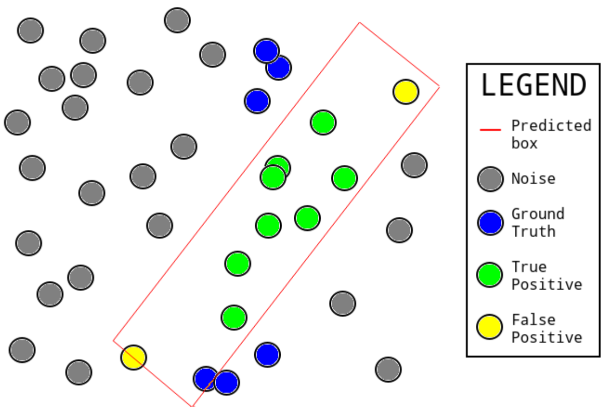


Figure 2.14: Point-wise IoU visualization. It depicts a predicted bounding box to illustrate the object cluster that is containing the predicted object.

Precision (Equation 2.8) measures how accurate the predictions are. i.e., the percentage of the correct predictions. For points, it is the ratio of true positive points (TP) and the total number of predicted positives (true positives and false positives).

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}. \tag{2.8}$$

Then, there is recall (Equation 2.9), which measures how well the model found all the positives. For points it is defined as the ratio of true positives (TP) and the total of ground truth points.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{2.9}$$

Note that FP, in Equation 2.8, stands for False Positives (Points wrongly predicted as part of the object), and FN, in Equation 2.9, stands for False Negatives (Points that failed to be predicted as part of the object).

The AP is then able to be calculated by taking the area under the curve formed by the precision and the recall. Usually, the recall values are segmented into eleven parts: $Recalls = r = [0, 0.1, 0.2, \ldots, 0.9, 1]$ and shown in Equation 2.10. Then, the maximum value of the precision is taken. The precision value is an interpolated precision that takes the maximum precision over all recalls greater than r.

$$\text{AP} = \frac{1}{6} \sum_{r \in \{0, 0.1, \ldots, 1\}} \max_{\text{Recall}(c) \geq r} \text{Precision}(c). \tag{2.10}$$

After obtaining the AP metric, the mAP is available for being obtained as well. In Equation 2.11, the AP scores are macro averaged, i.e., the AP score is calculated for each object class, and once they are all computed, the mAP is calculated by taking the mean from all of them.

$$\text{mAP} = \frac{1}{\tilde{K}} \sum_{\tilde{K}} \text{AP} \tag{2.11}$$

where, $\tilde{K} = K - 1$ is the number of object classes

2.7.3 Harmonic Mean of Precision and Recall (F1 score)

The F1 score is the metric that allow us to know how balanced are the False Positives (FP) against the False Negatives (FN) measurements of the model's predictions and the corresponding ground truth values. The Precision (Equation 2.8) which is related to the FPs, and the Recall (Equation 2.9) which is related to FNs are computed using the harmonic mean of them two. Thus, we can know the performance of the model when both, Precision and Recall, are performing at their best.

Formally, for a class k, we get the F1 score (Equation 2.12) using Equation 2.8 and Equation 2.9 to get the maximum F1 score at that class.

$$\mathrm{F1}_k = \max_c \frac{2}{\frac{1}{Pr(c)} + \frac{1}{Re(c)}}. \tag{2.12}$$

3

Dataset

In this chapter, we give an introduction to the dataset that each model is trained on. First, popular public datasets are compared to find the most suitable one. Second, details about the chosen dataset, RadarScenes [39], are listed. Lastly, we introduce the preprocessing for generating training samples.

3.1 A Comparison

With the rapid development of autonomous vehicle industry, famous automotive data sets such as KITTI [13] and Cityscapes [6] contains camera and some lidar data. A few new data sets that includes radar data are published recently, e.g., nuScenes [3], RadarScenes [39], CARRADA [30], and CRUW [44]. Although the number is still increasing, most of them are only applicable to the tasks of camera-radar fusion instead of our task, which is radar-only object detection.

The nuScenes data set is the first dataset that cotains the full autonomous vehicle sensor suite. It provides 3D bounding boxes as annotations and also 2D radar point clouds. However, it is not suitable for our task due to the sparsity of radar points. In [8], the author discusses that 72% of ground truth objects contain minimum two radar points in nuScenes, whereas only 43% of ground truth objects contain minimum four radar points. Compared with RadarScenes, it provides much denser 2D radar point cloud. RadarScenes is a newly published large data set that uses traditional automotive radar settings. It covers most of common traffic scenes, such as urban driving, commercial areas, country road and high ways. It comprises over 4 hours and 100 km of driving.

Other than point cloud dataset, spectral datasets are also taken into consideration. As described in [30], Range-Doppler-Azimuth data cube is the raw data before filtered by CFAR algorithm to get point cloud. Compressing the data cube along each dimension, we can respectively attain range-Doppler spectrum, range-azimuth spectrum and Doppler-azimuth spectrum. These spectra that contain positional and motion information could be the training samples for deep neural networks. CARRADA [30] and CRUW [44] are also recorded by conventional automotive radar systems. CARRADA contains small amount of data because it is recorded on a test track to reduce noise. Therefore, it could not cover various traffic scenarios. On the contrary, CRUW is a large dataset that covers multiple road types. However, it only contains range-azimuth spectral without Doppler information, which is usually vital for automotive applications.

See Table 3.1 for the comparison of four datasets. Except for the four dataset

Table 3.1: Overview of popular public radar data sets

	nuScenes	RadarScenes	CARRADA	CRUW
Sensor settings	Low Res Automotive	Automotive	Automotive	Automotive
FOV	360°	290°	180°	
Range	250m	100m	50m	
Range Resolution		0.15m	0.2m	0.23m
Doppler Resolution	0.1 km/h	0.1 km/h	1.5km/h	
Azimuth Resolution		$0.5° - 2°$		0.7°
Total Length	1500h	4h	21.2min	3.5h
Num_{seq}	1000	158	30	464
Sequence length	20s	13s-240s		
Traffic scenes	++	++	-	++
Class Number	23	11/5	3	3
Data Type	Point Cloud	Point Cloud	R-D, R-A spectra	R-A spectra
Annotation	3D bounding box	point-wise labels	spectral annotations	spectral boxes (19%)
Sparsity	++	+	None	None

[1] 'None' means the relevant information is not available.
[2] FOV: field of view, R-A: range-azimuth, R-D: range-Doppler, Num_{seq}: number of sequences

listed, the Oxford Radar Robot-Car data set [2] is also considered. It also carries the full sensor suite and consists of data from a total amount of 280 km of urban driving. However, the FMCW radar it used works in a nontraditional way, which is being mounted on the top of the vehicle and rotates 360°.

As the result of the above analysis, we decided to use RadarScenes given the following merits:

1. It provides relatively dense radar point cloud and the automotive radars are mounted in a traditional way as in ADAS;

2. It covers various scenarios and road types, e.g. urban driving, commercial areas, country road, high ways and etc., and has large amount of data;

3. It offers point-wise annotations that allows the training of both semantic instance segmentation and object detection models;

4. It contains 11 fine categories and remapped 5 coarse categories that are more balanced.

3.2 RadarScenes Introduction

3.2.1 Sensor Settings

RadarScenes is a large multi-class data set with bird's eye view point clouds and some reference images. Measurements is collected by an experimental vehicle equipped with four 77 GHz automotive radar sensors on the front bumper. The range of each one is up to 100 meters and the field of view(FOV) covers ±60°. See Fig 3.1 for the sensor settings. The sensors' range resolution and radial velocity resolution are specified by the manufacturer to be $\Delta_r = 0.15m$ and $\Delta_v = 0.1km/h$ respectively. Similar to the other automotive radars, the angular resolution drops when the distance to the sensor's boresight increases. It drops from $\Delta_\phi(\phi = \pm 0°) = 0.5°$ to $\Delta_\phi(\phi = \pm 60°) = 2°$.The sensor cycle is $60ms$ and four sensors working cycles are interleaved.

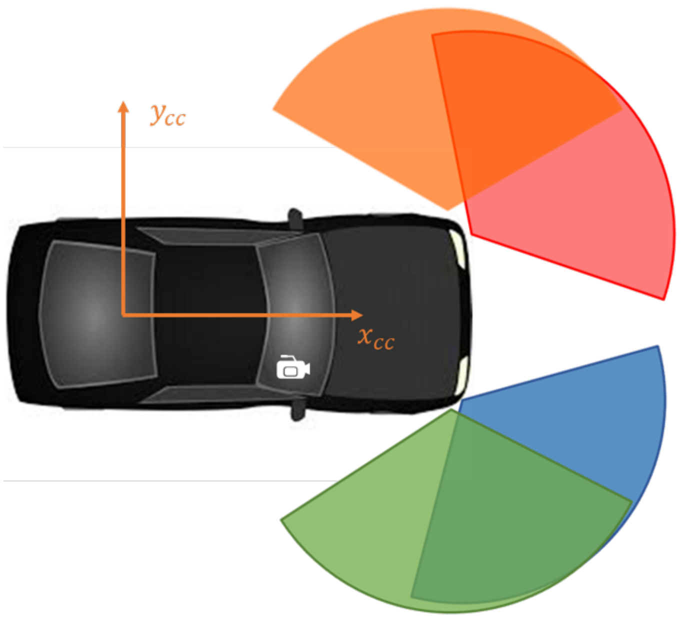


Figure 3.1: The car coordinate(cc) is marked orange. All the sensor measurements are relative to it. Four radar sensors are mounted at the front face close to the lights. The documentary camera is mounted behind the windscreen(white). The FOV (±60°)of each radar sensor is marked in different colors. From [39]©IEEE, Fig.3.

Furthermore, there is a documentary camera mounted behind the windscreen and facing the direction of traffic. The ego-vehicle's motion information is also recorded, e.g., position, direction, velocity and yaw rate. Using this data, the radar measurements can be compensated for ego-motion and transformed into a global coordinate system. See details in Section 3.4.1.

3.2.2 Data Structure

RadarScenes data set consists of 158 sequences, each of which corresponds to a recording cycle of the sensor suite. They covers different traffic scenarios and last from 13s to 4 minute. Each sequence contains a hdf5 file that stores "radar_data". With open-source tools in [39], it can be loaded as a numpy structured array, the structure of which is like a table with each row corresponds to a unique radar reflection point. A reflection point has the following attributes, each being a column of the table:

- timestamp: in micro seconds relative to some arbitrary origin
- sensor_id: 1, 2, 3, 4, id of the sensor that recorded the point
- range_sc: in meters, radial distance to the reflection
- azimuth_sc: in radians, azimuth angle to the reflection
- rcs: in dBsm,radar cross section of the reflection
- vr: radial velocity relative to the ego-vehicle
- x_cc and y_cc: in meters, position of the reflection in ego-vehicle coordinate
- x_seq and y_seq in meters, position of the reflection in the global coordinate with an arbitrary start point
- uuid: unique tag for the point
- track_id: id of the dynamic object this points belongs to. Empty, if it belong to "static" class
- label_id: class id, a integer from 0 to 11

3.2.3 Annotations

The *label_id* specifies the semantic class with an integer, whereas the *track id* identifies unique real-world objects over the entire data collection period, i.e., all the reflection points belonging to the same objects in a sequence share the same *track id*. Besides, all the points belonging to the same object also share the same *label id*. Since all the points belong to the same object can be visualized as a cluster of points, "cluster" and "object" are interchangeable in this report. With these two labels, points in the same cluster can be extracted with its class information, which forms the basis of the ground truth samples. This feature is vital for semantic instance segmentation models.

RadarScenes focuses on dynamic road users, therefore, all the objects are categorized

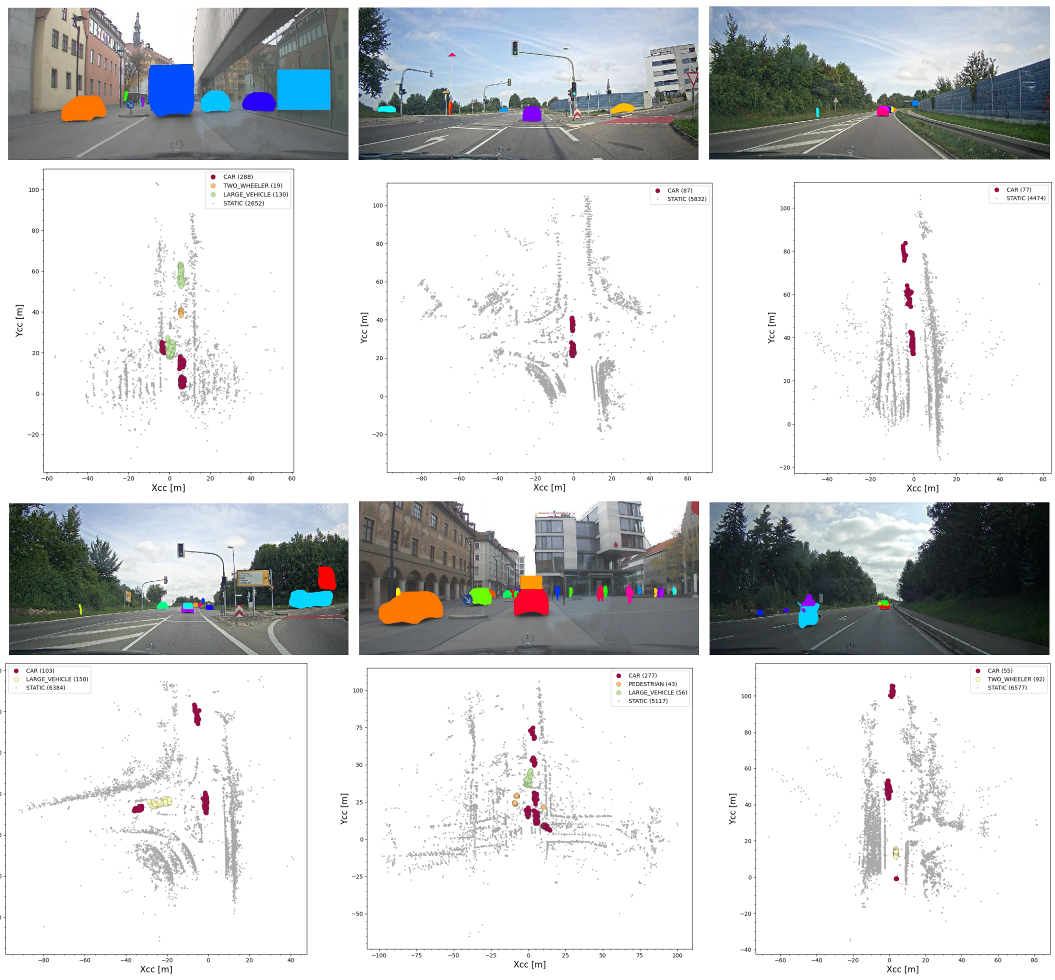


Figure 3.2: Up Left : Urban driving; Up Middle: Cross section; Up Right: Highway; Bottom Left:T section; Bottom Middle: Crowded area; Bottom Right: Motorbike

into eleven classes, namely: **car**, **large vehicle**, **truck**, **bus**, **train**, **bicycle**, **motorized two-wheeler**, **pedestrian**, **pedestrian group**, **animal** and **other**. The **other** class contains various types of dynamic road users that cannot be categorized into all the other classes, such as skaters. The **pedestrian group** contains objects in which individual pedestrians cannot be separable. What is more, huge amount of reflection points belong to static objects that are not road users, thus **static** label is assigned to them. On top of regular classes, there is a mapping function provided in [39] that maps the eleven classes into five more coarse classes, including CAR,LARGE VEHICLE,TWO-WHEELER,PEDESTRIAN,and PEDESTRIAN GROUP. The alternative class system leads to a more balanced sample distribution, which is desirable for the tasks of object detection.

In order to comply with GDPR, other road users are anonymized by repainting. This feature is unfavorable for sensor fusion task, but has no impact on our task.

3.3 Dataset Analysis

All sequences amount to a total of 4.3 hours driving. There are 118.9 million reflection radar points in RadarScenes, more than 90% of which are static points. There are only 832822 points that belongs to dynamic objects. There are 1260280 dynamic objects in all the frames, however, there are only 7516 unique dynamic objects in the whole data set.

Figure 3.5 illustrates the distribution of time intervals between two succession frames. The maximum time interval is 1799404 microsecond; the minimum time interval 1 microsecond; the median time interval is 15358 microsecond; the most frequent time interval is 1 microsecond. The numbers indicate that four radar sensors' working cycles are not synchronized. In Section 3.7, 500 milliseconds snippets are extracted from various sequences. Consequently, the number of frames in a 500 milliseconds snippet also varies. The maximum number of frames is 55, the minimum number of frames is 21, the median number of frames is 27 and the most frequent number of frames is 27. Thus, the conclusion is that a snippet consist of roughly 27 frames.

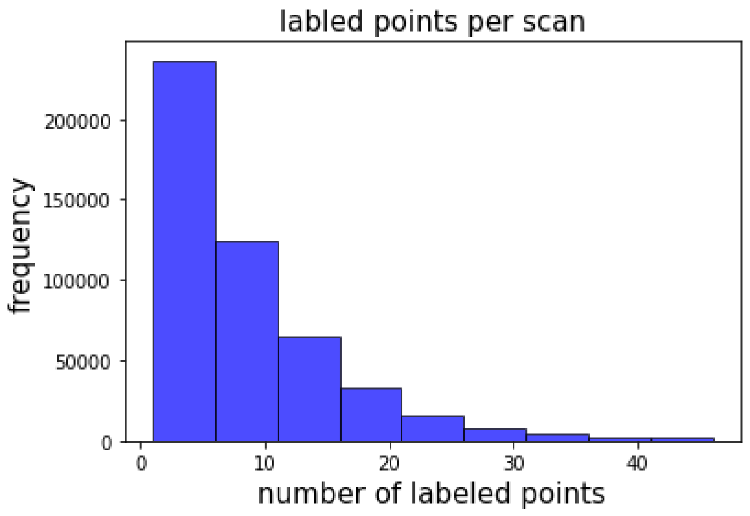


Figure 3.3: Histogram for the number labeled points in each frame. Most frames contains less than 10 labelled points, which is quite sparse and requires accumulation over time to improve the density.

3.4 Snippet Extraction

To begin with, let us define terminologies:

- Frame: The set of points that share the same time stamp. Its data is from a single sensor scan;
- Snippet: The temporal sequence of frames for a specific time length. This length is chosen as 500 ms in this thesis. It is a training and test sample;
- Sequence: A temporal sequence is collected during a working cycle of the sensor suite. Usually they corresponds to various traffic scenes in the real world. There are 158 sequences in RadarScenes, the length of which varies.

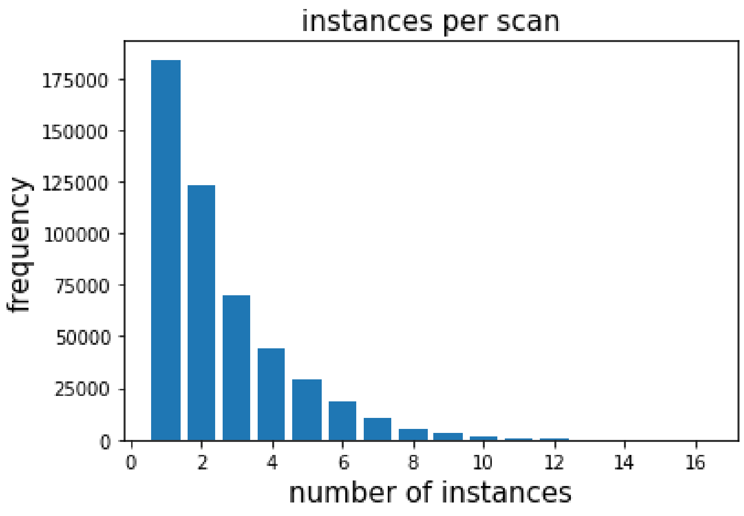


Figure 3.4: Histogram for the number of objects in each frame. Most frames contains less than 3 objects.

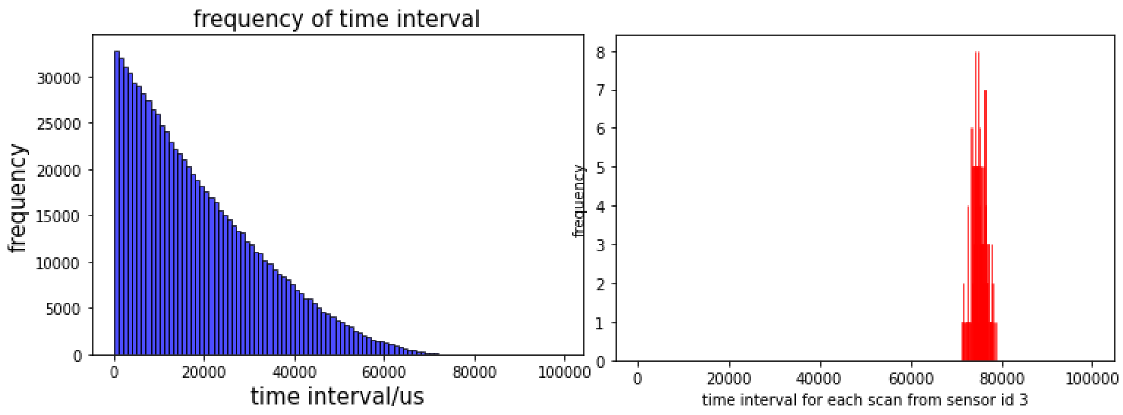


Figure 3.5: Left:Histogram for time intervals between two succession sensor scans. The frequency decreases as the interval length increases, which shows the fours sensors' cycles are not synchronized. Right: Histogram for time intervals between two succession sensor scans in sensor NO.3. All the time interval locates in a narrow band, which verifies that the working cycles lasts around 60 ms.

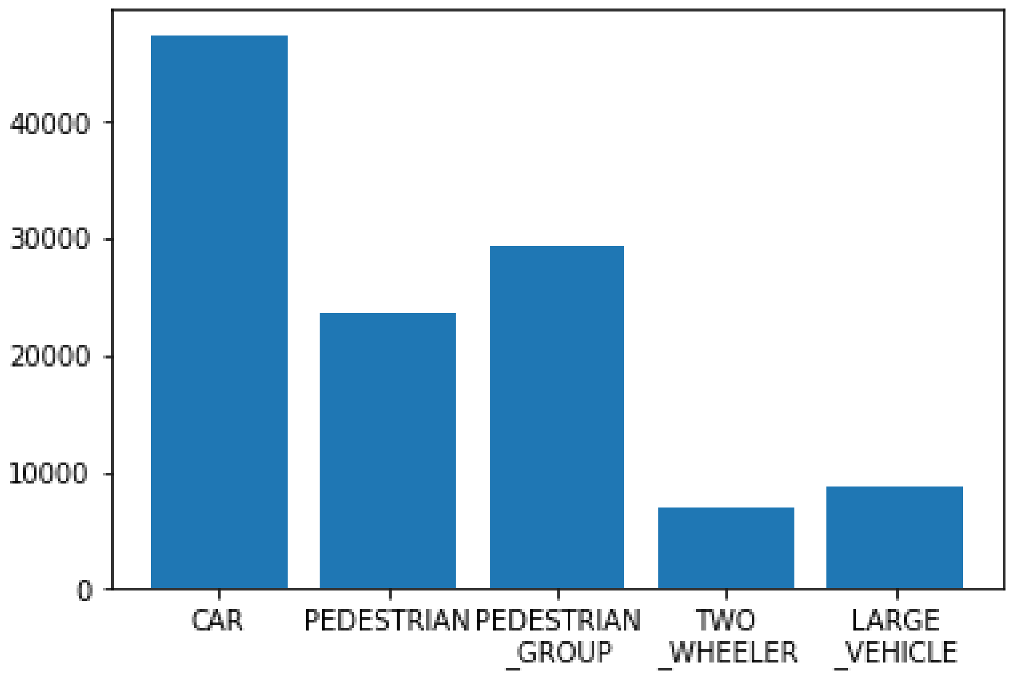


Figure 3.6: Distribution of number of instances of each class

One frame is an overly sparse sample to train a deep neural network on. Accumulation over time is a natural way to increase the reflection points' density. In this paper, we follow [8] and extract 500-milliseconds snippets, which can be preprocessed (for example grid mapping) or directly used as input for deep neural networks. The extraction procedure is described in Figure 3.7. Firstly, we iterate over all frames $f_i, i = 0, 1, ..., n$ in a sequence and count the time interval between two frames Δt_j. Time intervals are different because four sensors are not synchronized. The iteration does not end until the accumulation of interval reaches 500 milliseconds, which can be expressed as

$$\arg\min_n(|\sum_0^n \Delta t_j - 500ms|). \tag{3.1}$$

Second, the snippet is clipped to remove outliers that are not located in the range $-50m \leq y_{cc} \leq 50m, 0m \leq x_{cc} \leq 100m$.Third, we check if there are labelled targets in the snippet because clipping will lead to the loss of some target clusters and there are some snippets that even does not contain any targets before clipping. Finally, we iterate over all the target clusters and remove the clusters without volume, i.e., target clusters that contains only one point or two points. This can ease the task of image-based detector by removing the bounding boxes whose widths or heights are zero. Note that this step is not mandatory for the training of point cloud-based detectors, however, it is still implemented for the benefit of comparison between different detectors.

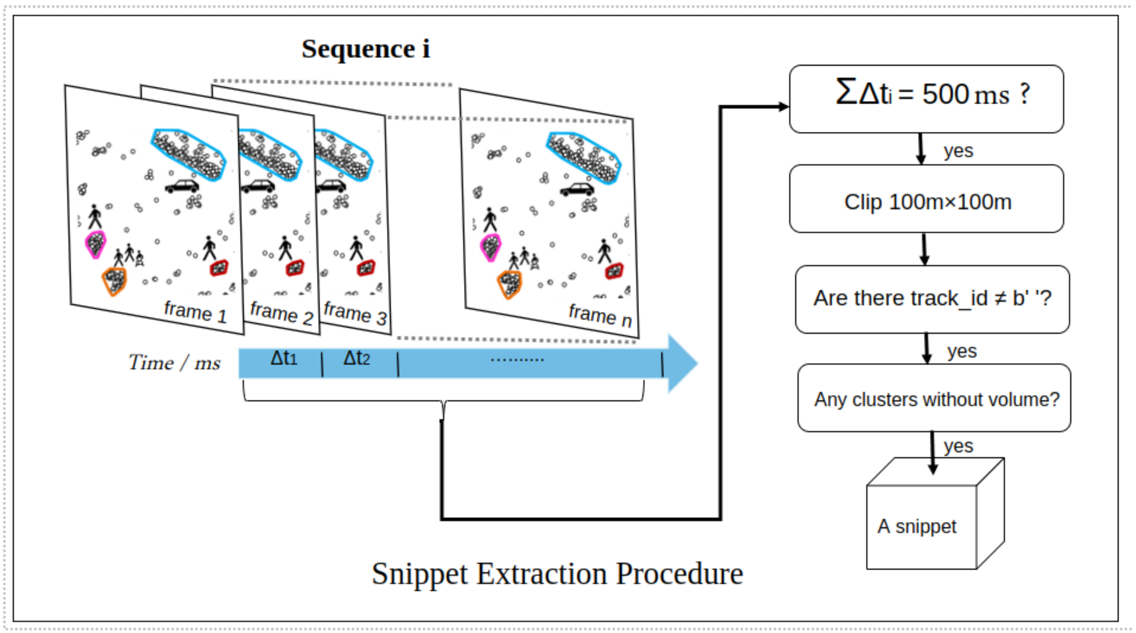


Figure 3.7: Snippet Extraction: 1. accumulate frames over time to reach 500 milliseconds; 2. Clip the snippet to remove outliers; 3. Check if there is no target in the snippet. If so, go to the next snippet; 4. Remove the clusters that has no volume, i.e., clusters contain only one or two points.

3.4.1 Ego-motion Compensation

All the reflection points are in the ego-vehicle coordinate, i.e., their range and azimuth are relative to the center center of the rear axle of the ego vehicle as depicted in figure 3.1. Simply stacking all the points in the same coordinate will cause mistakes because the vehicle itself keep moving, i.e., the origin of ego-vehicle coordinate keeps moving. Therefore, before treating the snippet as a training sample, coordinate transfer is mandatory. It transfer all the points in different frames into the coordinate of the first frame in the snippet. Theoretically such a transform requires a rotation matrix R_{ij} and a translation vector t_{ij}. i, j means transform frame f_j to the coordinate of f_i. To construct R_{ij}, t_{ij}, we need to know the odometry information of the sequence, such as how far the vehicle has moved forward and how much the yaw angle of the ego vehicle has changed. The compensated frame f_j^{comp} is given by

$$f_j = \begin{bmatrix} x_{j1} & y_{j1} \\ x_{j2} & y_{j2} \\ \vdots & \vdots \\ x_{jn} & y_{jn} \end{bmatrix},$$
$$f_j^{comp} = R_{ij} f_j + t_{ij}. \tag{3.2}$$

After the transformation, f_j^{comp} and f_i can be stacked together as one snippet. For this task we directly used the transform tool provided by [39].

What is more, the Doppler radial velocity also needs compensation. The car is moving at varying speed. Therefore, the measured Doppler velocity is the radial relative velocity, which will inevitably be affected by ego-motion. Take the signed

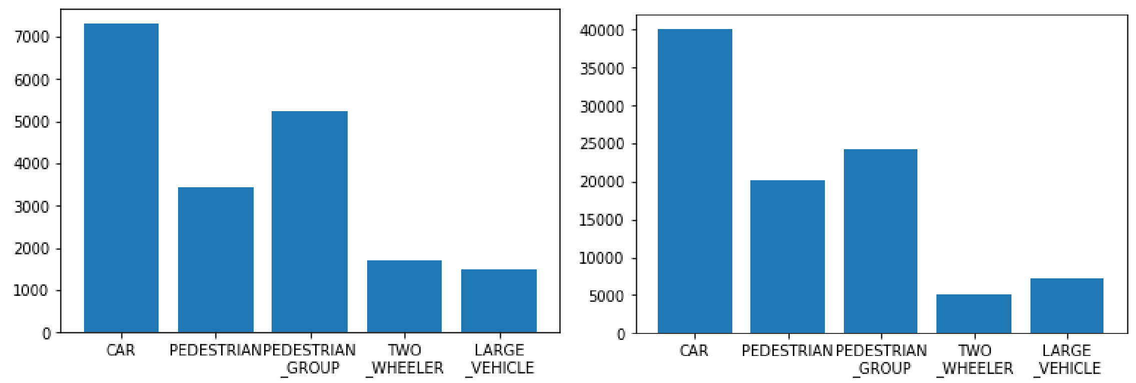


Figure 3.8: Left: The class distribution on the test set; Right: The class distribution on the combined set of the training set and test set.

absolute velocity, v_{ego} and the yaw rate of the ego-vehicle from the center of the rear axle, $\dot{\Phi}_{ego}$ into consideration. Given Doppler velocity v_r and its azimuth angle Φ, the compensated Doppler velocity $\widetilde{v_r}$ is given by

$$\widetilde{v_r} = v_r - \begin{bmatrix} v_{ego} + m_y \dot{\Phi}_{ego} \\ m_x \dot{\Phi}_{ego} \end{bmatrix}^T \begin{bmatrix} \cos(\Phi + m_\Phi) \\ \sin(\Phi + m_\Phi) \end{bmatrix}, \tag{3.3}$$

where m_x, m_y are the mounting position of the radar sensor, m_Φ is the rotation from the middle axis of the car. The first term represents the absolute velocity at the sensor, whereas the second term extracts the radial components along the direction Φ.

3.4.2 Train, Validation, Test Sets Split

There are 27595 snippets in total extracted from 158 sequences. As in [8] 158 sequences are split in to the training set, the validation set, and the test set by the ratio 64 : 16 : 20. The data set splitting is done with respect to each sequence, i.e., all the snippets in one sequence stays in the same split. This is because targets may resemble each other between different samples of the same sequence.

In [8], a brute force was applied to decide the best split. In this thesis, we study class distributions in both the training set and the test set and ensure that they have roughly the same distribution by manual inspection. Figure 3.8 presents the distributions. Compared to Figure 3.6, we can observe that the general distribution is maintained.

4 Methods

Two Deep Learning approaches were studied in this project. The first studied model is the image-based object detector combined with grid mapping. In this section, the aforementioned approach and the instance segmentation model: PointNet++ semantic segmentation network and radar Grid DBSCAN clustering, are addressed and explained.

Both approaches have, according to [36], good results in detecting dynamic road users from radar data points. There are some other methods which have been not considered because their efficiency are considerably low compared to these two methods. This is the case of PointPillars [24] applied to radar point clouds [36].

The baseline pipelines for both models is described thoroughly in a diagram in Figure 4.1.

4.1 Image Object Detection Network: YOLO with Grid Maps

Image-based 2D object detection is a well-developed field for years, whose mainstream approaches can be divided to one-stage detector, represented by YOLO [35], FPN [27], and two-stage detector, represented by Fast RCNN [14]. Both approaches basically detect 2D objects on an image with axis-aligned bounding boxes (AABB) and reached breaking performances on large-scale open datasets.

Inspired by recent advances in image-based detection, it is natural to preprocess point clouds into image-like datatype and utilize image-based object detector to process it. For static objects, point clouds can be accumulated over time to get occupancy grids.The same preprocessing can be used in dynamic object detection, however, on the contrast, the time length of accumulation should be limited to avoid object smearing.

Any object detection model, regardless of one-stage or two-stage, theoretically works fine on handling grid maps. Here YOLO v3 is adopted as in [8] because of its end-to-end architecture and real-time inference capability. To get the training inputs and targets, some preprocessing steps needs to be applied on the raw data from

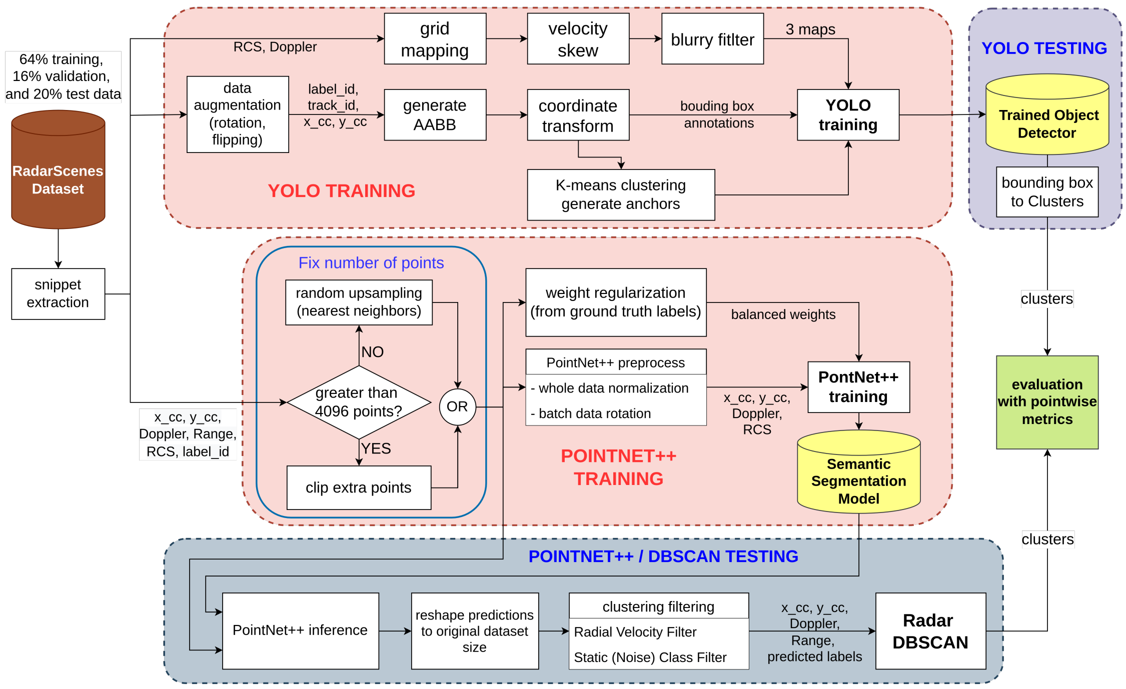

64% training, 16% validation, and 20% test data
RadarScenes Dataset
snippet extraction
RCS, Doppler
grid mapping
velocity skew
blurry filter
3 maps
data augmentation (rotation, flipping)
label_id, track_id, x_cc, y_cc
generate AABB
coordinate transform
bouding box annotations
YOLO training
K-means clustering generate anchors
YOLO TRAINING
YOLO TESTING
Trained Object Detector
bounding box to Clusters
clusters
evaluation with pointwise metrics
Fix number of points
random upsampling (nearest neighbors)
NO
greater than 4096 points?
OR
YES
clip extra points
x_cc, y_cc, Doppler, Range, RCS, label_id
weight regularization (from ground truth labels)
balanced weights
PointNet++ preprocess
- whole data normalization
- batch data rotation
x_cc, y_cc, Doppler, RCS
PontNet++ training
Semantic Segmentation Model
POINTNET++ TRAINING
clusters
POINTNET++ / DBSCAN TESTING
PointNet++ inference
reshape predictions to original dataset size
clustering filtering
Radial Velocity Filter
Static (Noise) Class Filter
x_cc, y_cc, Doppler, Range, predicted labels
Radar DBSCAN

Figure 4.1: Baseline pipelines for the two studied object detection models using RadarScenes data set.

RadarScenes, including bounding box extraction, coordinate transfer, grid mapping, Doppler velocity skew, blurry filter and data augmentation. Figure 4.1 illustrates the pipeline.

4.1.1 Bounding Box Extraction

Radarscenes does not provide 2D bounding boxes as a type of annotations, instead it provides point-wise class labels and track IDs. All the points with the same track ID in a frame belong to the same instance, thus semantic instance segmentation is possible combined with class labels. As Figure 4.2 shows, an instance is represented as a cluster of points, the bounding boxes of which can be extracted as the exterior rectangle. Both axis-aligned bounding boxes (AABB) and oriented bounding boxes (OBB) are generated. Besides, RadarScenes does not contain ground truth bounding boxes, thus the generated boxes may be different from the contour of the object. For example, most of the reflection points may be located in the side of the objects that is close to the radar. Furthermore, accumulation over time also elongate the shape of the cluster. This is probably a reason why bounding box detection is rarely adopted in handling dynamic radar point cloud.

An AABB is defined by four parameters: (x, y, w, h), where x, y are the coordinates of the box center and w, h are the width and height of the bounding box, whereas an OBB has an extra element as the yaw angle θ. AABB is the relatively easy one to extract: Firstly, given a cluster of points, take the minimum and maximum horizontal coordinates and vertical coordinates (x_1, y_1, x_2, y_2). Second, calculate parameters in (x, y, w, h). See 4.2 for an example.

An OBB takes an extra parameter, the yaw angle, to define the orientation. In

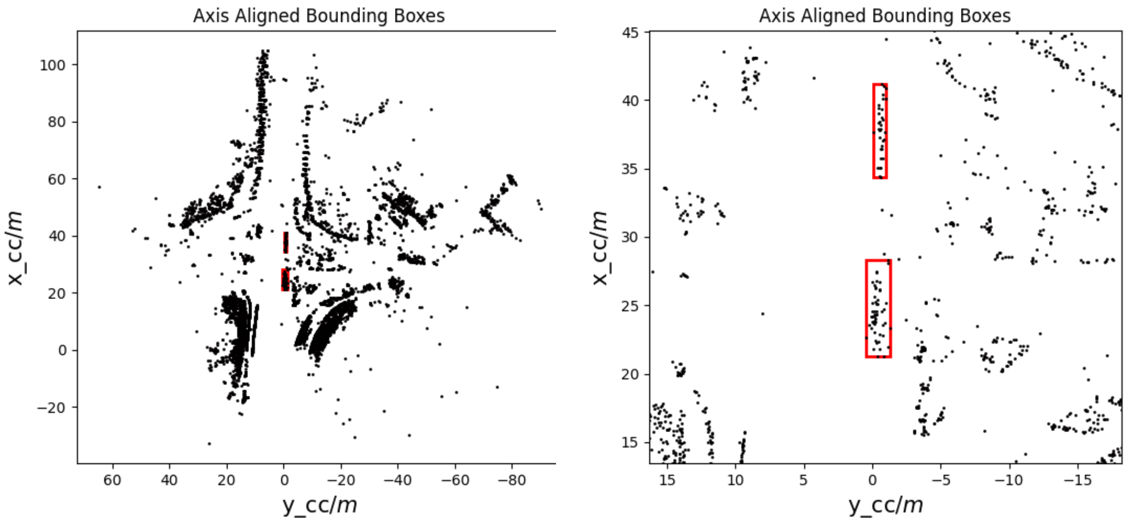


Figure 4.2: Sequence 109 start index 0 number of future frames 28; the left is the whole snippet, the right one is enlarged axis aligned bounding boxes

other words, it is defined by (x, y, w, h, θ). The aim is to find the minimum exterior bounding rectangle given a cluster of points. See Figure 4.3 for an example. Among several ways to reach this goal, here principal component analysis (PCA) is adopted. Firstly, the (self)covariance matrix of all the points is constructed. This can be expressed as

$$\begin{aligned} X &= \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \end{bmatrix}, X \in \mathbb{R}^{N\times 2}, \\ C &= cov(X) = E[(X - \overline{X})^T (X - \overline{X})], \end{aligned} \tag{4.1}$$

where N is the number of points in the cluster, x, y are the coordinates for a point and $\overline{X}$ is the stacked matrix of mean coordinates. Since $cov(X)$ is a real symmetric matrix, it always has two real eigenvalues $\lambda_1, \lambda_2 \in \mathbb{R}$ and their corresponding eigenvectors $v_1, v_2 \in \mathbb{R}^{2\times 1}$, which suggests the first and second principal directions of variances. We can stack the eigenvectors as a rotation matrix. This can be written as in

$$\begin{aligned} Cv_1 &= \lambda_1 v_1, \\ Cv_2 &= \lambda_2 v_2, \\ R = [v_1, v_2]^{-1} &= [v_1, v_2]^T. \end{aligned} \tag{4.2}$$

With R, we can rotate the cluster to an orthogonal basis as in

$$X' = RX, \tag{4.3}$$

i.e., the two main variation directions are horizontal and vertical after the rotation, and then find the AABB, X'_{AABB}, in the new coordinate as described above. Finally we rotate the AABB back to the original coordinate as

$$\begin{aligned} X_{OBB} &= R^T X'_{AABB}, \\ X_{OBB}, X'_{AABB} &\in \mathbb{R}^{4\times 2}. \end{aligned} \tag{4.4}$$

Here X_{AABB}, X'_{AABB} are the stacked coordinates of four corners in the original coordinate and transformed one respectively. With the positions of four corners, it is easy to determine parameters in $< x, y, w, h, \theta >$.

Coordinate Transfer

Both types of bounding boxes are generated in the ego-vehicle coordinate. However, as an image-based object detector, YOLO requires that the annotations should be in pixel coordinate. Figure 4.4 (c) illustrates that the origin of the pixel coordinate is at the up left corner, the vertical axis y_p is larger at the bottom of the picture and the horizontal axis is larger at the right of the picture. This is different from the car coordinate. x_{cc}, y_{cc} is the position in the car coordinate, whereas w_{cc}, h_{cc} are the width and height in the car coordinate respectively. The transformation from the ego-vehicle coordinate to the pixel coordinate can be formulated as

$$\begin{cases} x_p &= -\frac{y_{cc}}{L} + \frac{1}{2} \\ y_p &= 1 - \frac{x_{cc}}{L} \\ w_p &= \frac{h_{cc}}{L} \\ h_p &= \frac{w_{cc}}{L} \end{cases}, \tag{4.5}$$

where $L = 100m$ is the size of the scene, i.e., the radar range, $x_p, y_p, w_p, h_p \in [0, 1]$ are relative values in the pixel coordinate, i.e., absolute pixel coordinates can be restored by multiplying with the length of grid maps. In a similar fashion, the output of YOLO is in the same format as the annotations, i.e., in the pixel coordinate. Therefore, a reverse transformation is applied to visualize the result.

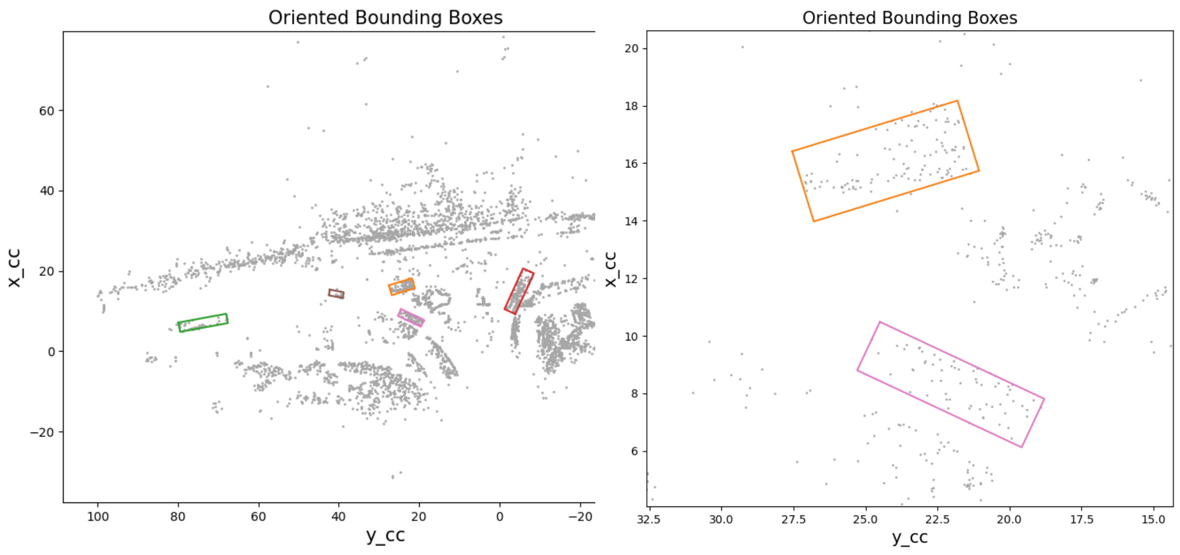


Figure 4.3: Oriented Bounding Boxes: The snippet from the first 28 frames in sequence 137.

4.1.2 Grid Mapping

In the beginning, we followed the approach in [36], where the number of grid cells are empirically set as 608. As YOLO requires, the input length and width should be a multiple of 32. Considering that the range of a snippet is $100m \times 100m$ and the length of a normal passenger vehicle is around 4 meters, 608×608 grids means that each grid cell corresponds to $0.16m$ and a passenger vehicle takes 25 cells. As in [36], three grid maps are extracted from each snippet, namely, maximum amplitude map and maximum Doppler map, minimum Doppler map. See Figure 4.4 for an example. For the first map, the value of each grid cell is the maximum amplitude value (RCS) among all the points that falls in the grid cell. For the second, the value of each grid cell is the maximum Doppler velocity among all the points that falls in the grid cell. The third one takes the the minimum Doppler velocity instead. This is because the Doppler radial velocity is singed, the positive value means a road user that is moving away from the radar and the negative value means a road user that is moving towards the radar. Therefore, two Doppler maps can reflect the velocity distribution in each cell.

4.1.3 Doppler Velocity Skew Function

The left of Figure 4.5 illustrates the distribution of absolute value of Doppler radial velocity. It is heavy-sided towards zero because most of the background points are static. In [36], a monotone forth polynomial is applied to skew the Doppler distribution. See the right of Figure 4.5 for the skew function. It is defined by interpolation of supporting points $(0, 0), (10, 0.7), (20, 0.9), (27.5, 0.95), (40, 1)$.

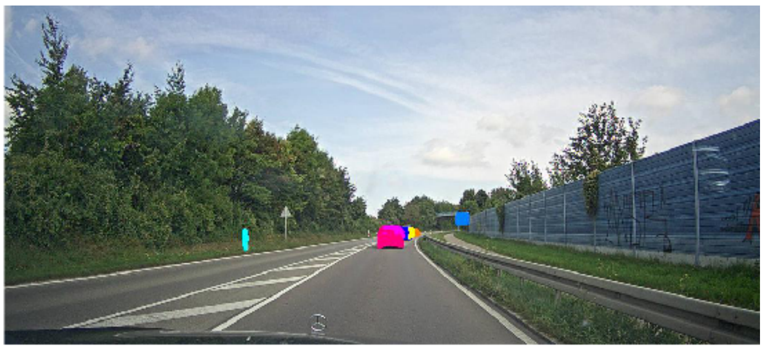
4.1.4 Blurry Filter

The information about how many points falls in a grid cell cannot be preserved in grid maps. It is intuitive to create a forth map in which the value of each grid cell corresponds to the number of radar points that are inside the cell. However, in [36], a blurry filter is applied on the first three maps with the help of the forth quantity map. The blurry filter propagates the values of non-empty cells in the first three maps to their empty neighbours if the corresponding values in the quantity map exceeds an empirical threshold. Figure 4.6 illustrates the propagation scheme. This filter is nonlinear and has to implement as a set of rules:

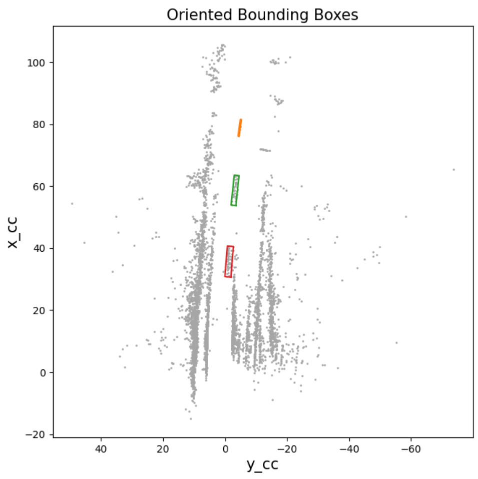
- Only propagate to empty cells;
- If propagation zones of two origins overlaps, fill in the empty cell with the value of closer origin.
- If distance is the same, fill in the empty cell with the value of origin which contains more points.

4.1.5 Data Augmentation

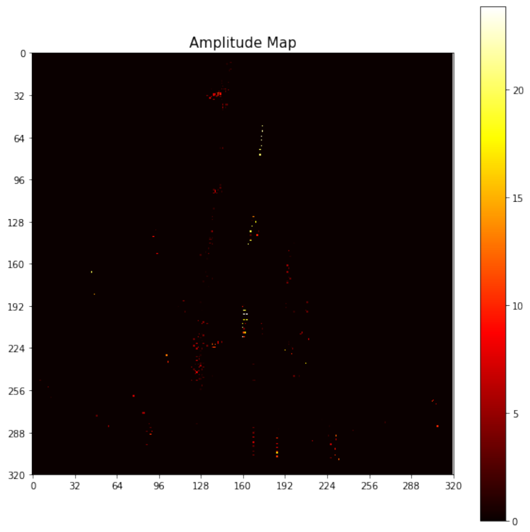
We apply the data augmentation (DA) directly on snippets before grid mapping and bounding boxes extraction. In [8], DA is randomly rotating grid maps by multiples of 30°. Since rotation may lead to loss of target objects after clipping. We don't



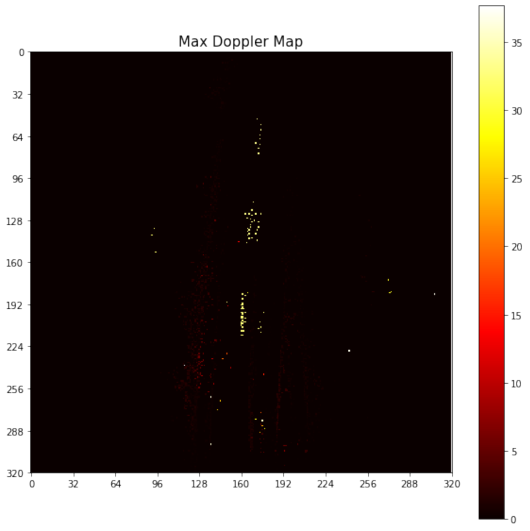
(a)



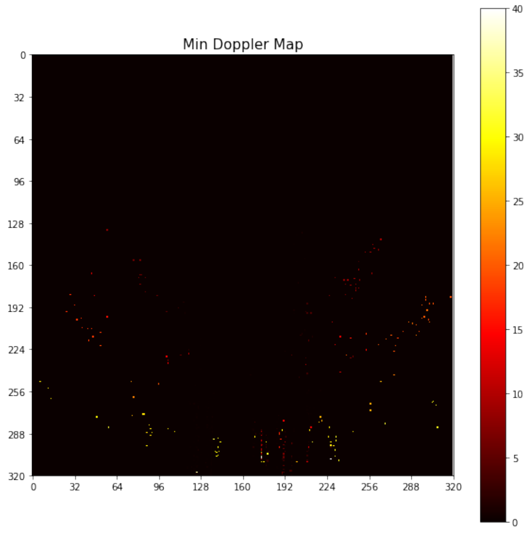
(b)



(c)



(d)



(e)

Figure 4.4: An example for grid mapping: a snippet from sequence 109 frame 816 to frame 845. (a)reference image;(b)point cloud with oriented bounding boxes. Three objects are visible;(c)Amplitude map. There are three clusters marked as light yellow corresponding to three objects; (d) Maximum Doppler map, the correspondence is more obvious; (e)Minimum Doppler map, the correspondence is less obvious. All three objects are moving away from the test vehicle, therefore max Doppler map shows the clearest correspondences.

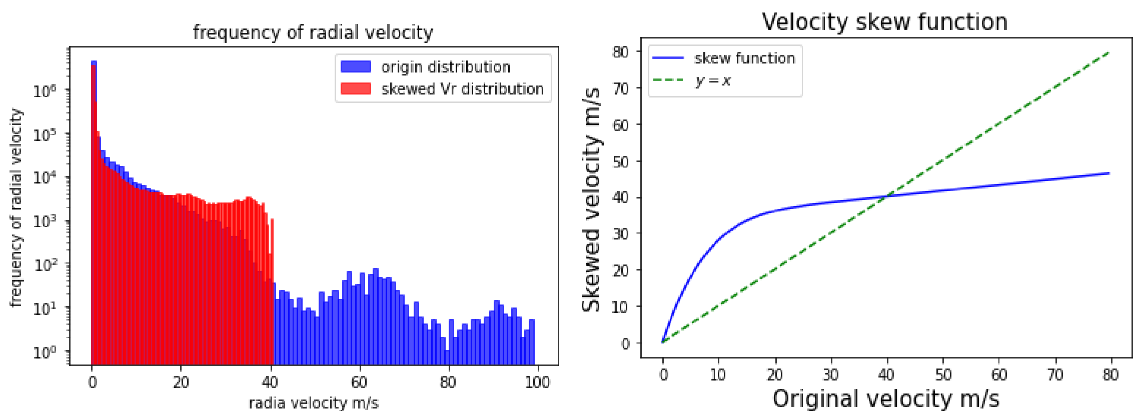


Figure 4.5: Left: Histogram of the radial velocity distribution; Right: Fourth order polynomial skew function.

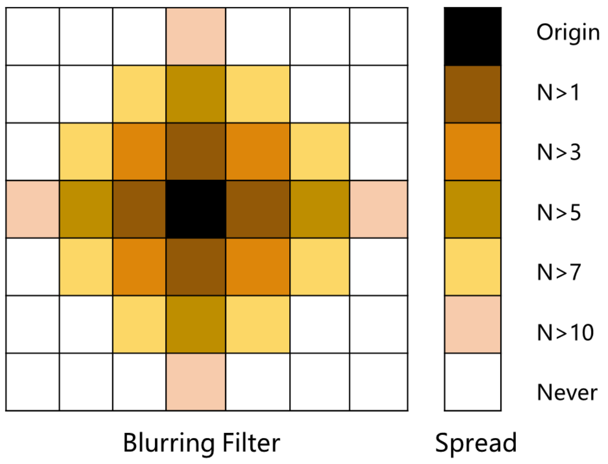


Figure 4.6: The propagation scheme for blurry filter from [8] IEEE, 2021, Fig. 6.

rotate the grid maps instead of snippets for the convenience of implementation. DA consists of random rotations, transitions and moreover, flipping, the first of which are similar to (3.2). A rotation matrix is constructed as

$$\theta_i = 30^\circ \times n_i$$
$$R_i = \begin{bmatrix} \cos\theta_i & -\sin\theta_i \\ \sin\theta_i & \cos\theta_i \end{bmatrix}. \tag{4.6}$$

Where i denotes the DA in the i^{th} iteration and $n_i \in \{0, 1, 2, \cdots, 11\}$ is a random integer. The transition moves the snippet along the horizontal axis by a random number in the region of $[-25, 25]$ m. Lastly, we randomly flip the snippet horizontally. Only horizontal transforms are applied is because we want to preserve the feature that the point cloud becomes sparser as the range increases.

4.2 Instance Segmentation Approach: Semantic Segmentation Network (PointNet++) and Radar Clustering (DBSCAN)

Now, we will dive into the second studied object detection model. According to [36], one of the most effective ways to execute perception in road environments using radar datasets is through clustering with adapted deep learning models. We selected two known effective frameworks: PointNet++ [33] as the semantic segmentation network (deep learning model) and the Density-based spatial clustering of applications with noise (DBSCAN) [10] as the clustering algorithm. The objective of the whole process is to find instance segments (objects) by performing two tasks. First, the semantic segmentation network predicts point-wise class labels from a point cloud P (snippet taken from RadarScenes), and then, the modified clustering algorithm groups these labeled points into clusters of the same type. Figure 4.7 shows how the data is processed at the different stages of the pipeline.

The clusters are formed following the guidelines:

- Each cluster unit will contain only one type of predicted class label for all the points within the such cluster. The same criteria are applied to all the clusters found in the snippet.
- The clusters are formed following a custom distance metric and a minimum neighbor points criteria specially designed for radars. See section 4.2.6.
- The points are filtered before training the network and the clustering stage (after having a trained model). Thus, the semantic network and the clustering algorithm will increase the efficiency of the labels and clusters, respectively. The filters and how they work are described in detail in the subsection 4.2.5.

4.2.1 Radar Point Cloud Extraction

As described in subsection 3.4, the input unit is the *snippet* taken from the sequences of radar data given in the RadarScenes data set, which are collected and formatted

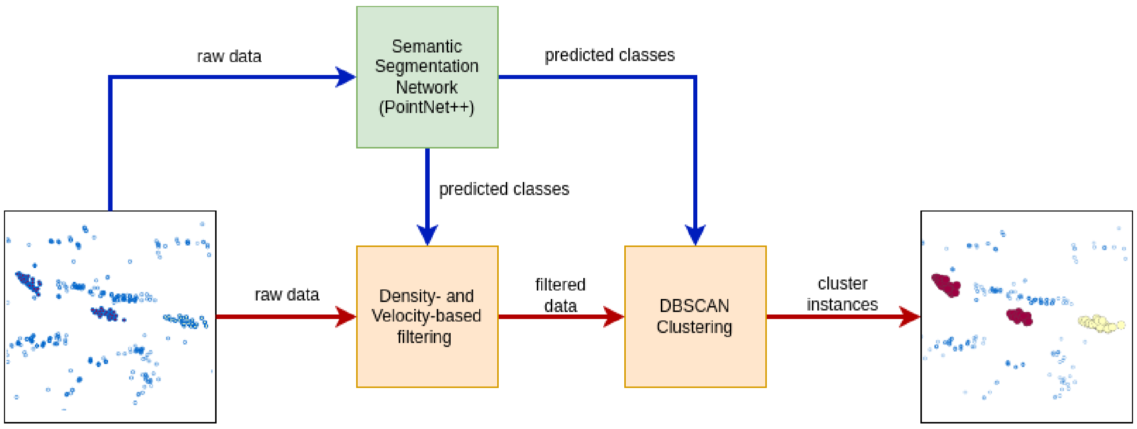


Figure 4.7: Pipeline for the object detection system using the algorithms of DBSCAN and PointNet++.

as stated in section 3.4. Note that the number of points in each snippet is not fixed, and it is only populated by the number of reflections present at that time. The time length was tested successfully in [37], and that is the reason why we adopted this as a baseline model (Analysis of these results in section 5.2).

The resulting snippet of such formatting will be the unit input to the PointNet++ pipeline. In the current model, each of these snippets will represent a point cloud P element in $\mathbb{R}^d$. Each point cloud P is defined as a set of $N \in \mathbb{N}$ individual points $p_i \in \mathbb{R}^d$ with no relevance in the order. Thus, p_i constitutes a multi-dimensional point in $\mathbb{R}^d$ where its dimension d is given by a) the number of features retrieved from the dataset and b) the algorithm the point is passing through. i.e., the information extracted from the dataset (stated in 3.2.2, data structure) is selected depending on the features the segmentation network (PointNet++) or the clustering algorithm (radar DBSCAN) require, respectively. Therefore, d will contain the number of corresponding radar features as follows:

- In the semantic segmentation network, each feature point in the snippet will contain two spatial coordinates x and y, and the ego-motion compensated Doppler velocity $\hat{v}_r$ and the radar cross section (RCS) σ. Hence, $d = 4$. The class labels *label_id* per point vector are also included in the training preserving the same indices as the feature points array.

- In the clustering algorithm, each reflection will contain two spatial coordinates x and y, and the ego-motion compensated Doppler velocity $\hat{v}_r$ and the range r. Hence, $d = 4$. These will not contain the class labels *label_id* as in training but will contain a prediction label per each point computed on-the-fly at the inference stage and before being passed to the clustering algorithm. Therefore, the predicted label vector will have the same indices as the feature points array.

Thus, $d = 4$ for both processes. The usage of these features in each algorithm is explained in the following sections.

Point Cloud in the Semantic Segmentation Network

Unlike the input of the first studied object detection model (YOLO), which uses fixed pixels in a 2D bird-eye image regardless of the number of points, the input size for this model is initially not fixed, as discussed earlier. Each raw point cloud unit has a number of points that are not fixed up to a number, i.e., the sizes of the point cloud set to feed the network are different in point lengths because the point cloud is populated depending on the objects that were present at the moment of extraction in the mentioned time length from section 4.2.1. The consequence is that a typical convolutional neural network will have processing issues accepting each point cloud (or rejecting the point cloud directly) because the mapping is performed by a convolutional kernel that expects a constant even grid every time.

To overcome this problem, we have selected PointNet++ as our architecture because it does not necessarily deal with constrained point cloud formats. Thus, its usage fits adequately for this dataset. As it is seen in section 2.5.1 from the article [32], the unstructured format of PointNet++ does not affect the algorithm as long as in each snippet: the points N are invariant to their $N!$ permutations, the points are not isolated from one another, and they have the tolerance to geometric transformations.

The only constraint for feeding this semantic framework is that it has to contain a fixed number of points N per snippet. Hence, for any training/validation/test snippet samples,

$$\begin{aligned} \text{Samples} = (P_1, \ldots, P_M) \quad &, \quad M \rightarrow \text{number of snippets} \\ P = (p_1, \ldots, p_N) \quad &, \quad N \rightarrow \text{number of points} \end{aligned} \tag{4.7}$$

As we discussed in equation 4.7 the number of points, N is not fixed in the beginning due to the random number of reflections accumulated over time for populating each point cloud (snippet). As a result, random upsampling and clipping procedures are implemented for each point cloud. These operations guarantee that the number of points will be fixed up to a predefined and constant number for all the point clouds intended to be fed (more of these procedures in subsection 4.2.2).

Point Cloud in the Clustering Algorithm

As is seen in Figure 4.1, each snippet in the test dataset is passed through the semantic segmentation network (PointNet++) and the clustering algorithm (DBSCAN) section of the pipeline in the exact mentioned Figure.

After having a trained model, the inference stage of the semantic network will only accept a constant number of points in each dataset snippet, as in the point clouds in training mentioned in subsection 4.2.1. On the other hand, the clustering algorithm is unaffected by the number of points in any snippet. Since the replicated points from the random upsample or clipping procedures can bias the clustering after the semantic inference, then they need to be returned to their original number of points, preserving the integrity of the snippet. This is easily done by discarding the points where the information is identical in all the features.

After recovering the original number of points for clustering, the point cloud needs to be passed through a filtering process in order to refine the quality of points and get rid of the noise as much as possible. The process of filtering is described in 4.2.5.

4.2.2 Upsampling and Clipping of Dataset

As it was stated in section 2.5.1, the semantic network PointNet++ accepts unordered point clouds with any number of features with any number of points without any problem as long as this number of points per snippet is constant during training or inference, i.e., all the snippets in the training/testing data set must keep a constant size before to be fed in batches to the network. This is due to each batch of snippets passing through a series of feature transformations, layers of MLPs (Multi-Layer Perceptrons), and max pooling operations (see more of these stages at [32], and [33]). The parameters of these operations are configured considering a constant number of points prior to the training/inference. Otherwise, the parameters, and therefore, the network architecture, would have to change their parameters constantly just for the non-constant number of points in each snippet, which is far from practical.

The obvious solution for keeping a constant network architecture is to modify the number of points in all the snippets by re-sampling them to a constant number. Resizing the snippet by adjusting the accumulation time (defined to be at $T = 500$ms) is an option. However, Scheiner et al. [37] recommend using fixed time slices and then either performing a random upsample until the selected number of points in the snippet is achieved or randomly clipping the points labeled as noise (static class) down to the selected number.

Neither the clipping process nor the random upsampling process represents a change in the outcome of the semantic segmentation network. The clipping is performed on the static class points by removing a number of them and keeping the non-static class ones, as shown in Figure 4.8 (a). The random upsampling method randomly replicates points following the nearest neighbor interpolation criterion, which in essence means that the algorithm calculates the number of missing points, then randomly selects existing points up to that number, and replicates them identically one or more times until that constant number of points is reached, Figure 4.8 (c). These extra points do not affect the training at all since the max pooling layers (feature mapping procedure, see more in [32]) map these identical points as if they were only one, and therefore, they do not influence the training/inference.

Thus, the selected number of points is optimized to 4096 according to [36] in order to apply the random upsample technique more times than the clipping technique, so the integrity of the snippet prevails. Moreover, the clipping and upsampling process follow corresponding guidelines,

- When *clipping*, reflections from the *static (noise)* class are removed. No *non-static* class reflections are removed, since the dataset is highly imbalanced towards static points (97 million static to 3 million non-static [37]). Such non-static points removal would severely affect the performance of the training/inference if applied.

- When *upsampling*, random reflections are re-sampled the required amount of times.

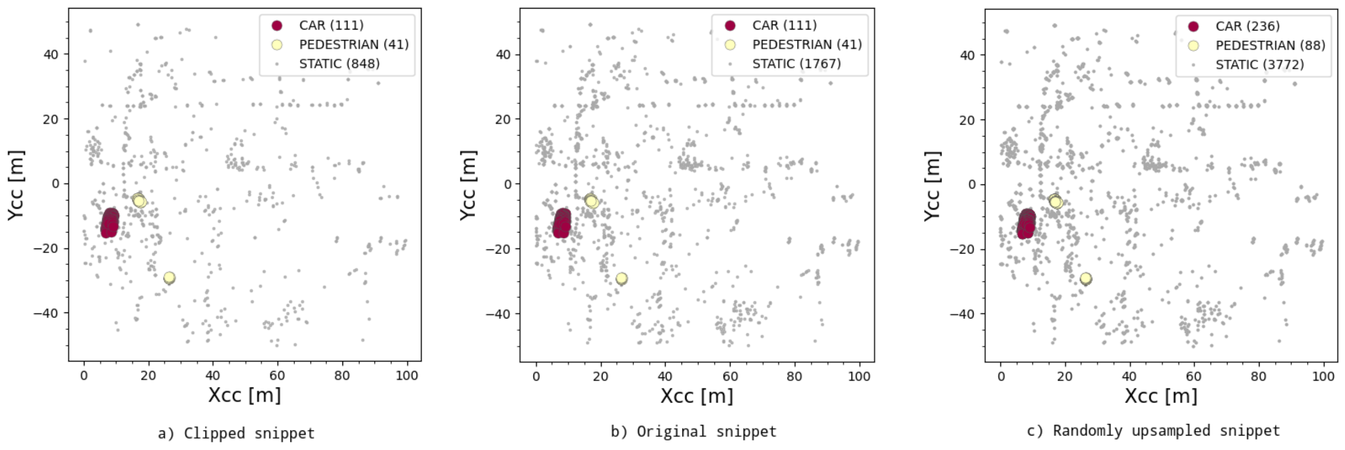


Figure 4.8: A snippet with the original number of points (b), after clipping (a), and after random upsampling (c) procedures. Note that the clipping is done over static class points in (a) preserving the non-static points. Also, only for this example one can see that the length of the snippet is not 4096 points

As a result of this process, each point cloud P (snippet) will have the shape,

$$P = [N\_points, n\_features] \quad , \quad N\_points = 4096, \tag{4.8}$$

where $n\_features$ represents any selected features from the data set. In this case, recall 4.2.1, where we consider: x, y, v_r, RCS. Thus, $n\_features = 4$.

4.2.3 Point Cloud Preprocessing for Training the Semantic Network

We explored and implemented two data preprocess stages to improve the training/inference of our pipeline.

It is necessary to acknowledge that the number of non-static labeled points against the static labeled ones is outstanding since we got a relation of 97 million static points against 3 million non-statics points (31:1 overall ratio) after counting all the points in the dataset. This causes a model acquires high accuracy in predicting the majority class (static) and poor performance in the minority classes. In our case, the minority classes possess more valuable knowledge than the static ones. For this reason, a stage of weights biases at the cross-entropy loss function stage while training was implemented to mitigate this undesired effect successfully. [41].

Also, after an exhaustive visual inspection of the snippets, we realized that there are sequences of snippets where points did not add much information from one snippet to the next one. These cases occurred when the non-static objects were not moving much (for instance, cars at a T cross intersection waiting for the light of the semaphore to change to green, or pedestrians standing for long periods). Therefore, a data augmentation process (jittering) was performed to enhance the information at each snippet. Hence, the shape and density of the non-static points change; therefore, the snippet would be richer in information. Thus, the network can learn even more from this synthetic data.

Weight Bias for the Loss Function

First, due to the mentioned imbalance in the dataset, initial weights were counted and applied in such a way that while training, each static class point in the snippet was penalized, and each non-static class point was rewarded at each update in the loss function from epoch to epoch, i.e., from counting the ground truth labels, we computed weights per class w. Then, these weights are inserted in the updates of the loss L_i resulting from the *Categorical Cross-Entropy Loss* cost function while training, as is shown in Equation 4.9,

$$L_i = -w \times [y\_true_i \times \log(y\_pred_i) + (1 - y\_true_i) \times \log(1 - y\_pred_i)] \tag{4.9}$$

where i refers to the class the cross-entropy is being calculated for ($i = 0, 1, 2, 3, 4, 5$ for *cars, pedestrians, group of pedestrians, two wheeler, large vehicles*, and *static* classes, respectively), *y_true* and *y_pred* are the vectors containing the ground truth values (annotations in dataset) for each point, and the predicted values in each point while training respectively, and w are the weights that will bias the cross entropy computations over the distributions (total number points in one-hot vector per class *y_true* or *y_pred* in our case) according to our dataset statistic count.

Moreover, since each split of the training/validation/test dataset contains around the same ratio of imbalanced data as in the entire data set, we can argue about a constant behavior of the imbalanced data in any random split. However, before training, we are calculating the actual initial weights per dataset. To depict the imbalance in the RadarScenes dataset, we show this imbalance in percentages in Table 4.1. Here, we show to what extent the initial weights will influence the optimizer after the cross entropy loss function in almost any training process.

Table 4.1: Estimated number of points in percentages across the entire dataset from RadarScenes

car	*ped*	*ped.group*	*bike*	*truck*	*static*
1.23%	0.31%	0.74%	0.11%	0.60%	97.01%

Even though we know these percentages, we calculated the actual initial weights per training split, i.e., customized weights $w(i)$ per class i for the split we are using. To achieve this, we introduced inverse weights (on ground truth label per point) for each count n as

$$\begin{aligned} n_{class} &= [n_{car} \quad n_{ped} \quad n_{grp} \quad n_{cyc} \quad n_{trk} \quad n_{noise}], \\ w(i) &= \frac{\frac{1}{n_{class}(i)}}{\sum_{j=1}^{N} \frac{1}{n_{class}(j)}}. \end{aligned} \tag{4.10}$$

For instance, if we have $n_{class} = [20 \quad 10 \quad 1000]$ (cars, pedestrians and noise) labels then, the corresponding weights are: $w_1 = 0.3311$, $w_2 = 0.6623$ and $w_3 = 0.0066$.

The actual weights computed for each class (car, pedestrians, group of pedestrians, two wheeler, large vehicles, static) in the training data set are shown in relation 4.11,

$$w = [0.07841 \quad 0.19531 \quad 0.10594 \quad 0.46180 \quad 0.15459 \quad 0.00317]. \tag{4.11}$$

Data Augmentation

Although the initial weight regularization mitigates most of the effects of having an imbalanced data set, the data was still sparse and could be more representative. We needed data augmentation techniques focused on enriching the non-static points in the training data set to decrease the generalization gap formed by the training data curve and the validation data curve, preventing overfitting. Therefore, we applied two techniques from Schumann et al., [37], with different adaptations, as we will show.

First, we applied random noise to each feature dimension, point by point in non-static objects, so that each feature (x, y, v_r, RCS) was randomly altered and copied within a range of a standard deviation of $\sigma = [0, 0.1]$ (up to 0.3 suggested in [37]). We applied this method to each snippet because they are sampled from a time sequence, and the snippets right behind and right ahead of that snippet are very similar to each other, adding little new relevant information to the training and making it susceptible to overfitting. Thus, after applying random noise, the result is an augmented jittered version on the non-static points, which gives newly enhanced information in shape and density, snippet by snippet of such points, and prevents the network of training over similar features. Also, they are left out in the same probability as σ.

Next, the same random noise was applied to static noise but without affecting the radial velocity feature, but the other three left (x, y, RCS). Unlike the non-static points, the random noise on static points is just for altering the points and not making them grow in number. One snippet before and after the jitter data augmentation process is shown in Figure 4.9.

4.2.4 PointNet++ Semantic Segmentation Architecture

This version for the semantic segmentation network has the architecture shown in Figure 4.10. The selected architecture is an adaptation of the best candidate shown in [36] and [37]. The network parameters contain: a number of MSG modules, a number of samples N_{sample} per MSG module, the number of neighborhoods in each MSG module, their radius r, the number of neighbor points N_{neigh} per point, and the shape and number for the convolutional layers; these were selected according to the specifications in [33] and [37], and tuned using random validation sets. This had to be done this way since a complete sampling of the parameter space is not feasible.

The adaptations are located in the radii and number of neighbors in MSG modules MSG1 and MSG2 with $r_1 = 2[\text{m}]$, $n_{neigh} = 10$, and $r_1 = 2.5[\text{m}]$, $n_{neigh} = 16$, respectively and shown in Figure 4.10. These values and those shown later were selected after testing and learning how effective they were by looking at the accuracy,

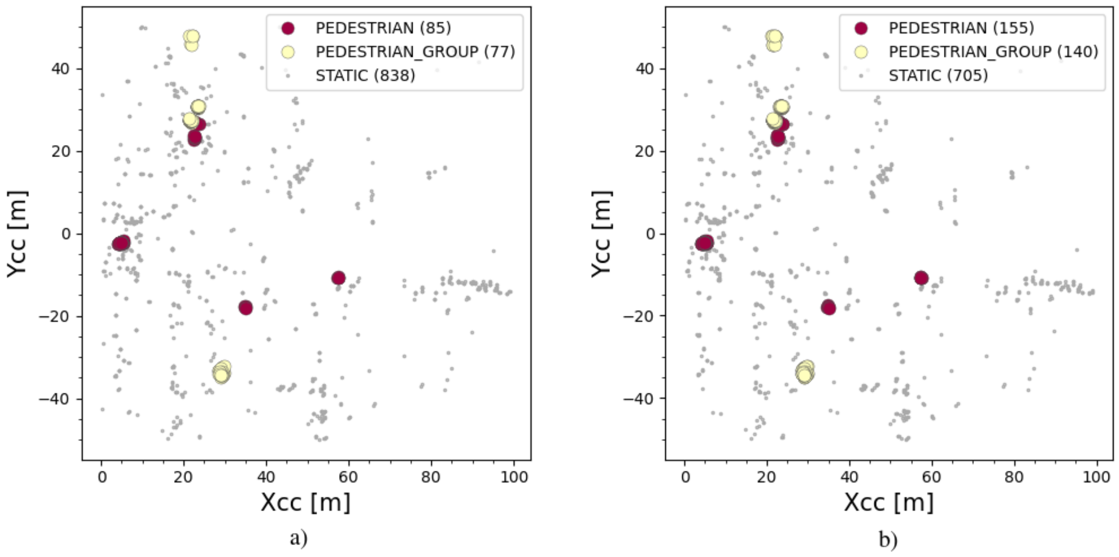


Figure 4.9: A snippet with the original number of points (a) and after data augmentation (b). Note the increasing number of points in dynamic classes and reduced by the same number in the static class. Although the jittering is not as visible as expected in the plots, the favorable effects of the training are remarkable.

loss, and confusion matrix metrics. There are other configurations for Pointnet++ with radar points like the one presented by Liu et al. in [28]. Our modified version adapted from [36] and [37] performed as efficiently as their baseline model.

Figure 4.10 shows the following architecture: three Multi-Scale Grouping (MSG) modules from PointNet++, three Feature Propagation (FP) modules also from PointNet++, three 1-dimension convolutional layers with two dropout regularization methods in between of them, and a Softmax activation function for retrieving the points' classes.

As shown in 4.7, the purpose of the network is to identify each point with a class label semantically; therefore, as the MSG modules are connected to the FP modules as described in previous section 2.5.2 the network can propagate the classification to each point.

Also, from Figure 4.10, the Multi-Layer Perceptrons (MLPs) kernel sizes in the MSG modules are:

- MSG 1: $[[32, 32, 64], [64, 64, 128]]$
- MSG 2: $[[32, 32, 64], [64, 64, 128]]$
- MSG 3: $[[64, 64, 128], [64, 64, 128]]$

4.2.5 Point Cloud Filtering for Clustering

Prior to applying the radar clustering stage, a filtering process is applied to the data in order to prune the noise and enhance the quality of the clusters. The selected filtering process is chosen, so the DBSCAN algorithm can relax the hyper-parameters

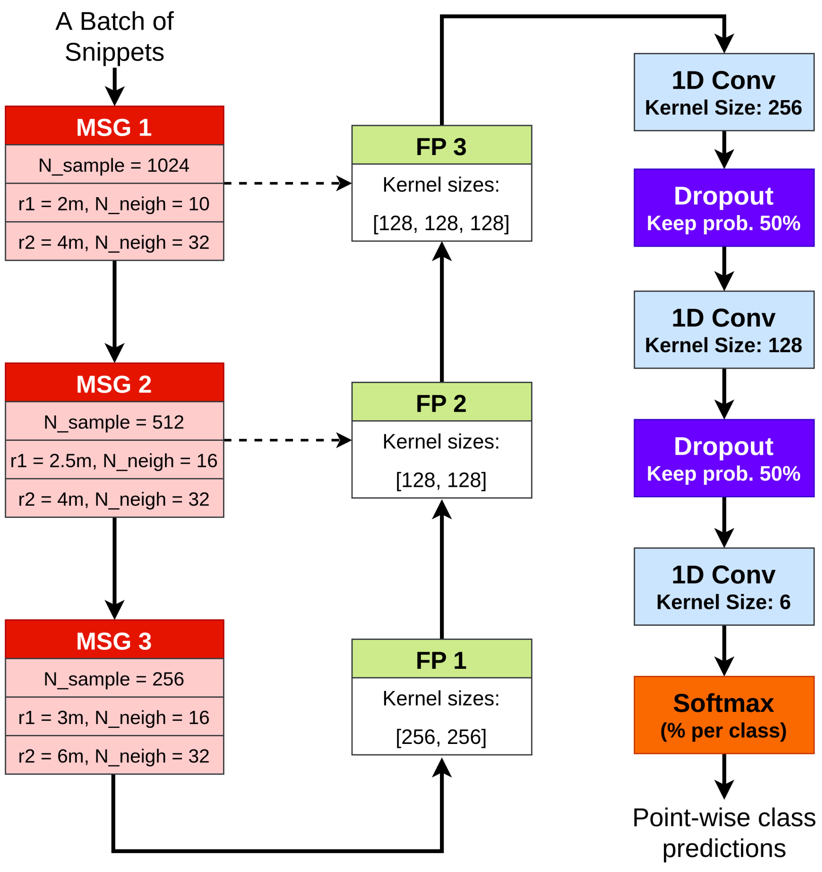


Figure 4.10: Structure of the semantic segmentation network. The segmented line indicates skip connections from which extracted features from MSG modules are passed to the FP modules. Improved from [36] and [37].

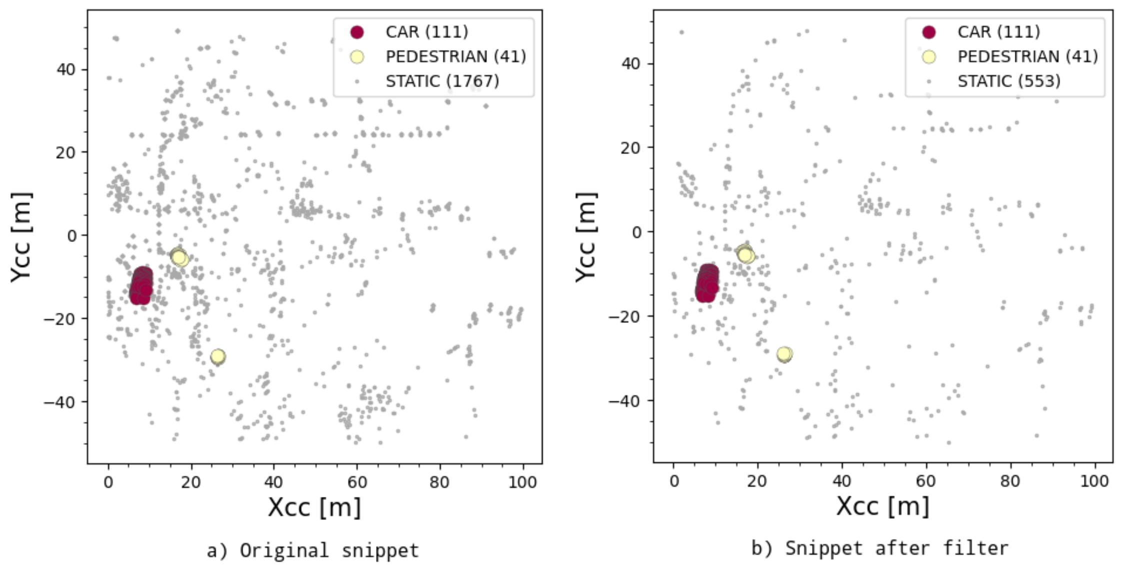


Figure 4.11: Original snippet (a), pruned snippet after filtering (b). Notice the number of static points dropped. In this example, $\eta_{v_r} = 0.0022$.

(minimum number of neighbors and minimum radius) and allow them to have a broader range of values.

The points that are below a certain radial velocity v_r (Doppler) threshold and were predicted with a *static* class label are deleted. Equation 4.12 defines this filter over the point cloud P (snippet).

$$P_{\text{filtered}} = \begin{cases} \text{where} & |v_{r_{P_{\text{original}}}}| < \eta_{v_r} \\ \text{where} & \text{labels}_{P_{\text{original}}} = \text{static class} \end{cases}, \tag{4.12}$$

where η_{v_r} is the velocity threshold and the rest of the equation is the segregation of the points that belong to the static class too. The result is a pruned snippet with less clutter, as is shown in Figure 4.11.

4.2.6 Class-Sensitive Filtering and Clustering

The selected algorithm is a modified version of the euclidean DBSCAN clustering algorithm seen in [10]. Recall that the clusters are formed in the euclidean DBSCAN by tuning the parameters N_{min} and ϵ.

We decided to use a clustering method inspired by the DBSCAN clustering technique shown by Liu et al. in [28] and by Scheiner N, et al. in [36]. Liu and his team decided to perform the clustering at each non-static class with no modifications in the distance metric radius or the minimum points. This means that the algorithm will perform clustering five times (one per non-static class: *cars, pedestrians, group of pedestrians, two-wheelers, and large vehicle*) per snippet. It will add the clusters found at each class clustering iteration to the snippet clusters with a default DBSCAN algorithm. It is worth mentioning that before the clustering starts, all the points will have the noise (static) label and will be changed as the DBSCAN iterations are performed.

Then, we took a different approach from Liu and his team by adopting a radar distance metric radius inspired by Nicholas and his research team, which consists of a radar-based distance metric. In order to take advantage of the radar data, we decided to use the $e - region$, which is a multi-dimensional neighborhood distance metric around each point that will be used to define a cluster in radar data. Equation 4.13 defines the $e - region$ as follows,

$$\begin{cases} \sqrt{\Delta x^2 + \Delta y^2 + \epsilon_{v_r}^{-2} \times \Delta v_r^2} < \epsilon_{xyv_r} \\ \Delta t < \epsilon_t \end{cases}, \tag{4.13}$$

where, Δx and Δy are the distance differences between two analyzed points, Δv_r is the radial velocity difference between the same mentioned points, the number of radar points collected during a time is Δt, and finally ϵ_{v_r}, ϵ_{xyv_r} and ϵ_t are the thresholds of radial velocity, e-region and time, respectively. The time threshold represents the accumulation time of 500 ms described before in the Dataset chapter. Likewise ϵ in the standard DBSCAN, the threshold ϵ_{xyv_r} will dictate which points are treated as core points, density reachable points, or noise points. The three thresholds applied at each iteration of non-static class allows the clustering algorithm to perform finer tuning of its parameters and form a more accurate clusters. To implement the e-region, we decided to modify the scikit DBSCAN platform implemented by Pedregosa et al. [31].

Both approaches in one are described in the Algorithm 1 and the corresponding outcome (using ground truth) of this process is shown in Figure 4.12.

Algorithm 1 Radar-based DBSCAN clustering

```
Require: $X$                                           ▷ All the featured points in a snippet $X$
Require: $\epsilon_{xyv_r}$, $\epsilon_{v_r}$, min_pts_per_class_list                  ▷ DBSCAN parameters
Output: clusters in $X$
  for class in classes do
      min_pts = min_pts_per_class_list(class)
      for i in X do
          e-region_in_points_list ← e-region($x_i, x_{i+1}$)
      end for
      clusters ← DBSCAN(e-region_in_points_list ≤ $\epsilon_{xyv_r}$, min_pts)
  end for
```

The parameters are optimized by clustering over the ground truth labels obtained from the RadarScenes original annotations before using any trained model from PointNet++. These parameters are:

- $\epsilon_{xyv_r} = 4.0$
- $\epsilon_{v_r} = 2.02$

And for the number of neighbors per class, the values are,

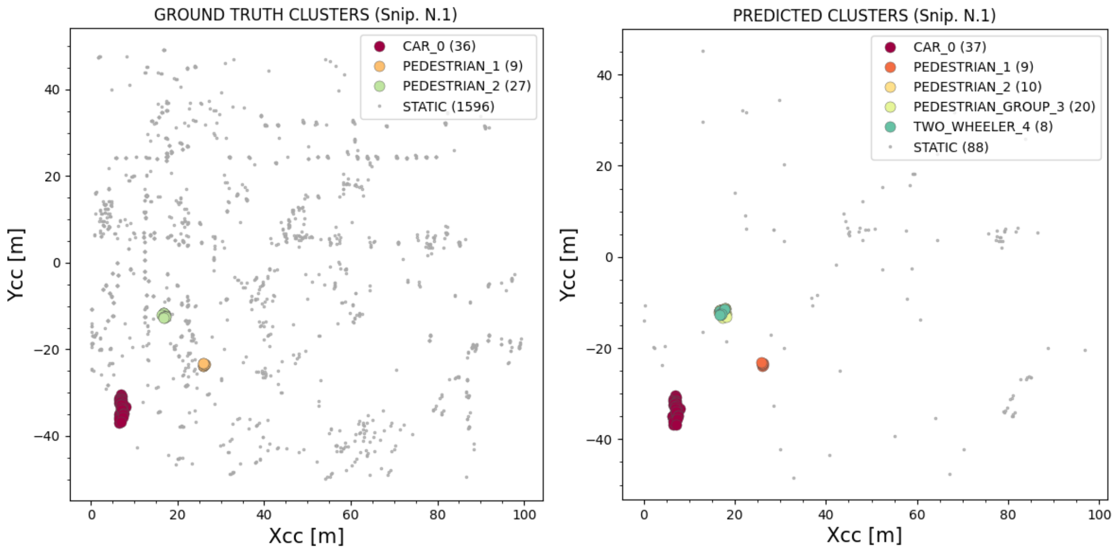


Figure 4.12: Original snippet with ground truth clusters (LEFT) and the same snippet with predicted labels and clusters (RIGHT). In this case, one car and two pedestrians clusters are shown along with the noise (static) points.

- $Nmin_{car} = 10$
- $Nmin_{ped} = 7$
- $Nmin_{grp} = 8$
- $Nmin_{bike} = 8$
- $Nmin_{truck} = 14$

5

Results and Discussion

In this chapter, we will first introduce the training and evaluation of two deep neural networks frameworks (YOLO v3 and PointNet++ with DBSCAN Clustering). Both frameworks are optimized with different loss functions and evaluated with their original metrics, e.g., accuracy for PointNet++ and pixel-based mAP for YOLO v3. Afterwards, a trained PointNet++ model needs to be inserted into the pipeline described in Figure 4.7 so that we can compare two approaches with point-wise metrics described in 2.7.2.

5.1 YOLO v3 Training and Evaluation

Our learning rate scheme is as follows:

- first 1000 iterations(the burn-in period): the learning rate gradually increases from 1×10^{-7} to 1×10^{-3}
- continue training with 1×10^{-3} for 4×10^{5} iterations
- 1×10^{-4} for 2×10^{5} iterations
- 1×10^{-5} for 1×10^{5} iterations

Batch size is 32 and the optimizer is Adam.

5.1.1 Ablation Study

Models without data augmentation were trained at the first step. We study the effect of blurry filters and velocity skew function separately. In 4.1.5, rotation and flipping are introduced. Different combinations of data augmentation strategies are tested. Random flipping consists of flipping horizontally, flipping vertically and the mixture. Random rotation consists of rotation by multiples of 30° and random rotation. Taken no flipping and no rotation into count, there are 12 combinations of data augmentation. We chose the combination with the top performance on the validation set and tested it with IOU = 0.3 because the elongated shape of bounding boxes will easily be considered as false negative because of low IOU.

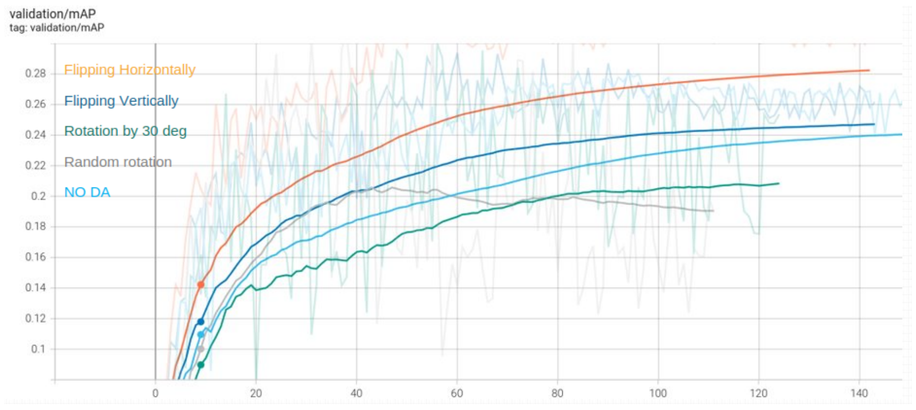


Figure 5.1: YOLO approach's performance on the validation set with various data augmentation operation

Table 5.1 presents the scores for DA strategies on the validation set. (Note: the scores are low because 1. the relatively unbalanced class distributions in the validation set; 2. the score is pixel-based mAP instead of point-wise mAP). In general flipping vertically can add 8-10 percent to mAP and the performance on the validation set is more stable. However, rotation will lead to an unstable score on the validation set. The combination of rotation and flipping did not further increase the scores, instead it destabilized the scores. Thus, we conclude that flipping horizontally is the optimal data augmentation operation.

Frames accumulation time, as a important hyperparameter, plays a main role in controlling the density of points in a snippet. Here we trained models with a series of accumulation time (from 100 ms to 1 s) and evaluate them with snippets that have the same length.Three models with the same hyper-parameters are trained on the the training set consisting of snippets of each length. Afterwards, the saved checkpoint at 70 epochs of each model is applied on the test set that has the same length. Fig 5.2 presents the result. Note that for classes Car, Two Wheeler and Pedestrian Group there is a clear improvement as accumulation time exceeds 600 ms. However, for class Truck and Pedestrian, the improvements are not significant. Especially AP for Pedestrian class remains at a low level, which turns out to be the bottleneck of the model.

5.2 Semantic Segmentation Network (PointNet++) Training and Evaluation

Our learning rate scheme is driven by,

$$lr = lr_0 \times \left(\frac{lr_{decay}}{100}\right)^{\left(\frac{\text{epoch}}{\text{step}}\right)}, \tag{5.1}$$

where lr is the computed learning rate, lr_0 is the initial learning rate at the beginning of the training. lr_{decay} is the learning rate decay which determines how much the initial learning rate will decay over the training in percentage, step is the step size which indicates for how many epochs the new computed learning rate will last until the next step, and epoch corresponds to the current iteration of the training.

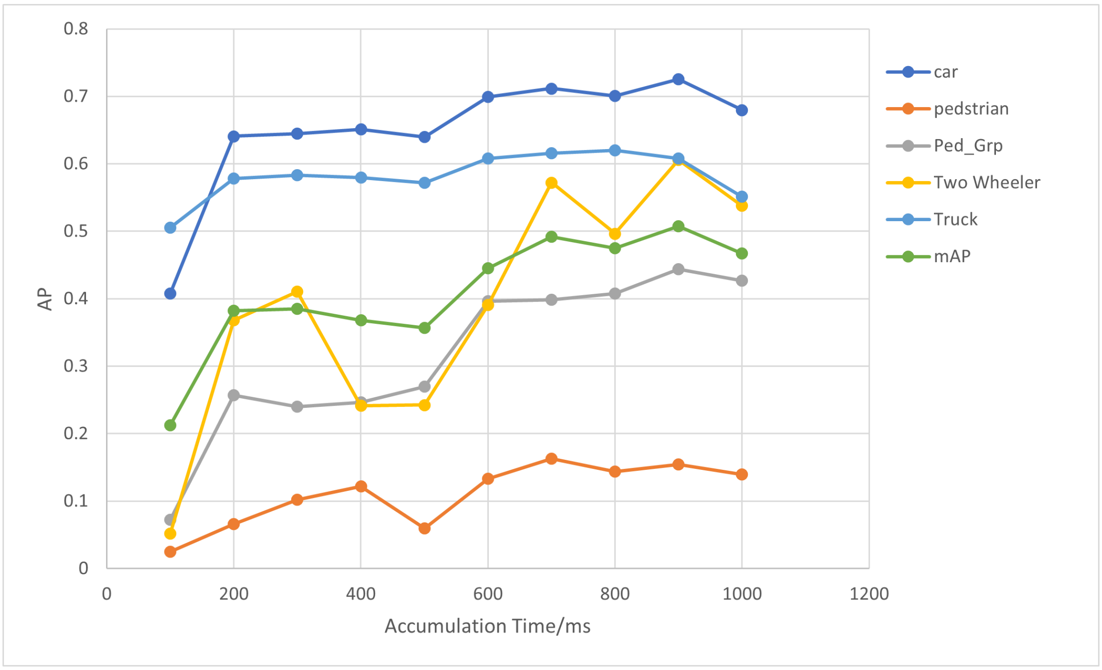


Figure 5.2: mAP and AP of each class on the test set changes over the accumulation time.

The training is conducted for 60 up to 70 epochs in around all trianings (baseline and oriented) . This suggestion is supported by [33], [36], and based on our own experience, i.e., the network starts to overfit in a baseline training after these number of epochs. The same behavior is shown for the epochs when normalization, regularization and data augmentation methods were applied.

The batch size at which the trainings performed was 64 because the GPU memory was larger enough to handle this size. The optimizer selected for any training is Adam (introduced by Kingma D. and Ba J. in [23]) with default parameters ($\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 1 \times 10^{-8}$) is preferred over the Stochastic Gradient Descent (SGD).

5.2.1 Semantic Segmentation Network (PointNet++) Baseline Model

After several trainings for determining our most successful base model for the semantic segmentation network, we narrowed down to a series of specific values of parameters and hyper-parameters. The following results could not be surpassed due to two reasons: the inherent nature of the data set and the performance capabilities of the network itself. These parameters are,

- Initial learning rate: $lr = 0.007$ up to $lr = 0.009$. The learning rate could be below 0.003 but it would be too slow for 30 to 40 epochs only and in some cases got stuck. Above 0.009 the network became unstable, aggressive and tended to converge too quickly to a suboptimal solution without recovering from there, i.e., predicting wrong values in non-static classes.

- Step size for learning rate decays: range between 12 and 15 steps in 55 and 60

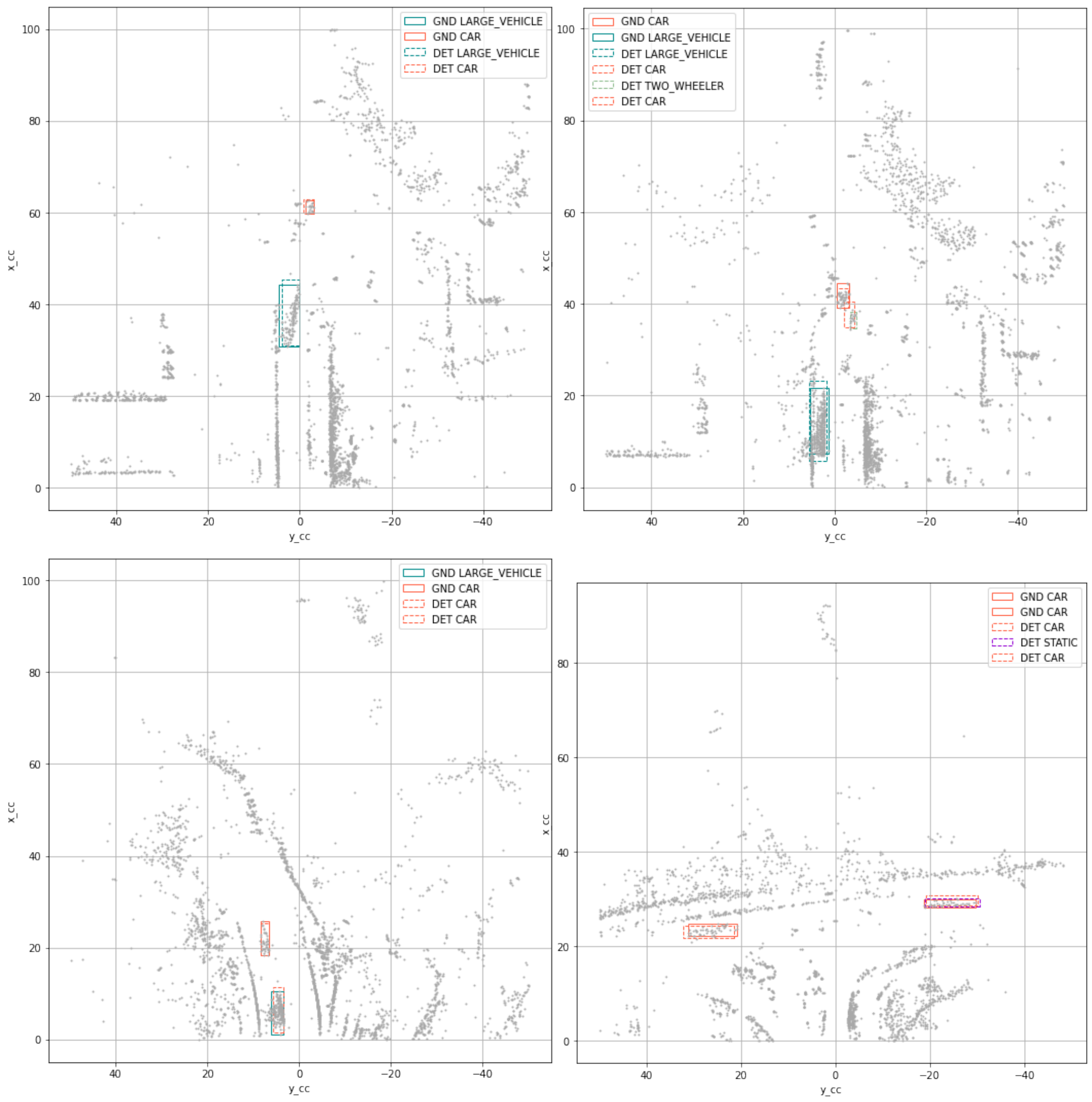


Figure 5.3: Visualization ground truth bounding boxes vs. detection bouding boxes

epochs, respectively. These values mean that the learning rate decays around 3 times during the whole training.

- Learning rate decay: 0.8, which means that the learning rate will decay 20% after the step size is reached. Then it will decay another 20%, and so on.
- Weight decay: 0.0, since it is recommended in [37] to avoid weight decay for the Adam optimizer in PointNet++.
- Data Augmentation (DA) techniques (random noise on features, and dynamic points dropout, subsection 4.2.3) are applied in order to consider the baseline model. The model is highly improved with these DA techniques against models without them, due to which these techniques are adapted to begin with. Also in [36], the authors suggested the use of to use these data augmentation techniques to enhance RadarScenes dataset in any training.

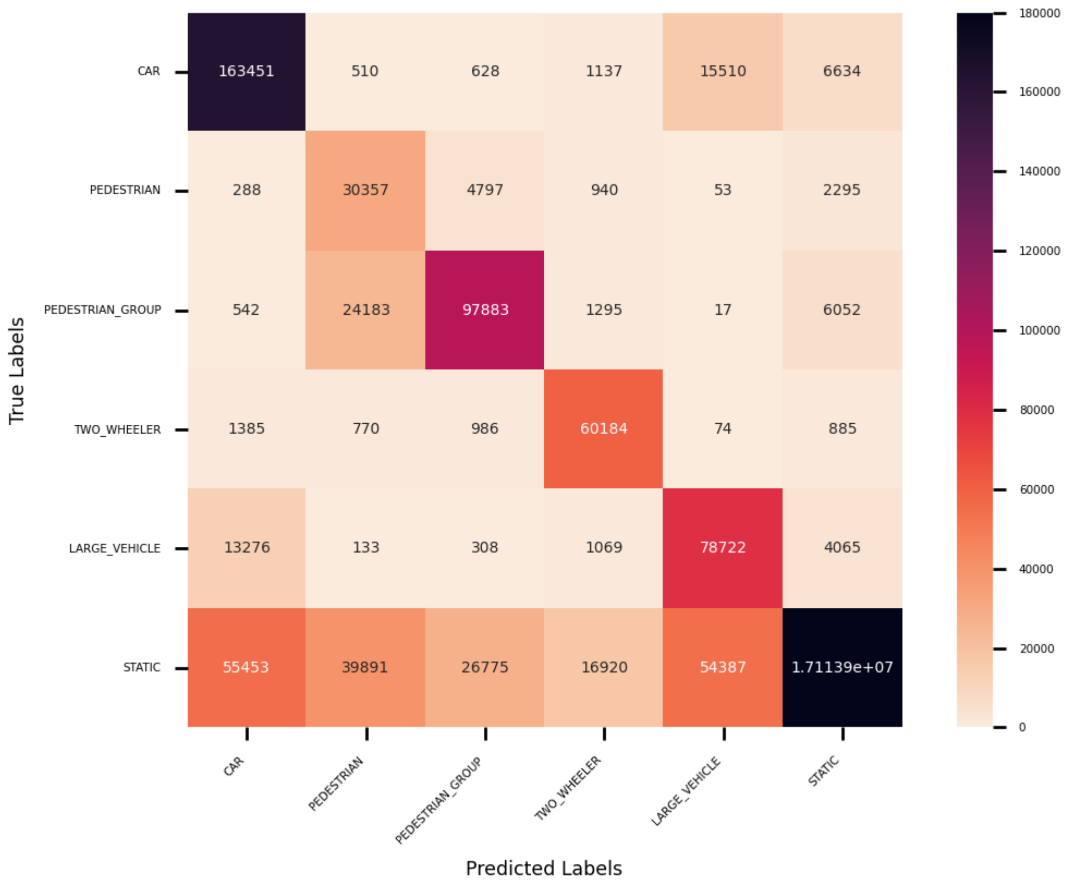
Our baseline model outputs the following confusion matrices in Figures 5.4 and 5.5. The reason for having both matrices, the relative and absolute, is that the relative does not highlight the many number of dynamic false positives (last row in Figure 5.4) predicted by the network. We can argue that these matrices are indicators of a good model since: a) the diagonals are greater in number compared to the rest of the matrix, indicating the good accuracy of the predictions performed by the model, and b) it is more beneficial to have more false positives (wrong detections of dynamic objects in original static ones, last row in the absolute confusion matrix) rather than false negatives (wrong detections of static classes in actual dynamic objects, last column in the absolute confusion matrix), i.e., larger dynamic objects than in the annotations; in this type of applications, larger objects lead to more sensitive and faster detections systems, however, this is also a drawback because the model could detect objects where there are none and therefore, unwanted results in an autonomous driver system, for instance.

Another metric that support the robustness of the model is the semantic Intersection over Union (IoU) on class label points (ground truth labels and predicted labels), and the training accuracy. The IoU and accuracy that corroborate our previous claim was performed on an unseen TEST DATASET. It is important to acknowledge that these metrics have to be taken at the moment where the best model was saved; to get such best model, the obtained IoU at each epoch iteration was compared with the previous best IoU and updated if the labels were predicted better than the previous versions using IoU. Thus, we ensured a model closer to the ground truth if the epoch resulted in a better maxima. The iteration at where the best model occurred was at iteration 59/60 but we took the one at 60 instead (explanation elamborated in the next paragraph), and the corresponding metrics were,

- Accuracy: 98.28%
- Loss: 59.02%
- Average IoU on each class: 59.66%

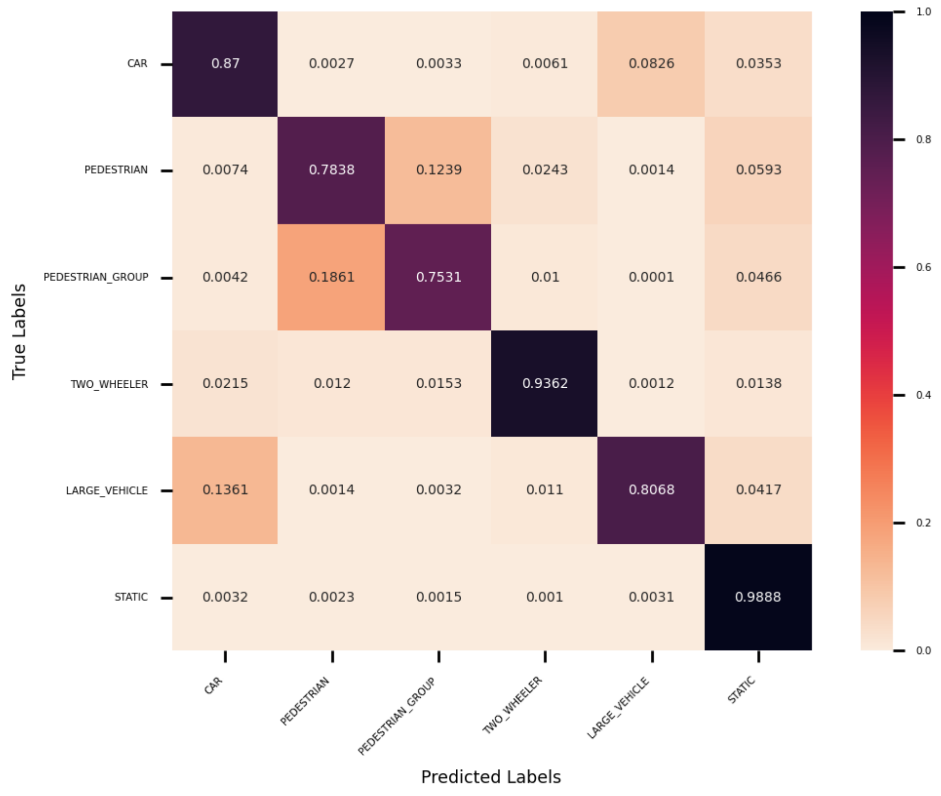
- CAR IoU: 58.4%
- PEDESTRIAN IoU: 27.1%
- GROUP OF PEDESTRIANS IoU: 60.7%
- TWO WHEELER IoU: 66.0%
- LARGE VEHICLE IoU: 47.2%
- STATIC IoU: 98.6%

Regardless of the great accuracy in the test dataset, one should pay close attention to the mean loss. The value of 59.02% in mean loss error may be misleading since it is not the lowest across all the epochs. One can wrongly select the epoch where the loss is the lowest because the generalization gap would be the smallest against the training loss. However, the loss serves as a secondary metric to add a better criteria and support IoU which is our primary metric for model selection. Figure 5.6 shows the mean loss in unseen data. One can see that at epoch 60 a lower loss is obtained compared to epoch 59 but Figure 5.7 shows a little improvement in IoU at such epoch 59. Then, since the IoU has not a relevant improvement, then we consider epoch 60 for getting the best model because the loss decayed in greater number. In both cases we should check the confusion matrix to pick between epochs because we will see the actual points predicted right. Finally, we took this stand because the data is imbalanced and the interpretation of values could be missed if just one metric is considered.



	CAR	PEDESTRIAN	PEDESTRIAN_GROUP	TWO_WHEELER	LARGE_VEHICLE	STATIC
CAR	163451	510	628	1137	15510	6634
PEDESTRIAN	288	30357	4797	940	53	2295
PEDESTRIAN_GROUP	542	24183	97883	1295	17	6052
TWO_WHEELER	1385	770	986	60184	74	885
LARGE_VEHICLE	13276	133	308	1069	78722	4065
STATIC	55453	39891	26775	16920	54387	1.71139e+07

Figure 5.4: Confusion matrix (absolute) for the baseline model in the test data set



True \ Predicted	CAR	PEDESTRIAN	PEDESTRIAN_GROUP	TWO_WHEELER	LARGE_VEHICLE	STATIC
CAR	0.87	0.0027	0.0033	0.0061	0.0826	0.0353
PEDESTRIAN	0.0074	0.7838	0.1239	0.0243	0.0014	0.0593
PEDESTRIAN_GROUP	0.0042	0.1861	0.7531	0.01	0.0001	0.0466
TWO_WHEELER	0.0215	0.012	0.0153	0.9362	0.0012	0.0138
LARGE_VEHICLE	0.1361	0.0014	0.0032	0.011	0.8068	0.0417
STATIC	0.0032	0.0023	0.0015	0.001	0.0031	0.9888

Figure 5.5: Confusion matrix (relative) for the baseline model in the test data set

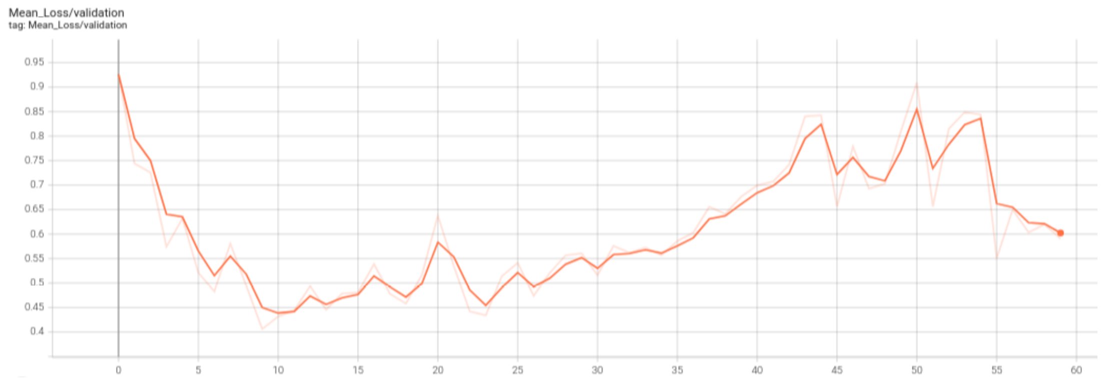


Figure 5.6: Loss error in the unseen data for our semantic network baseline model.

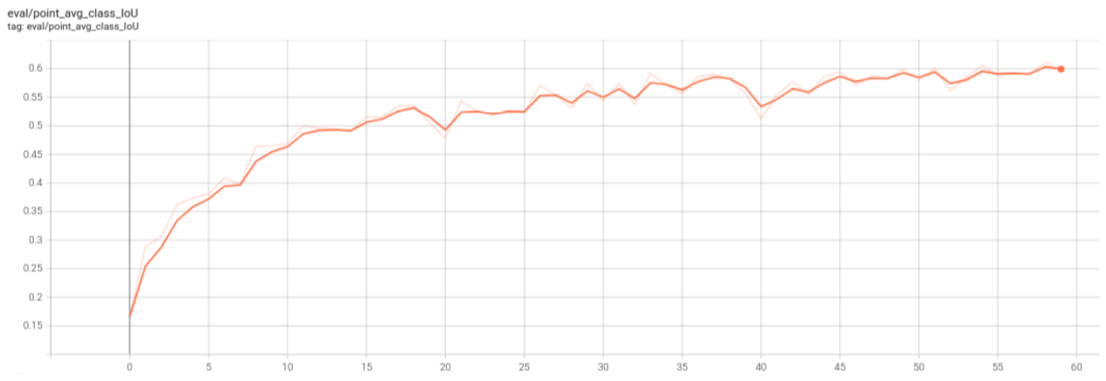


Figure 5.7: IoU in the unseen data for our semantic network baseline model.

5.2.2 Inference and Clustering

Table 5.1 shows the Average Precision (AP) scores for all the clusters found at each class after the semantic segmentation inference and posterior clustering in the same unseen data at point-wise IoU threshold of 0.3 (30%) and 0.5 (50%), respectively. The mAP and F1 score are shown too. Also, in Figure 5.8, two examples are shown,

Table 5.1: Metrics of AP per class

	AP(%)	
	IOU = 30%	IOU = 50%
car	53.29%	45.91%
ped	26.30%	23.05%
ped.group	44.41%	30.12%
bike	68.24%	66.35%
truck	40.98%	37.35%
mAP	46.64%	40.55%
F1	66.47%	60.99%

5.3 Comparison of Two Approaches

Table 5.2 presents the test results of two approaches with point-wise mAP and F1 score.

5.4 Discussion

A longer length of time accumulation improves AP of each class while length is below 700 ms and then APs show the tendency to decrease when the length is above 900 ms. The improvements on various classes also differ: It shows a huge AP increase for the class TWO_WHEELER and PEDESTRIAN_GROUP, whereas the increases on the other three classes are not as significant. It reveals that simply increase the accumulation time length will not significantly improve AP on the PEDESTRIAN class.

For the pipeline of PointNet++ and radar DBSCAN clustering, unfortunately as it is shown in Table 5.2, the model did not reach the mAP and F1 of 43.64% and 54.55%, respectively at $IoU = 30\%$, neither the mAP and F1 with 40.03% and 54.37%, respectively at $IoU = 50\%$ (presented in [36]). Our values, as shown in the same table, are at the half of the values of the mentioned values. We believe that the model is corrupted in the training stage because the labels are predicted wrongly and not as they are supposed to from the research document we cited before. From our best knowledge we believe that the pipeline is corrupted at the time the data

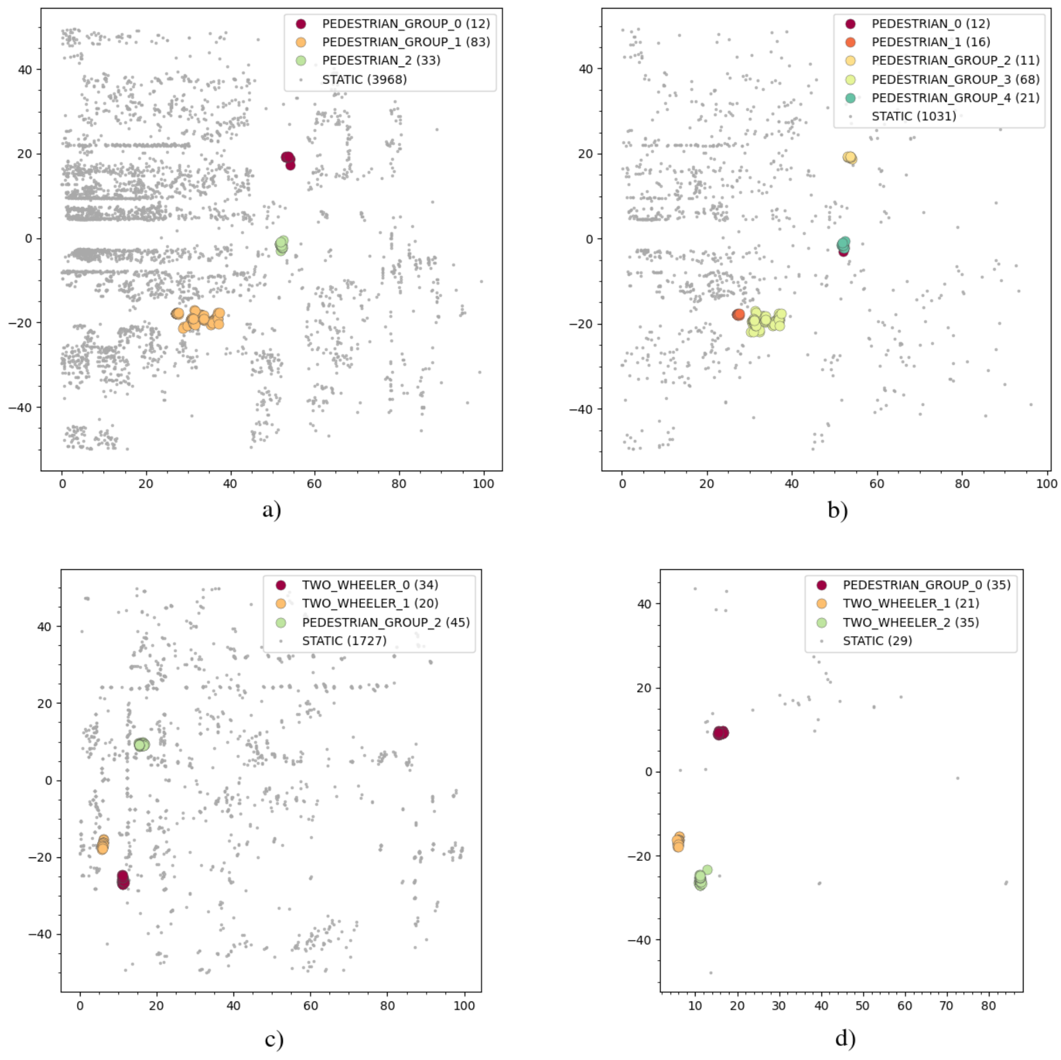


Figure 5.8: Inference (predicted labels and clustering) results on two typical traffic scenarios of two main approaches with ground truth clusters as the references. Ground truth are in the left hand side (a and c) and the corresponding predicted ones at the right hand side (b and d). The top two are the same snippet (a and b) and the two in the bottom are both the same (c and d). Note the effect of the filter in the predicted ones, also notice the little differences from a to b where there are more clusters than in the original plot.

is being handled and passed through the architecture. This is a totally solvable. However, for time reasons, we can expect that further collaborations may correct this error.

Small Object Detection

Table 5.2 shows that grid mapping and YOLO approach is good at detecting large road user class, including CAR, CYCLIST, and TRUCK; however, its performance on small road user class, including pedestrian and pedestrian group, is worse. We argue that this is because of the sparsity of the point cloud instead of the imbalanced class distribution. In Figure 3.8, we find that the instance numbers of PEDESTRIAN and PEDESTRIAN GROUP are 3-4 multiples of TRUCK and CYCLIST, however, the APs of PEDESTRIAN and PEDESTRIAN GROUP are roughly one half of the APs of TRUCK and CYCLIST. Therefore, the class distribution is not the main reason.

If we take a close look to the clusters, small road users clusters usually contains way less points than the big road user clusters. The RCS is also positively related to the size of the target according to [12]. Therefore, we calculate the statistics about RCS and number of points in each cluster. The result are shown in Table 5.3 and 5.4.

To analyze the influence on mAP of three factors, number of instances, average number of points in each instance, average RCS, we need to calculate the correlation coefficients. We regard the mean values in Table 5.4 and Table 5.3, the APs of five classes in 5.2 and number of instances in Figure 3.8 as random variables. Here is the result:

- $\rho_{mAP,instance} = -0.0408$
- $\rho_{mAP,RCS} = 0.2334$
- $\rho_{mAP,points} = 0.3302$

It indicates that number of points in each cluster has the highest linear correlation with AP and the average RCS are the second highest. The number of instances weakly negatively correlates to AP. This can provide some insights about how to improve AP on the pedestrian class. For example, interpolation of the points in pedestrian clusters can increase the density without changing the contour. However, the interpolation of position is relatively easy, the interpolation of RCS and Doppler radial velocity need to be further explored.

5.4.1 Weight Initialization

The number of strategies that one can apply to the network in order to make it better for predictions is large. However, these strategies can be reduced considering the theory behind the logic of each model.

One of these cases in the semantic network PointNet++ was the weights initialization in each layer (forward and backpropagation). According to [17] and [18], a conventional deep neural network struggles to converge when the weights are initialized using normal distributions with fixed standard deviations when the variance of the weights obtained is not considered. This normal initialization outputs huge or tiny activation values at the activation functions in every node, making them explode or vanish the gradient during backpropagation, i.e., the network goes unstable with unreasonable weight updates (when exploding) or stalls at some small values without updating the weights anymore (when vanishing). To tackle this problem, there have been invented and crafted weight initialization strategies that diminish these undesirables effects.

PointNet paper [32] does not specify a method for weight initialization by default. We decided to experiment with Xavier weight initialization since it maintains the variance at initializing the weights and it is one of the most used techniques in deep learning [16]. However, we observed that the mAP did not surpass the band of 30% and the mAP achieved at $IoU = 50\%$ in [36] was at least 39%. We decided then to use the Kaiming He weight initialization method [19] and the results were outstanding, surpassing the band of 46% at $IoU = 30\%$, and the band of 40% at $IoU = 50\%$. This is because the aforementioned method takes into account the activation function used in PointNet which is ReLU, and Xavier initialization is more suitable for sigmoid and $tanh$ activation functions.

It is worth mentioning that the He initialization, along with the weight bias initialization from dataset seen in subsection 4.2.3 improved the speed of training and made the network more robust and accurate. Moreover, the lack of one of these strategies resulted in suboptimal values every time.

5.4.2 DBSCAN Settings

We improved in a band of 2% to 4% the results of [36] at mAP and a band of 6% to 11% in F1 score in the same reference work. The improvement came thanks to two strategies:

- We changed the $e-region$ original formula showed in [36] by iterating at each dynamic class our new formula described in subsection 4.2.6 and showed in Equation (4.13). Thus, we could also avoid the range feature at each point used for determining the minimum number of points per cluster, suggested in [36].

- We decided to tune the filter showed in section 4.2.5 by using statistics in the dataset. Recall that the filter drop static points using the radial velocity as the threshold when a static point is predicted in PointNet++. We improved the shape of the objects by determining the mean radial velocity of the static points in order to drop a balanced number of static points and not all of them. This strategy helps the DBSCAN algorithm to include certain misclassified points as static by the network when in reality were dynamic class points. Thus, the clustering algorithm can accept or reject these wrongly static classified points into some cluster by looking to the radial velocity too.

Table 5.2: Result scores for all methods on the test set

Method	IOU=0.3							IOU=0.5						
	AP_{ped}	AP_{grp}	AP_{cyc}	AP_{car}	AP_{trk}	mAP	$F_{1,pt}$	AP_{ped}	AP_{grp}	AP_{cyc}	AP_{car}	AP_{trk}	mAP	$F_{1,pt}$
PointNet++/DBSCAN	26.30	44.41	68.24	53.29	40.98	**46.64**	**66.47**	30.44	35.61	62.27	46.63	26.17	**40.23**	**60.70**
YOLOv3/Gridmappings	51.62	15.32	42.98	67.36	58.28	**51.62**	**50.59**	13.379	37.66	65.93	71.77	56.94	**49.17**	**42.83**
$PointNet++/DBSCAN^3$	29.42	53.06	56.37	53.15	26.19	43.64	54.55	27.47	47.56	54.85	49.16	21.10	40.03	54.37
$YOLOv3/Gridmappings^3$	28.28	57.51	64.87	75.54	62.18	57.67	53.04	26.96	54.88	63.68	67.99	56.31	53.96	52.86

[1] IOU is point-wise, mAP and F1 sscore are both based on it. See definition in 2.7.1

[2] FOV: field of view, R-A: range-azimuth, R-D: range-Doppler, Num_{seq}: number of sequences

[3] Result from [8]

Table 5.3: Number of points in a cluster of each class

	Pedestrian	*PedGrp*	*Cyclist*	*Car*	*Truck*
max	75	213	117	295	680
min[1]	3	3	3	3	3
mean	16	30	28	38	97
std[2]	9	23	16	33	88

[1] Minimum amount of points is 3 because in Fig3.7 all the clusters without points have been removed
[2] Standard deviation

Table 5.4: RCS of points in a cluster of each class

	Pedestrian	*PedGrp*	*Cyclist*	*Car*	*Truck*
max	28.6	36.0	37.7	44.6	50.0
min	-30.6	-30.6	-30.6	-30.6	-30.6
mean	-8.4	-7.1	-10.3	-3.9	0.1
std[2]	6.8	6.7	7.2	11.3	10.5

[1] Unit:dBsm
[2] Standard deviation

6

Conclusion and Future Research

In this chapter, we will summarize all the work with a conclusion and give several perspectives about the future improvement.

6.1 Conclusion

In this thesis, we performed an object detection task on BEV radar point cloud focusing on dynamic road users by reproducing two of the methods that achieved the top results described in [8]. To this end, we examined two different approaches of existing deep neural networks on a large-scale real-world dataset. The first approach is an image-based object detector (YOLO v3) with grid mappings, where the point cloud will be first transformed to grid maps and then sent to train the model. The second approach is a semantic segmentation network (Poitnet++) combined with a traditional clustering algorithm (DBSCAN), where each point is assigned semantic class labels first and then grouped as instance clusters. We adapted the metric for point clusters to compare two approaches, including IOU, mAP, and F1 score. The reproduction of both approaches achieved a comparable result as in [8], among which the image-based object detector outperformed the other one. Therefore, we adopted the image-based object detector approach to investigate the effect of time accumulation on APs of all the classes. We found that low AP of the pedestrian class is the performance bottleneck, and accumulating a more prolonged period cannot significantly improve it.

The second approach, PointNet++ and radar DBSCAN clustering, was developed successfully with a few variants and with a little improvement, from the original work [36], in the metrics of mAP and F1. We could adapt a deep neural network with a clustering algorithm with success, which is also modular for other implementations and robust.

6.2 Future Research

New public dataset

Unlike large image datasets for the detection task, e.g., COCO [26], and other radar datasets mentioned in 3.1 , bounding boxes are not included as an annotation in [8] and thus have to be generated by the preprocessing as described in 4.1.1. This lead to several disadvantages for automotive applications: 1. The extracted bounding box dimension is time-dependent, i.e., the shape of bounding boxes will elongate if the accumulation time increases; 2. An exterior bounding box of a point cluster cannot authentically represent the real-world contour of a target object because the reflection points are denser on the side of the target near the ego-vehicle and sparser on the far side, 3. It introduces extra steps to the pipeline, i.e., bounding box extraction and box-cluster transform, thus increasing the computation burden. In summary, the lack of bounding box annotation in RadarScenes is a significant obstacle to applying image-based object detectors. We look forward to new public datasets with such an annotation.

Time Fusion

In 3.4, we describe the procedure to extract a 500-ms snippet as a training sample. The time length for frame accumulation, 500ms, is set empirically. Snippets are extracted sequentially from all the sequences, which will lead to different densities of target clusters in a snippet. A series of different values for frame accumulation has been tested as an ablation study. Furthermore, we could realize self-adaptable accumulation by the self-attention mechanism. [5] introduces the dynamic convolution operation by adding input-dependent weights to each convolution kernel as a lightweight self-attention operation. It boosted the representation power at little computation cost and could easily replace the standard convolution layer. Most importantly, the self-attention mechanism would allow the model to learn to assign a higher weight to keyframes and a lower weight to the frame where a new object emerges in the snippet. In other words, the model will learn a self-adaptable accumulation.

Visual Layer Perceptron and Long Short-Term Memory in the second approach (PointNet++/DBSCAN)

From [28], a way to enhance the accuracy of the semantic neural network PointNet++ is through visual-layer perceptrons. According to Liu, et al, one can modify the output of pointnet and adapt an extra layer capable of *Center Shift Vectors* and condense the points with the same predicted label closer to each other. This has advantages when it comes to using DBSCAN later because the clusters would be easier to generate with less errors. The work from [36] using Long Short-Term Memory (LSTM) cells in the PointNet++ architecture (to produce a recurrent neural network) is successful and it could be implemented along with the visual-layer perceptron work to enhance greatly the precision of the pipeline. The two of them work over the same foundations of this project. This means that our software could be used as a container for incorporating two new ideas mentioned.

Bibliography

[1] F. J. Abdu, Y. Zhang, M. Fu, Y. Li, and Z. Deng. Application of deep learning on millimeter-wave radar signals: A review. *Sensors*, 21(6), 2021. ISSN 1424-8220. doi: 10.3390/s21061951. URL https://www.mdpi.com/1424-8220/21/6/1951.

[2] D. Barnes, M. Gadd, P. Murcutt, P. Newman, and I. Posner. The oxford radar robotcar dataset: A radar extension to the oxford robotcar dataset. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6433–6438. IEEE, 2020.

[3] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11621–11631, 2020.

[4] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia. Multi-view 3d object detection network for autonomous driving. *CoRR*, abs/1611.07759, 2016. URL http://arxiv.org/abs/1611.07759.

[5] Y. Chen, X. Dai, M. Liu, D. Chen, L. Yuan, and Z. Liu. Dynamic convolution: Attention over convolution kernels. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11030–11039, 2020.

[6] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016.

[7] J. Dickmann, J. Lombacher, O. Schumann, N. Scheiner, S. Dehkordi, T. Giese, and B. Duraisamy. *Radar for Autonomous Driving – Paradigm Shift from Mere Detection to Semantic Environment Understanding: Von der Assistenz zum automatisierten Fahren 4. Internationale ATZ-Fachtagung Automatisiertes Fahren*, pages 1–17. 01 2019. ISBN 978-3-658-23750-9. doi: 10.1007/978-3-658-23751-6_1.

[8] M. Dreher, E. Erçelik, T. Bänziger, and A. Knol. Radar-based 2d car detection using deep neural networks. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–8. IEEE, 2020.

[9] X. Du, M. H. A. Jr., S. Karaman, and D. Rus. A general pipeline for 3d detection of vehicles. *CoRR*, abs/1803.00387, 2018. URL http://arxiv.org/abs/1803.00387.

[10] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. of 2nd International Conference on Knowledge Discovery and*, pages 226–231, 1996.

[11] Estimation lemma. Estimation lemma — Wikipedia, the free encyclopedia, 2010. URL https://en.wikipedia.org/wiki/Radar.

[12] Estimation lemma. Estimation lemma — Wikipedia, the free encyclopedia, 2022. URL http://en.wikipedia.org/w/index.php?title=Estimation_lemma&oldid=375747928. [Online; accessed 29-September-2012].

[13] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3354–3361. IEEE, 2012.

[14] R. Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[15] T. M. Giselsson, R. N. Jørgensen, P. K. Jensen, M. Dyrmann, and H. S. Midtiby. A public image database for benchmark of plant seedling classification algorithms. *arXiv preprint arXiv:1711.05458*, 2017.

[16] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In Y. W. Teh and M. Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL https://proceedings.mlr.press/v9/glorot10a.html.

[17] S. Goel. Medium kaiming he initialization. https://medium.com/@shauryagoel/kaiming-he-initialization-a8d9ed0b5899. Accessed: 2019-07-14.

[18] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[19] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015. URL http://arxiv.org/abs/1502.01852.

[20] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[21] Y. Henri. Itu world radiocommunication conference (wrc-15) allocates spectrum for future innovation. *Air & Space L.*, 41:119, 2016.

[22] C. Iovescu and S. Rao. The fundamentals of millimeter wave sensors. *Texas Instruments*, pages 1–8, 2017.

[23] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

[24] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12689–12697, 2019. doi: 10.1109/CVPR.2019.01298.

[25] M. Liang, B. Yang, S. Wang, and R. Urtasun. Deep continuous fusion for multi-sensor 3d object detection. *CoRR*, abs/2012.10992, 2020. URL https://arxiv.org/abs/2012.10992.

[26] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[27] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[28] J. Liu, W. Xiong, L. Bai, Y. Xia, and B. Zhu. Deep instance segmentation with high-resolution automotive radar. *CoRR*, abs/2110.01775, 2021. URL https://arxiv.org/abs/2110.01775.

[29] M. Noman, V. Stankovic, and A. Tawfik. Object detection techniques: Overview and performance comparison. In *2019 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)*, pages 1–5, 2019. doi: 10.1109/ISSPIT47144.2019.9001879.

[30] A. Ouaknine, A. Newson, J. Rebut, F. Tupin, and P. Perez. Carrada dataset: Camera and automotive radar with range-angle-doppler annotations. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 5068–5075. IEEE, 2021.

[31] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[32] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[33] 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*, 30, 2017.

[34] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[35] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[36] N. Scheiner, F. Kraus, N. Appenrodt, J. Dickmann, and B. Sick. Object detection for automotive radar point clouds–a comparison. *AI Perspectives*, 3(1): 1–23, 2021.

[37] O. Schumann, M. Hahn, J. Dickmann, and C. Wöhler. Semantic segmentation on radar point clouds. In *2018 21st International Conference on Information Fusion (FUSION)*, pages 2179–2186, 2018. doi: 10.23919/ICIF.2018.8455344.

[38] O. Schumann, J. Lombacher, M. Hahn, C. Wöhler, and J. Dickmann. Scene understanding with automotive radar. *IEEE Transactions on Intelligent Vehicles*, 5(2):188–203, 2020. doi: 10.1109/TIV.2019.2955853.

[39] O. Schumann, M. Hahn, N. Scheiner, F. Weishaupt, J. F. Tilly, J. Dickmann, and C. Wöhler. Radarscenes: A real-world radar point cloud data set for automotive applications. In *2021 IEEE 24th International Conference on Information Fusion (FUSION)*, pages 1–8. IEEE, 2021.

[40] O. Schumann, C. Wöhler, and K. Dietmayer. *Machine Learning Applied to Radar Data: Classification and Semantic Instance Segmentation of Moving Road Users*. Universitätsbibliothek Dortmund, 2021. URL https://books.google.se/books?id=DI93zgEACAAJ.

[41] J. Tanha, Y. Abdi, N. Samadi, N. Razzaghi, and M. Asadpour. Boosting methods for multi-class imbalanced data classification: an experimental review. *Journal of Big Data*, 7(1):1–47, 2020.

[42] D. Tian, Y. Han, B. Wang, T. Guan, H. Gu, and W. Wei. Review of object instance segmentation based on deep learning. *Journal of Electronic Imaging*, 31(4):1 – 18, 2021. doi: 10.1117/1.JEI.31.4.041205. URL https://doi.org/10.1117/1.JEI.31.4.041205.

[43] K. Wang, R. Cheng, K. Yang, J. Bai, and N. Long. Fusion of millimeter wave radar and rgb-depth sensors for assisted navigation of the visually impaired. page 5, 10 2018. doi: 10.1117/12.2324626.

[44] Y. Wang, G. Wang, H.-M. Hsu, H. Liu, and J.-N. Hwang. Rethinking of radar's role: A camera-radar dataset and systematic annotator via coordinate alignment. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2815–2824, 2021.

[45] X. Wu, D. Sahoo, and S. C. Hoi. Recent advances in deep learning for object detection. *Neurocomputing*, 396:39–64, 2020. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2020.01.085. URL https://www.sciencedirect.com/science/article/pii/S0925231220301430.

[46] D. Zhou, J. Fang, X. Song, C. Guan, J. Yin, Y. Dai, and R. Yang. Iou loss for 2d/3d object detection. *CoRR*, abs/1908.03851, 2019. URL http://arxiv.org/abs/1908.03851.

Master's Theses in Mathematical Sciences 2022:E65
ISSN 1404-6342

LUTFMA-3488-2022

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/