Radar-based Environment Perception for Automotive Applications

Advances in Artificial Intelligence Solutions for Radar Point Clouds

der Fakultät für Elektrotechnik, Informationstechnik und Medientechnik
der Bergischen Universität Wuppertal genehmigte

Dissertation

---

zur Erlangung des akademischen Grades
eines Doktors der Ingenieurwissenschaften

von

M. Sc. Alessandro Cennamo

aus

San Pietro V.co (Italy)

Wuppertal 2022

Tag der Prüfung: 15.07.2022
Hauptreferent: Prof. Dr.-Ing. Anton Kummert
Korreferent: Prof. Dr.-Ing. Tobias Meisen

Abstract

The last decade has witnessed a growing interest in autonomous driving applications. For this reason, vehicles have been equipped with more and more sensors – such as camera, lidar and radar – that enable perception of the surrounding environment. These sensors, however, provide raw data which do not automatically solve the perception tasks. Therefore, it is necessary the application of processing algorithms in order to extract information of interest, like the position of a vehicle or the color of a traffic light. This report is intended to provide innovations in the field of environment perception for automotive applications, with specific focus on Deep Learning (DL)/Machine Learning (ML) algorithms and radar data.

The first contribution presented in this document concerns the semantic segmentation of automotive radar point clouds. It consists in an architecture based on neural networks (NNs) that evolves the state-of-the-art by addressing intrinsic properties of the radar sensor. The resulting method, called RadarPCNN, proves superior performance on a custom dataset composed on real-world radar reflections. It exploits a pre-processing module to learn insightful object-related signatures from the radar features. Furthermore, it leverages the PointNet++ [64] framework, adopting the mean-shift (MS) algorithm during the sampling stage to optimize usage of the scarce spatial information. Finally, an attention mechanism is designed to merge local information extracted at different stages. All the solutions devised to improve the semantic segmentation performance on radar point clouds are validated and results effective, improving the state-of-the-art performance.

The stunning ability of point-wise processing methods to correctly classify object-reflections, notably pedestrians, has fueled the interest for a deeper investigation. Indeed, pedestrians are objects which are very difficult to recognize from radar data, due to their weak signature from the sensor perspective. For this reason, it has been decided to conduct a survey on the usage of radar features for the pedestrian detection task. The characteristics of PointNet++ [64], PointNet [65] and random forest (RF) are evaluated. It is proved that Doppler represents the most informative feature for the task. Moreover, it is shown that PointNet++ has unique usages of the radar features compared with the other approaches. Indeed, it is the only tested method to exploit the spatial point position to infer the pedestrian class prediction. Finally, it is shown that, despite Doppler severely influences the precision of the algorithm, the spatial coordinates are important to achieve high recall scores.

Though semantic segmentation provides details about the objects-distribution in the scene, it does not deliver exhaustive descriptions. For this reason, it has been decided to design a point-wise processing architecture that produce object-estimations

from raw radar point clouds. The proposed system optimizes the state-of-the-art radar solution [92] by means of a two-stage procedure. The first stage proposes points as object center-proposals and move them closer to the true center of the respective object. The second stage leverages the outputs of the first stage to estimate the object box-parameters. In particular, it discards the template-based formulation from the prior art in favor of a regression-aided one. Furthermore, it adopts a spherical regression task to estimate the box-orientation. Finally, the addition of a confidence output score enables filtering of the box-predictions in a post-processing step. The proposed model is compared with the prior art, proving better detection performance and box-accuracy, while using less operations. Moreover, it is successfully evaluated on a multi-class object recognition task, with various input settings.

Finally, limitations and weaknesses of DL/ML techniques are addressed. In particular, adversarial examples, i.e. malicious inputs purposefully altered to fool the system into a wrong outcome. This document introduces a novel adversarial detector that relies on statistical information from the training-set to identify malicious samples. Given an input, it extracts a signature by leveraging the prediction of a NN to several distorted versions. Then, it uses a novel metric to compare the signature with a class representative vector and compute the detection score. The proposed method results very effective in various attack settings, achieving state-of-the-art detection performance.

Scientific Contributions

This section contains the scientific contributions performed during the course of the PhD research period.

The following list contains a collection of articles published in conference proceedings:

[106] Alessandro Cennamo, Ido Freeman, and Anton Kummert. "A Statistical Defense Approach for Detecting Adversarial Examples". In: *Proceedings of the 2020 International Conference on Pattern Recognition and Intelligent Systems*. New York, NY, USA: Association for Computing Machinery, 2020. ISBN: 9781450387699. URL: https://doi.org/10.1145/3415048.3416103

[108] Alessandro Cennamo, Florian Kästner, and Anton Kummert. "Leveraging Radar Features to Improve Point Clouds Segmentation with Neural Networks". In: *Proceedings of the 21st EANN (Engineering Applications of Neural Networks) 2020 Conference - Proceedings of the EANN 2020, Halkidiki, Greece, June 5-7, 2020*. Ed. by Lazaros Iliadis et al. Vol. 2. Proceedings of the International Neural Networks Society. Springer, 2020, pp. 119–131. DOI: 10.1007/978-3-030-48791-1_8. URL: https://doi.org/10.1007/978-3-030-48791-1_8

[121] Alessandro Cennamo, Florian Kaestner, and Anton Kummert. "Towards Pedestrian Detection in Radar Point Clouds with Pointnets". In: *2021 International Conference on Machine Vision and Applications*. New York, NY, USA: Association for Computing Machinery, 2021, 1–7. ISBN: 9781450389556. URL: https://doi.org/10.1145/3459066.3459067

The following article has been selected for journal publication:

[120] Alessandro Cennamo, Florian Kaestner, and Anton Kummert. "A Neural Network Based System for Efficient Semantic Segmentation of Radar Point Clouds". In: *Neural Processing Letters* (2021). DOI: 10.1007/s11063-021-10544-4. URL: https://doi.org/10.1007%2Fs11063-021-10544-4

The following list contains a collection of works resulting in patent publication related to this thesis:

[91] Alessandro Cennamo et al. "Method and System for Determining whether Input Data to be Classified is Manipulated Input Data". European pat. 19182110.7. Aptiv Technologies Ltd. June 24, 2019. URL: https://data.epo.org/publication-server/document?iDocId=6430743&iFormat=0

[107] Alessandro Cennamo and Florian Kaestner. "Method and device for detecting objects". European pat. 20202567.2. Aptiv Technologies Ltd. Oct. 19, 2020

The following article not related to the contents of this thesis has been published in a conference proceeding:

[118] Marco Braun et al. "Semantic Segmentation of Radar Detections using Convolutions on Point Clouds". In: *Journal of Physics: Conference Series* 1924 (May 2021), p. 012003. DOI: 10.1088/1742-6596/1924/1/012003

The following list contains a collection of works resulting in patent publication which are not related to the contents of this document:

[111] Mirko Meuter et al. "Methods and System for Detection of Objects in the vicinity of a Vehicle". European pat. 20187674.5. Aptiv Technologies Ltd. Sept. 16, 2020

[122] Mirko Meuter et al. "Device and method for determining objects around a vehicle". European pat. 21190164.0. Aptiv Technologies Ltd. Aug. 6, 2021

[119] Alessandro Cennamo and Florian Kaestner. "Computer Implemented Method, Computer System and Computer Readable Medium for Object Detection in a Vehicle". European pat. 21195945.7. Aptiv Technologies Ltd. Sept. 10, 2021

Contents

List of Figures **IX**

List of Tables **XVII**

List of Abbreviations **XIX**

1 Introduction **1**
1.1 This Dissertation . . . 2
1.1.1 Motivations and Applications . . . 2
1.1.2 Challenges . . . 3
1.1.3 Achievements . . . 4

2 Theoretical Background **5**
2.1 Radar . . . 5
2.1.1 FMCW Radar: Sensor and Signal . . . 6
2.1.2 Processing Chain . . . 9
2.2 Deep Learning . . . 12
2.2.1 Neural Network . . . 13
2.2.2 Stochastic Gradient Descent and Back-propagation . . . 15
2.2.3 Tasks . . . 17
2.3 Deep Learning on Radar . . . 18
2.3.1 Classical Approaches . . . 19
2.3.2 Point Cloud Approaches . . . 19
2.3.3 Range-Doppler Map Approaches . . . 22

3 RadarPCNN: an Innovative Architecture for Semantic Segmentation of Radar Point Clouds **25**
3.1 Point Cloud Semantic Segmentation . . . 26
3.2 RadarPCNN . . . 27
3.2.1 Pre-processing Module . . . 28
3.2.2 Altered PointNet++ . . . 30
3.2.3 Attention Mechanism . . . 34
3.2.4 Classification Network . . . 35
3.3 Experimental Setup . . . 36
3.3.1 Dataset . . . 36
3.3.2 Settings . . . 38
3.3.3 Benchmarking Solutions . . . 41
3.4 Results . . . 42
3.4.1 Confusion Matrices . . . 45

3.4.2 Ablation Studies . . . 46
3.4.3 Visual Analysis . . . 56

4 A Survey on the usage of Radar Features in Pointnets for Pedestrian Detection 61
4.1 Radar-based Pedestrian Detection . . . 62
4.2 Radar Features in Point-wise Techniques . . . 64
4.2.1 PointNet++ . . . 65
4.2.2 PointNet . . . 65
4.2.3 Shared FC . . . 66
4.2.4 Random Forest . . . 67
4.3 Experimental Setup . . . 69
4.3.1 Dataset . . . 69
4.3.2 Experiment Procedure . . . 69
4.3.3 Network Configurations . . . 71
4.4 Results . . . 72
4.4.1 Feature Perturbation . . . 73
4.4.2 Doppler Analysis . . . 76
4.4.3 Focus on PointNet++ . . . 84

5 An Advanced Point-based Architecture for Detecting Objects in Radar Point Clouds 87
5.1 Environment Perception through Radar Point Clouds . . . 88
5.2 Multi-Class Object Detection Architecture . . . 90
5.2.1 First Stage . . . 92
5.2.2 Second Stage . . . 94
5.3 Experimental Setup . . . 104
5.3.1 Dataset . . . 104
5.3.2 Data-Association and Post-processing . . . 106
5.3.3 Settings . . . 107
5.3.4 Benchmark . . . 110
5.4 Results . . . 111
5.4.1 Box-Regression . . . 113
5.4.2 Multi-Class Task . . . 115
5.4.3 Visual Results . . . 118

6 A Novel Statistical Detector to Defend against Weaknesses of Deep Learning 123
6.1 Adversarial Examples . . . 124
6.1.1 Attack Methods . . . 125
6.1.2 Defense Strategies . . . 126
6.2 A Statistical Scheme to Detect Adversarial Examples . . . 128
6.2.1 Detection Concept . . . 128
6.2.2 Class Representative Vectors . . . 132
6.2.3 Similarity Score Function . . . 132
6.2.4 Distortions . . . 136
6.2.5 Detection Framework . . . 138
6.3 Experimental Setup . . . 138
6.3.1 Datasets . . . 139

6.3.2 Adversarial Attacks . . . 140
6.3.3 Prior Art Benchmark . . . 141
6.3.4 Classification Models . . . 142
6.4 Results . . . 142
6.4.1 White-box Results . . . 143
6.4.2 Compatibility with other Defensive Solutions . . . 145
6.4.3 Distortions Configuration . . . 147
6.4.4 Black-box Results . . . 149
6.4.5 Ablation Studies . . . 149

7 Conclusions **153**

Bibliography **157**

Appendix A **169**
A.1 Point-Sampling with MS and FPS . . . 169
A.2 Visual Examples of RadarPCNN Predictions . . . 170
A.3 RadarPCNN Predictions: Bushes as Pedestrians . . . 173
A.4 Visual Examples of Object Box-Predictions . . . 174

List of Figures

2.1 Saw-tooth modulated FMCW radar signal. The Tx signal (black) consists in a ramp sweeping frequencies in the bandwidth B. The Rx signal (gray) experiences a frequency shift Δf proportional to the round-trip time delay Δt. The Doppler effect produces a frequency shift f_D corrupting the range measurement. 6

2.2 AoA framework. The target S reflects the Tx EM wave. When it gets back to the sensor (bottom), a flat wavefront can be assumed. Each antenna element receives the signal with a different phase-shift $\Delta\phi$, proportional to θ – i.e. the AoA. 8

2.3 FMCW radar analog processing. A voltage ramp steers a VCO. The Rx signal is down-mixed with the Tx one to recover the beat frequency. After filtering out the high frequency component, the signal is converted by a ADC. 10

2.4 FMCW radar processing example. Tx (black) and Rx (gray) signals are down-mixed together generating a signal which oscillates at the beat frequency. This signal is discretized and arranged in a matrix where each column contains samples from the same chirp period. DFT across the columns of the matrix (red marked cells) extracts ranging information. A second DFT across the rows of the matrix which results from the first DFT (green marked cells) produces velocity information. The outcome of this operation is the RD-map. 10

2.5 Radar data. The same scene represented by RD-map (front-right sensor) and point cloud. Best viewed in colors. 11

2.6 Overview of FMCW radar digital processing chain. The discretized signal is processed with two dimensional DFT (fast- and slow-time) to produce RD-maps. The CFAR algorithm detects target signals from noisy ones. Finally, an AoA algorithm extracts angular information to generate a point cloud of detections. 12

2.7 Graphical illustration of a fully-connected NN. 14

2.8 Computational graph of a network during training. The dashed gray arrows show the back-propagation path with corresponding derivatives. 17

2.9 PointNet architecture. Shared FC layers ensure learning of order-invariant features. Max-pooling extracts a global descriptor from point-features. By means of a shared FC classification network it is possible to perform classification or segmentation. The $\oplus$ symbol defines the concatenation operation. 21

2.10 PointNet++ SA layer operations. The sampling process selects some of the input points (star-marked). Then, the points which lie nearby are grouped together, forming a local point cloud. Finally, the local point clouds are processed with PointNet to extract a signature for each group. 22

3.1 Lidar and radar point clouds describing the same scene where the ego-vehicle is turning left at a crossroad. Black rectangles represent objects of interest. 27

3.2 RadarPCNN architecture. The proposed pre-processing module learns features for each point. Mean shift is used to alter the sampling process of a single-stage PointNet++. An attention mechanism fuses point signatures and a shared FC-network produces the class scores. Best viewed in color. 28

3.3 The proposed pre-processing module. The input features of every point are combined, by means of a shared FC-network, to obtain a richer feature-set. 29

3.4 Mean shift algorithm applied to a one-dimensional problem. The red points represent the data observations x_i. The black-dotted lines are the kernels applied to each data-point. Gaussian kernels in eq. (3.8) are used, with $h = 0.8$ and $c = h\sqrt{\pi}$. The green curve is the distribution $f(x)$ computed as in eq. (3.7). The blue points are the local maxima selected by the algorithm. Best viewed in color. 32

3.5 Attention mechanism. It fuses together different signatures of the same point through a learned weighted combination. Best viewed in color. 35

3.6 Sensor mounting configuration and coverage. The ego-vehicle is shown at the origin. Different sensors are represented by triangle markers with different colors, oriented according to the sensor. Colored lines indicate the boundaries of the FoV. Gray regions shows the area covered by the sensors. Darker gray areas result from the superposition of multiple sensors. Best viewed in color. 37

3.7 Confusion matrices of RadarPCNN and PointNet++ [81]. The colormap shows the points count: the darker the color, the higher the points number (in brackets). Best viewed in color. 46

3.8 Comparison between MS and FPS under various configurations of the number of representative points. Best viewed in color. 49

3.9 Examples of representative points sampled with MS and FPS. Radar reflections are represented with points, colored based on their RCS value. Red stars show the space location sampled. Objects present in the scene are marked with black boxes. Best viewed in color. . . . 51

3.10 2D examples of the proposed grouping techniques. Black-dots represent the data-points, while white stars indicate representative points. Dashed-shapes represent the neighborhood boundaries. Circles account for 2D projections of 3D spheres, while ellipses illustrate 2D projections of ellipsoids. 52

3.11 2D examples of the grouping configuration implemented. The same color coding of fig. 3.10 is adopted. The gray box represents a vehicle. The spherical method uses a simple sphere. The dynamic ellipsoid method has the ability to learn the parameters of the ellipsoid. The static method uses five different configurations and learns the best combination through an attention mechanism. 53
3.12 RadarPCNN predictions. Yellow/red points are moving vehicle/pedestrian predictions. Ground-truth BBs follow the same color-map. Gray BBs belong to stationary vehicles, while orange BBs mark vehicles with absolute speed between 1 and 2.5 m/s. The z-component is zeroed and aligned with the road for visualization clarity. Left. Front/rear camera view. Right. 3D visualization. Best viewed in color. 58
3.13 RadarPCNN predictions. Bushes points on the right of the ego-vehicle predicted as pedestrian. The same color-coding as in fig. 3.12 is adopted. Left. Right/front camera view. Right. 3D visualization. Best viewed in color. 59
3.14 Histogram of Doppler values from bushes points falsely predicted as pedestrian (orange bars) and stationary points correctly predicted (blue hatched bars). The black dotted line shows as reference the distribution of Doppler values from GT pedestrian points. Best viewed in color. 60

4.1 Example of decision tree diagram for a binary classification task. Black boxes denote computational nodes, while gray boxes show leave nodes. x_i refers to the i-th feature of $\boldsymbol{x}$, while γ_i represents the corresponding threshold. Since the same input feature can be used by different nodes, a super-script index is used to identify the threshold for a specific node. 67
4.2 Models performance on Doppler (solid) and XY (dotted) altered test-set with different noise magnitudes. Every color represents a different model. Doppler results in the most important feature while PointNet++ is the only model using XY information. Best viewed in colors. 74
4.3 Models performance on RCS altered test-set with different noise magnitudes. Cross markers report the models performance while completely destroying the information. Best viewed in colors. 75
4.4 Doppler distributions of pedestrian predictions. Different colors represent different models. The dotted black curve refers to the Doppler distribution of GT pedestrian points. Best viewed in color. 77
4.5 Possible actions of moving pedestrians. The sensors are identified with triangle markers, oriented according to the mounting setting. The green arrows show the absolute velocity of the pedestrian. Red and cyan arrows represent, respectively, the tangential and radial components of the absolute velocity. Radar Doppler measures the length of the cyan arrow. Best viewed in colors. 79
4.6 Doppler distributions of missed pedestrian points. Different colors represent different models. The dotted black curve refers to the Doppler distribution of GT pedestrian points. Best viewed in color. . 81

4.7 Models performance vs noise magnitude under random (solid) and shift (dotted) noise applied to the Doppler feature. Best viewed in colors. 83
4.8 Column-chart showing the distribution of pedestrian predictions or GT points across different classes vs different noise magnitudes. Column-segments of different colors represent different classes. Best viewed in color. 85

5.1 Examples of various type of environment perception tasks. Points represent radar observations while gray boxes identify GT objects in the scene. The point color shows the semantic segmentation output class: background (black), vehicle (green) and pedestrian (red). Colored dashed contours identify regions which contain points from the same physical instance. Colored dashed boxes refer to predicted box-object proposals. Best viewed in color. 88
5.2 Overview of the proposed architecture. 91
5.3 First stage of the proposed architecture. The pre-processing module computes deep point-descriptors abstracting the radar features. PointNet++ is used to encode information of local spatial proximity between points. For every point, it is computed a center-proposal score and an offset vector. 92
5.4 GT for learning the outputs of the first stage. The gray rectangles represent GT boxes and the red stars show the object centers. The green and orange colored areas define, respectively, center- and transition-region. Green points are labeled as object centers, while orange points are masked out. Black points are assigned to the background class. The gray color marks generic points. Green arrows show the GT offset associated with each point. Best viewed in color. 93
5.5 Second stage of the proposed architecture. The input is shifted using the offset predictions from the first stage. The center proposal output is used to tamper the sampling process of an SA layer. FPS is used to down-select the center proposals and generate object proposals. The grouping and feature extraction (PointNet) operations compute a feature-descriptor for each object proposal. Four output layers estimate the BB parameters. 95
5.6 Visual example of the network operations of the second stage. Points are colored according to the center proposal output of the first stage. Yellow represents vehicle-center, while gray points are classified as background. Boxes correspond to GT objects in the scene. The green color marks moving vehicles, orange for slowly moving vehicles and pedestrian boxes are denoted in red. Best viewed in color. . . . 96
5.7 Example of residual offset. The gray rectangle represents a GT BB and the red star its center. The points are center proposals. The green points are the ones which survive the FPS selection. Green arrows show the GT offset vectors while red arrows display an example of not accurate offset predictions. Blue crosses determine the location of the green points after being shifted. Cyan arrows show residual offset vectors. Best viewed in color. 98

5.8 Yaw-regression head. The gray box represents an object and the red star its center. The orange vector shows the yaw-orientation of the box. Blue and green arrows display the network predictions. The green area represents the classification output which permits the estimation of the red arrow. Best viewed in color. 101

5.9 Computation of the approximated IoU. The gray and green boxes represent, respectively, the GT and estimated BBs. Arrows display the orientation of the boxes. The red box shows the estimated BB after alignment with the GT yaw. The yellow area represent INT_{align}. The solid and dashed lines in the plot show, respectively, the approximated and exact IoU score. Every line corresponds to a different configuration. Size estimation is considered exact – i.e. both GT and estimated box are $4.5m$ long and $2m$ wide. The position of the predicted box is changed by the values reported in legend – e.g. $xy = (1, 0)$ indicates that the predicted box has a $1m$ estimation error along the x-axis. Best viewed in color. 103

5.10 Distribution of the size of BB from different classes. The blue lines refer to the width parameter, while the red lines report statistics about the length. Best viewed in color. 105

5.11 Average IoU score of proposed solution and prior art on different values of confidence threshold. Different network configurations are shown with different colors. Best viewed in color. 113

5.12 Yaw RMSE of proposed solution and prior art on different values of confidence threshold. Different network configurations are shown with different colors. Best viewed in color. 115

5.13 PR curves produced by the proposed archicture. Different configurations are reported in different colors. The solid lines show the performance on moving vehicles, while the dashed lines refer to the pedestrian class. The performance of the prior art models on moving vehicles are reported for reference – cross and diamond markers. Best viewed in color. 116

5.14 Average IoU score of the proposed architecture on different values of confidence threshold. Different network configurations are shown with different colors. The solid lines show the score on moving vehicles, while the dashed lines refer to the pedestrian class. Best viewed in color. 117

5.15 Visual examples showing the location estimation task performed by the proposed architecture. Cyan arrows represent offset vectors predicted by the first stage. Red arrows show the residual offset vectors estimated by the second stage. Yellow points are center proposals, while gray points are classified as background. The blue color marks the center of object-predictions. Boxes represent GT vehicles: moving (green) and slow-moving (orange). Best viewed in color. 119

5.16 Visual example of the box-predictions produced by the proposed architecture. The same scene is proposed before and after NMS. Green boxes show moving vehicle predictions, while red cylinders represent pedestrians. GT boxes are reported in gray to avoid visual confusion. Gray points show the input point cloud fed to the network. Left. Front/rear camera view. Right. 3D visualization. Best viewed in color. 121

6.1 Instance of an adversarial example. The legitimate image is correctly classified as penguin. By altering it with an adversarial perturbation, the system is misled to predict the monkey class with high confidence. Yet, both images look natural and identical to a human. Best viewed in color. 124

6.2 Orthogonality property of eq. (6.22). Legitimate samples produce signatures aligned with their class-representative vector. Adversarial examples result in signatures perpendicular to the class-representative vector selected. Best viewed in color. 134

6.3 Histogram of detection scores extracted using eq. (6.26) for legitimate (black) and adversarial (hatched fill) samples. Inputs are sampled from the GTSRB test-set and perturbed using the C&W method. A single distortion is used (median smoothing). 135

6.4 Visual effects of the distortions on a legitimate sample of the GTSRB dataset. Best viewed in color. 136

6.5 Scheme of the proposed detection framework. The input is distorted to extract its signature. The original input class prediction is used to select the class-representative vector. The normalized projection metric computes the detection score. A thresholding operation determines whether the input is legitimate or adversarial. 139

6.6 A GTSRB sample attacked with the methods used for evaluation. Best viewed in color. 140

6.7 ROC curves of the proposed detector (blue) and FS (orange) against the C&W attack on the CIFAR10 test-set. Both the detectors use median filtering and reduction of the bit-depth as distortions. Best viewed in color. 144

6.8 ROC curves showing the effects of the proposed solutions against the C&W attack on the CIFAR10 dataset. The orange curve is produced by FS – eq. (6.30) – and the gray one by the proposed detector (using all the solutions) – eq. (6.22). The blue curve shows the effects of the proposed normalized projection score metric in eq. (6.26). The green curve is produced using the detection criteria in eq. (6.31) and shows the improvement due to the first-order statistics as class representative vectors. Median filtering and reduction of the bit-depth resolution are used as distortions. Best viewed in color. 150

A.1 Both methods sample most of the objects of interest. Yet, MS provides a better description of the space, while FPS results more uniform. 169

A.2 MS produces a very clean scene with the sampled points, while FPS provides a corrupted representation. Moreover, MS respects the scene, sampling points for all vehicles. FPS misses some of them. Pedestrians are not all sampled due to the sparse sampling strategy – another layer of the network focuses on such smaller objects. 170
A.3 Incoming moving vehicle correctly detected by the network. Points from all the other parked vehicles are classified as the negative class. The pedestrian on the rear does not have any associated reflection. Since it is occluded by a parked car, the sensor could not see this instance. Top-left: front camera. Bottom-left: rear camera. Right: 3D view. 171
A.4 The scene contains only static objects and the network can correctly predict the negative class for every point. Top-left: front camera. Bottom-left: rear camera. Right: 3D view. 171
A.5 A complex scene containing both moving and stationary vehicles at a big cross-road. The network can correctly recognize as static the cars in front of the traffic light (on the left). Both large and small moving instances are classified as moving vehicles. The different altitude levels generate visual-errors in the camera views. Top-left: left facing camera. Bottom-left: front camera. Right: 3D view. 172
A.6 A pedestrian in front of the ego-vehicle, near parked cars, correctly detected by the network. The rest of the scene is classified as static. Top-left: front camera. Bottom-left: rear camera. Right: 3D view. . 172
A.7 Bushes from a hedge near the pedestrian are erroneously classified. They share the spatial properties of the nearby pedestrian: both instances are found next to stationary objects. Top-left: left facing camera. Bottom-left: front camera. 173
A.8 Bushes on the left and right of the ego-vehicle are mistakenly predicted as pedestrians. Top-left: left facing camera. Bottom-left: right facing camera. 173
A.9 Downtown dense scene. The only moving vehicle is detected. Though there are some false positives, most of the pedestrians are detected. The other vehicles in the scene are stationary parked cars. Top-left: front camera. Bottom-left: left-facing camera. Right: 3D view. . . . 174
A.10 Downtown scene. Two moving vehicles are correctly recognized. The third moving vehicle in the rear is not detected because it provides a single radar reflection. The pedestrians along the curbstone on the right are detected, but the one in the rear has a location error of few meters. The group of pedestrians in the rear are also correctly recognized, although they are quite far away. The three vehicles on the right represent stationary cars parked in a private area. Top-left: front camera. Bottom-left: rear camera. Right: 3D view. 175
A.11 Urban scene. Moving vehicles and the pedestrian in front are correctly recognized. The two vehicles on the top of the scene are not detected because they have none or few radar reflections. Predictions with correct orientation have higher confidence. The vehicles on the left are standing behind a red traffic light. Top-left: front camera. Bottom-left: rear camera. Right: 3D view. 176

A.12 The pedestrian on the rear-right size and the moving vehicles in the ego-car lane are detected. The vehicles on the left are standing behind a red traffic light. The pedestrian on the rear-left side is missed. The pedestrians on the front-left side cannot be detected because they have no reflections. The network can detect objects very far away, such as the vehicle at the top of the scene. Top-left: front camera. Bottom-left: rear camera. Right: 3D view. 177
A.13 Highway scene. Vehicles in the ego-car roadway are recognized. Vehicles in the incoming roadway are not detected because they do not provide any radar reflections. The network can predict boxes for objects far away. Though multiple boxes are estimated for the vehicles in the rear, the most accurate ones obtain larger confidence values. Top-left: front camera. Bottom-left: rear camera. Right: 3D view. . 178

List of Tables

3.1 Distribution of points in the dataset. Percentage and total number of points are reported. 38
3.2 Semantic segmentation results on the radar dataset (%). Inference time is computed on Nvidia GeForce RTX 2080 (ms). 43
3.3 RadarPCNN performance using different fusion methods (%). Inference time is computed on Nvidia GeForce RTX 2080 (ms). 48
3.4 RadarPCNN performance using different grouping methods (%). Inference time is computed on Nvidia GeForce RTX 2080 (ms). 55
3.5 RadarPCNN performance under different usage of the Doppler feature (%). 57

4.1 Summary of the models evaluated on the original test-set. F_1 scores on the positive classes are reported in percentages. 73
4.2 F_1 score performance of the models trained with and without the RCS feature. RCS is useful for the semantic segmentation task. 76

5.1 Statistics of box-labels from vehicle and pedestrian objects in the dataset (train and test set). The count columns report the total amount of instances in the dataset as well as the average and maximum per-frame count. The length and width columns, instead, contain information about the object sizes (in meters). Statistics of both moving and stationary objects (All) as well as only moving objects (Mov.) are reported. 105
5.2 Comparison between proposed approach and prior art on the moving vehicle detection task. In brackets are reported the statistics of the first stage network (first and second stage for the prior art). 112

6.1 Performance of the tested classifiers against every test-set (legitimate and adversarial) with no defense deployed. Accuracy and average prediction confidence are reported in percentage. Top: White-box results. Bottom: Black-box results. 143
6.2 AUC scores of FS and the proposed detector against the various attack-sets (%). 145
6.3 Performance of the victim models with and without adversarial training in terms of accuracy (%). 146
6.4 AUC scores of the proposed detector and FS with and without adversarial training (%). 147

6.5 AUC scores of FS and the proposed detector for different choices of the distortions (%). The configuration with two distortions uses median smoothing and reduction of bit-depth resolution. 148
6.6 Detection rates of FS and the proposed detector in black-box settings (%). 149

List of Abbreviations

ACC Adaptive Cruise-Control.

ADC Analog-to-Digital Converter.

AI Artificial Intelligence.

AoA angle-of-arrival.

AUC area under the curve.

BB bounding-box.

BV beam-vector.

C&W Carlini and Wagner.

CDC compressed data-cube.

CFAR Constant False-Alarm Rate.

CM confusion matrix.

CNN convolutional neural network.

CUT cell-under-test.

CV computer vision.

CW Continuous-Wave.

DBSCAN Density Based Spatial Clustering for Applications with Noise.

DF DeepFool.

DFT Discrete Fourier Transform.

DL Deep Learning.

EM electromagnetic.

ESPRIT Estimation of Signal Parameters via Rotational Invariance Techniques.

FC fully-connected.

FFT Fast Fourier Transform.

FGSM Fast Gradient Sign Method.

FLOPS floating-point operations.

FMCW Frequency-Modulated Continuous-Wave.

FoV field-of-view.

FP feature-propagation.

FPS farthest point sampling.

FS Feature Squeezing.

GT ground-truth.

GTSRB German Traffic Sign Recognition Benchmark.

HGD High-level representation Guided Denoiser.

IoU Intersection-over-Union.

KNN K-nearest-neighbors.

LPF low-pass filter.

LSTM long-short term memory.

ML Machine Learning.

MLP multi-layer perceptrons.

MS mean-shift.

MSE mean squared error.

MUSIC MUltiple SIgnal Characterization.

NMS non-maxima suppression.

NN neural network.

OGM occupancy grid-map.

RAD range-antenna-Doppler.

RCS radar-cross section.

RD-map Range-Doppler map.

ReLU Rectified Linear Unit.

RF random forest.

RMSE root mean squared error.

RNN Recurrent neural network.

ROC Receiver Operating Curve.

Rx received.

SA set-abstraction.

SGD stochastic gradient descent.

SVM Support Vector Machine.

Tx transmitted.

VCO Voltage-Controlled Oscillator.

VRU vulnerable road user.

Chapter 1

Introduction

The automotive sector represents one of the fastest growing industries in the world. In particular, the last decade has shown an increasing interest in autonomous (or self-driving) vehicles which represent the major innovation in the sector. This technological advancement has the potential to produce more comfortable services, but also to address and solve critical safety concerns. Indeed, studies have shown that the majority of car accidents can be attributed to human errors which could have been prevented by machine operating in their place [27].

In order to achieve fully-autonomous driving capabilities, a system must be able to build an environmental model: i.e. obtain a complete picture of what is going on around it. This can be accomplished by means of sensors. Modern vehicles are equipped with a broad package of sensors, ranging from camera to lidar, radar and ultrasonic. Each sensor has its own perks. For instance, camera generates very accurate semantic information, enabling the discrimination of target properties – e.g. colors – which is not possible with other sensors. Lidar provides highly precise ranging information, detecting the presence of objects up to hundreds of meters away. Radar is less accurate than lidar, but covers larger distances and provides velocity information. Ultrasonic sensors describe the immediate neighborhood of the vehicle with very high precision. Nevertheless, it is the combination of all these components that enables detailed perception of the surrounding environment. Indeed, by equipping a vehicle with a suite of sensors, it is possible to detect distant objects as well as recognize traffic signs, thus providing the means for achieving human-level awareness of both the scene and its actors.

In humans, sensing and perception processes happen seamlessly: one glance at the scene and the targets of interest are suddenly spotted and recognized. The same does not hold true for sensors. They are devices that sense the environment, producing as output data that need to be elaborated in order to extract the information of interest. In autonomous vehicle systems, sensor-data are generally transmitted to a perception module. This contains the algorithms which process the data and is responsible for extrapolating the knowledge required to build a model of the environment. For this reason, the task performed by the perception module is considered highly safety-critical. Note, indeed, that most autonomous functionalities builds on its outputs, which directly determine the actions to undertake for a smooth drive.

There exist a vast variety of algorithms for processing automotive sensor data. However, in the last decades, Artificial Intelligence (AI) methods have emerged among the others for their abilities to solve very advanced and complicated tasks.

They are designed to mimic the human awareness-building process: supported by Machine Learning (ML)/Deep Learning (DL) technologies, these algorithms have the ability to learn from experience. As a result, these methods can produce very detailed descriptions of the scene: e.g. they can provide information about the limits of the roadway as well as detect position and shape of relevant objects along with their future behaviors. Therefore, AI enables the perception module to build an accurate and reliable model of the surrounding environment, which can then be used to take appropriate actions. For instance, the vehicle control system can steer the wheel to make a turn when approaching a bend or execute an emergency braking to avoid a kid which has abruptly entered the roadway.

1.1 This Dissertation

The perception system of autonomous vehicles offers various interesting research topics. This report is focused on the algorithms, rather than the sensing elements feeding the module. In particular, it has been decided to study the opportunities offered by the radar sensor, investigating possible methods to leverage the data for various perception tasks. From an algorithmic perspective, instead, it has been decided to focus on AI technologies, investigating their ability to operate on the radar domain and addressing their limitations and weaknesses.

1.1.1 Motivations and Applications

Among the various sensors with which a vehicle is equipped, radar has unique properties. Indeed, despite it provides less accurate ranging information than lidar, it possesses certain characteristics which are not found in other sensors. For instance, radar maintains reliable and robust measurements in a broad range of operating conditions [103]. Camera and lidar rely on visible light to collect data. This makes them very similar to the human sensing system, but forces the same drawbacks. Indeed, critical weather conditions – such as fog or heavy rain – significantly deteriorates the capacity of the sensors to observe the environment. Even worse effects take place in case of direct sunlight or very dark environments[1], with the sensors resulting completely blind. Contrarily, radar is barely affected by adverse operating conditions. For this reason, it can support semi-autonomous applications, providing a valid backup to the human driver when the visibility is scarce. Other unique properties of radar sensors can be found in their measurements. Indeed, it can record targets up to few hundreds of meters away. In addition, it can instantaneously measure velocity, a very characteristic target property which results very helpful during the perception stage. Finally, radar can be placed behind vehicle body panels – away from dirt and water – and is substantially cheaper than lidar technologies.

Over the course of the years, radar has been progressively more involved in the automotive industry, enabling a broad variety of applications. A well-know example is the Adaptive Cruise-Control (ACC) system [17] which can be found in many consumer vehicles. ACC allows the vehicle to adapt its velocity based on the traffic conditions around. In order to determine the action to take, it relies on radar sensors to detect objects and keep track of their distance to the vehicle.

[1]Lidar can work in scarce visibility conditions (e.g. nighttime) as it is its own source of light.

Solutions like the ones proposed in chapters 3 and 5 can be used to assist this functionality. Another widespread application which is supported by radar is the Automatic Emergency Braking (AEB) system [28]. AEB works as an emergency functionality, breaking off in case an obstacle is detected on the roadway in the immediate proximity of the vehicle. Studies have proved that AEB systems have the ability to avoid serious accidents [39], saving lives in case pedestrians are involved [34]. For these reasons, it is important to obtain a better understanding about how the perception module operates, especially for the detection of vulnerable road user (VRU) such as pedestrians, as proposed in chapter 4.

The applications mentioned so far require that the vehicle pays attention to a specific region of the surrounding environment. Since the task is limited to a handful of different outcomes, it can be efficiently executed by means of traditional algorithms. However, in order to enable more advanced applications, it is necessary to solve more complicated perception tasks. This can be achieved by means of AI solutions. AI algorithms differ from traditional methods as they use programs that do not provide instruction about how to perform the task. Indeed, an AI algorithm consists in a set of rules that specify the process for learning patterns from the data. Once identified the most relevant patterns and their relation with the solution of the task, the algorithm builds a model which contains the rules to best execute it. This procedure provides a major advantage over traditional algorithms, as the programmer does not need to cover every corner case. Indeed, traditional programs require hard-coding of the rules to follow, covering every probable scenario that the system might encounter. Instead, by learning from the data, AI algorithms can generalize their knowledge about the task to manifold scenarios.

In the last decades, AI solutions have spread across a broad range of sectors. They have been adopted as virtual voice assistants [51] as well as for performing medical diagnosis [57]. Yet, it is only in the recent years that they have found their place in the automotive industry [22, 38], enabling various advanced applications such as driver-monitoring [117], traffic sign recognition [95] and classification of different road users [98]. For instance, in a Traffic Sign Recognition (TSR) system [95] the algorithm takes as input a raw image and outputs the traffic sign contained. In this way, the vehicle can take appropriate actions to comply with the Highway Code – e.g. adjusting the velocity to respect a 50 km/h speed-limit sign.

Finally, this report provides an investigation on AI applied to camera-data. Camera is a sensor which complements very well with radar. Indeed, it does not provide any ranging or velocity information, but enables discrimination of various object properties which are not detectable with other sensors. For instance, camera allows recognition of traffic signs or the color of a traffic light. In addition, AI is renowned for working particularly well on camera-data, as certified by the extensive literature research [124, 115, 101, 100, 66]. Consequently, it is possible to perform a deeper investigation on the algorithm, capturing fine details about its operations. For instance, camera-data enables the exposure of weaknesses and limitations of this algorithm, which can then be addressed and solved, as proposed in chapter 6.

1.1.2 Challenges

The last decades have witnessed an unprecedented expansion of AI, establishing new benchmarks in various tasks. Yet, although research and advancements on certain

domains have grown rapidly, others have been lagged behind. Due to the shortage of publicly available data, radar falls in this latter category, showing a significant underdevelopment in terms of AI solutions. Therefore, despite the ability of AI to generalize to other domains, the major challenge faced in this report is represented by the limited availability of research in the radar domain.

Radar provides data in two different formats: low-level Range-Doppler maps (RD-maps) or point clouds. The former have some similarities to camera-data: they are both structured data-formats which are well-suited for being processed with standard AI techniques. However, radar "images" display the energy distribution across range and velocity spectrum, which is drastically different than the information-content of conventional camera-images. Point clouds, instead, lacks the structure proper of image-data, thus preventing the adoption of standard camera-based techniques. In order to process point clouds, therefore, it is necessary to explore AI techniques devised for other domains. Lidar is another sensor which produces data in the form of point clouds. Since it presents a comprehensive literature research, it would be straightforward to utilize lidar-based AI techniques on radar point clouds. However, the information provided by these two sensors are critically different. Lidar point clouds contain several millions of points which accurately follow the boundaries of the objects. As a result, it is possible to leverage information of proximity between points to recover details from the data. Contrarily, radar point clouds are sparser: the targets are represented by dozens of points and the information is embedded in the radar features rather than the point spatial distribution. Furthermore, compared to both camera and lidar, the radar sensing technology provides data which are inherently noisy. Because of all these reasons, it is not possible to simply generalize AI algorithms to radar data, but it is necessary to perform some tweaks to accommodate the characteristics of the sensor.

1.1.3 Achievements

The next chapter provides a brief overview about the theoretical concepts behind the work in this report. Then, the remaining chapters describe the work performed during the PhD period, which can be summarized with the key achievements listed below:

- Development of a novel point-wise processing DL method for semantic segmentation of raw radar point clouds. It is shown the importance of adapting the algorithm to the sensor properties to achieve an efficient processing.
- Assessment of the usage of radar features in point-wise processing DL methods for the detection of pedestrians. It is proved the major role played by the radar Doppler feature in various processing schemes.
- Development of a novel point-wise processing DL method for multi-class object detection from raw radar point clouds. It is proved the ability of DL to produce accurate box-predictions for different objects using radar point clouds.
- Development of an advanced defensive strategy to counteract the effect of adversarial examples. It is shown how it is possible to detect malicious samples using statistics from the training-set.

Chapter 2

Theoretical Background

The progressively growing interest in autonomous vehicle applications has dramatically increased the research in both radar and ML/DL. Though radar has always been extensively used in the automotive industry – mostly due to its low costs –, it is only in the recent years that researchers have started to use it for environment perception tasks. At the same time, the last decades have witnessed the explosion of DL algorithms which have shown superior performance in various perception problems, such as classification or object detection, thus enabling a broad variety of applications. This chapter has the goal to provide the reader with a brief overview about the theoretical concepts which are behind the scientific contributions presented in the remainder of the document.

2.1 Radar

RAdio Detection And Ranging, abbreviated radar, is a low-cost sensor which combines reliability under various weather conditions (e.g. rain/fog) with the ability to perform instantaneous velocity measurements. These aspects make it particularly well-suited for automotive applications which require sensing of the environment.

Radar perceives the environment by radiating electromagnetic (EM) waves which bounce off targets and are collected back by the sensor. Based on the sensor technology, the transmitted (Tx) signal can have different patterns. The resulting received (Rx) signal, instead, often maintains similar characteristics to the Tx one, but it is altered by the presence of targets. Consequently, by processing the received signal, it is possible to recover information about the surrounding environment. Historically, radar has commonly been used to determine the range distance between target and sensor. Consider, for instance, a pulse-radar. This sensor operates by issuing a signal in a defined time window. Then, it switched to an idle state, awaiting for the reflected signal. Since the EM wave needs to physically move (at light speed) from the sensor to target and back, it is possible to determine the distance between sensor and target by measuring the time delay between Tx and Rx signal – a.k.a. **round-trip delay**. Another information that can be recovered from the Rx signal is the target radial velocity[1]. Indeed, if the signal is reflected by a moving target, it experiences a perturbation – generally in frequency – which is proportional to the relative radial velocity between target and sensor. This instantaneous velocity mea-

[1]Not every radar technology can measure this property.

sure is very interesting for various applications, as it enables the distinction between moving and stationary targets without requiring them to be observed over time.

Despite, during the course of the years, scientists have developed several sensor technologies, this report focuses on Frequency-Modulated Continuous-Wave (FMCW) radar, as it is the technology adopted in the research proposed herein, as well as the most widespread in the automotive industry for environment perception tasks.

2.1.1 FMCW Radar: Sensor and Signal

FMCW radars have the advantage of simultaneously measuring target range and relative velocity with high accuracy and resolution. Contrarily to pulse-radars, sensors of the Continuous-Wave (CW) family radiate power continuously, instead of switching between transmission and idle state. This aspect results in safer operations, as the total transmitted power is not concentrated in a small time window, thus reducing the peak of radiated power. However, in order to make measurements technically possible, CW sensors require changes in frequency (or phase) to provide the necessary timing mark.

The FMCW technology adopts changes in frequency: the sensor radiates a signal using different frequencies over time. One of the most common signal pattern in automotive is the saw-tooth modulation, which consists in the transmission of a signal (a.k.a. **chirp**) with a frequency ramp. Figure 2.1 shows an example for this configuration. Note how the Tx signal (black) starts at a specific frequency. Then, it increases over time, until it reaches its maximum value – defined by the bandwidth parameter B. The total amount of time required to transmit the signal, T_{chirp}, is called **chirp time**. Afterwards, the signal switches back to the initial frequency, marking the start of the transmission of the next chirp. When the signal hits a target and comes back at the sensor, it experiences a time delay Δt proportional to the traveled path. This delay produces an instantaneous frequency mismatch Δf (also called **beat frequency**) between Tx (black) and Rx (gray) signal that can be

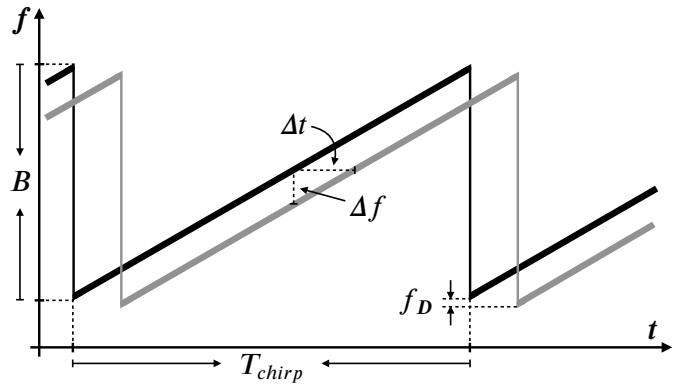


Figure 2.1: Saw-tooth modulated FMCW radar signal. The Tx signal (black) consists in a ramp sweeping frequencies in the bandwidth B. The Rx signal (gray) experiences a frequency shift Δf proportional to the round-trip time delay Δt. The Doppler effect produces a frequency shift f_D corrupting the range measurement.

measured to determine target ranging information.

Formally, it is possible to recover the range distance from time measurements as

$$R = \frac{c \cdot \Delta t}{2} \ , \tag{2.1}$$

where c represents the velocity of light in the propagation medium and Δt the round-trip time delay. However, in a FMCW radar, Δt is not directly observed, as the beat frequency Δf is measured. Let now k be the slope of the ramp in fig. 2.1, B the bandwidth of the signal (or **frequency sweep**) and T_{chirp} the chirp time. It is possible to write the round-trip time delay as

$$\Delta t = \frac{\Delta f}{k} \quad \text{where} \quad k = \frac{B}{T_{chirp}} \ , \tag{2.2}$$

which allows eq. (2.1) to be rewritten in terms of the frequency measurement, i.e.

$$R = \frac{c \cdot \Delta f}{2 \cdot k} \ . \tag{2.3}$$

In order to achieve a correct range measurement, FMCW radar needs the Rx signal to return before the next chirp is transmitted. Therefore, the maximum range that can be measured without ambiguity depends on the chirp time as

$$R_{max} = \frac{c \cdot T_{chirp}}{2} \ . \tag{2.4}$$

For the range resolution, instead, the slope k represents the key parameter as, given a fixed time delay, steeper slopes result in higher frequency shifts. Consequently, to increase the resolution while maintaining the range limits, it is necessary to increase the bandwidth B of the signal.

The derivation made so far holds true for stationary targets. In case the reflecting target, instead, has a not null radial velocity, an additional phenomenon must be considered. Indeed, the Rx signal experience a frequency shift f_D – a.k.a **Doppler frequency shift** – which translates the whole signal down or up in frequency, depending on the object's movement – i.e. away or towards the sensor. Although this effect produces an alteration in the range measurement, generally, the Doppler shift is so small compared to the beat frequency that can be safely neglected in eq. (2.3). Figure 2.1 contains a graphical example of the Doppler frequency shift.

In order to recover velocity information, FMCW radar uses consecutive chirps to exploit the fact that a moving object changes its position over time. In automotive sensors, the chirp time is generally few milliseconds or lower [46], therefore, the target movement between consecutive chirps will be very limited – at 100 km/h, it travels less than 3 cm, assuming $T_{chirp} = 1\ ms$. As a result, the Doppler frequency shift will be so smaller than the beat frequency that can be considered as a phase shift. For this reason, once calculated the range distance, it is possible to obtain the Doppler frequency shift, f_D, by analyzing the residual oscillation in the signal[2]. Then, the target relative radial velocity can be calculated as

$$v_D = -\frac{f_D \cdot \lambda}{2} \ , \tag{2.5}$$

[2]More details in section 2.1.2.

where λ is the signal wavelength. The resolution of the measurement depends on the number of consecutive chirps observed for its computation, n, as

$$v_{Dres} = \frac{\lambda}{2 \cdot T_f} = \frac{\lambda}{2 \cdot n \cdot T_{chirp}} \ , \tag{2.6}$$

where $T_f = n \cdot T_{chirp}$ is the so-called **frame time**. Note how the higher n, the better the resolution but the longer the signal is observed before obtaining the target velocity – thus affecting the instantaneity of the process. Finally, since the velocity is the result of a phase measurement, to avoid ambiguity it is necessary that the phase difference between adjacent chirps is lesser than π. Therefore, it is possible to define the maximum unambiguous velocity as

$$v_{Dmax} = \frac{\lambda}{4 \cdot T_{chirp}} \ , \tag{2.7}$$

which happens when the Doppler frequency shift produced matches half the chirp time frequency – i.e. $f_D = \frac{1}{2 \cdot T_{chirp}}$. Note how, given a fixed velocity, the lower the chirp time, the smaller the target movement between adjacent chirps and the lower the phase shift. Therefore, fast chirp signal requires a higher velocity to produce a phase shift greater than π. Another interpretation considers that the chirp time has influence on the phase sampling frequency: the lower it is, the more samples are produced.

In order to conclusively locate the target in the scene, its angular displacement w.r.t. the sensor has to be determined. As any other radar technology, FMCW radar solves this task by measuring the angle-of-arrival (AoA) of the Rx signal. However, to accomplish this, the sensor must be composed of multiple antenna elements.

Figure 2.2 describes the framework for AoA measurements, where the sensor is represented by the three antennas at the bottom. The EM wave reflected by the

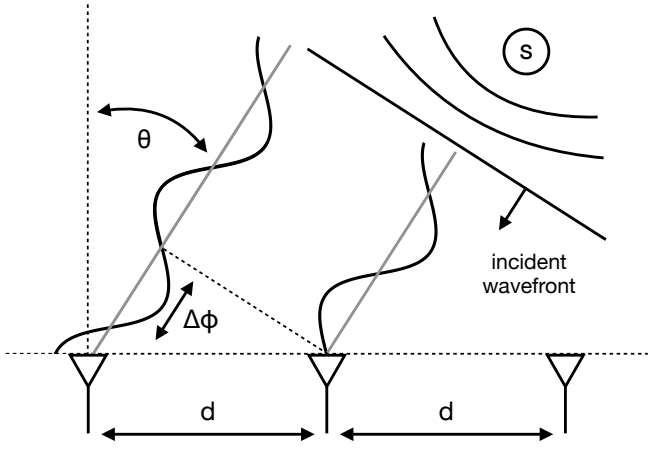


Figure 2.2: AoA framework. The target S reflects the Tx EM wave. When it gets back to the sensor (bottom), a flat wavefront can be assumed. Each antenna element receives the signal with a different phase-shift $\Delta\phi$, proportional to θ – i.e. the AoA.

target S radiates with circular fronts. However, as the signal travels in the space, it achieves the **far field conditions**[3] and the wavefront can be assumed flat [21]. When the signal approaches the radar, it is sensed by each antenna element with a time delay proportional to θ, the AoA. In the example depicted in fig. 2.2, the reflected wave hits the right-most antenna element first and then gets to the ones on the left– as it must travel a longer path. This additional time delay affects the range measurement, however, similarly to the Doppler effect, the resulting frequency shift is so small that can be considered as a phase shift. Formally, given the distance d between adjacent antennas, it is possible to express the phase shift as

$$\Delta\phi = \frac{2\pi d \sin\theta}{\lambda} \ . \tag{2.8}$$

Therefore, once measured $\Delta\phi$, it is possible to invert eq. (2.8) to obtain θ. In practice, since radar are generally made-up of several antennas, advanced algorithms are used to extrapolate the AoA[4]. Angle resolution and field-of-view (FoV), instead, are defined during the design phase as they depend on the geometry of the elements.

2.1.2 Processing Chain

FMCW radars have the advantage of performing most of the processing in the digital domain. Therefore, the required hardware is very minimal. Figure 2.3 contains the analog processing scheme[5]. A ramp signal steers the Voltage-Controlled Oscillator (VCO) that generates the Tx signal. The Rx signal is then mixed with the Tx one and fed to a low-pass filter (LPF). As the mixing operation generates a signal with two frequency components (sum and difference of the input signals frequencies), the LPF is necessary to filter out the high frequency component. The output of this down-mixing operation is a signal which oscillates at the beat frequency Δf. Finally, the resulting signal is discretized by an Analog-to-Digital Converter (ADC) to enable digital processing. The top part of fig. 2.4 illustrates an example of the whole process. The top graph plots the frequency of Tx (black) and Rx (gray) signals over time. The center graph reports the signal resulting from the frequency-difference between Tx and Rx signal. Note how it oscillates at a constant frequency, i.e. the beat frequency Δf. Finally, the bottom graph plots the amplitude of the down-mixed signal and shows the samples collected by the ADC.

The discretized signal contains a collection of samples in the time domain. However, the ranging information of the target are encoded in the frequency. To analyze its spectrum, the signal can be processed with the Discrete Fourier Transform (DFT), which is efficiently performed in the digital domain by the Fast Fourier Transform (FFT). This operation produces a digital signal which contains the energy distribution over the frequency domain. Therefore, it is possible to measure the target range by using the frequency associated with the sample which has the highest energy. In practice, as shown in the bottom part of fig. 2.4, the samples from every chirp in a frame are collected and arranged in a matrix. Then, a DFT across its columns extracts the beat frequency and the ranging information is computed by means of eq. (2.3).

[3] In automotive applications far field conditions are achieved after few centimeters.

[4] More details in section 2.1.2.

[5] Detailed aspects of the hardware are not reported as they are out-of-the-scope of this report.

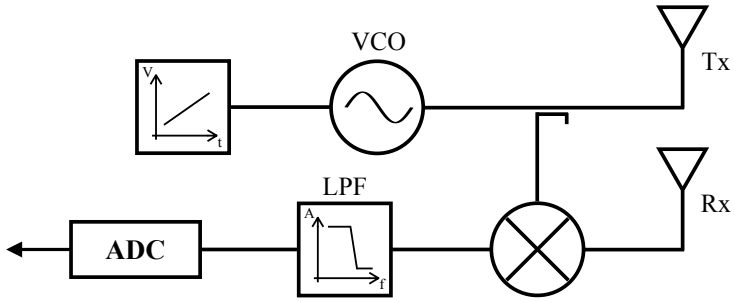


Figure 2.3: FMCW radar analog processing. A voltage ramp steers a VCO. The Rx signal is down-mixed with the Tx one to recover the beat frequency. After filtering out the high frequency component, the signal is converted by a ADC.

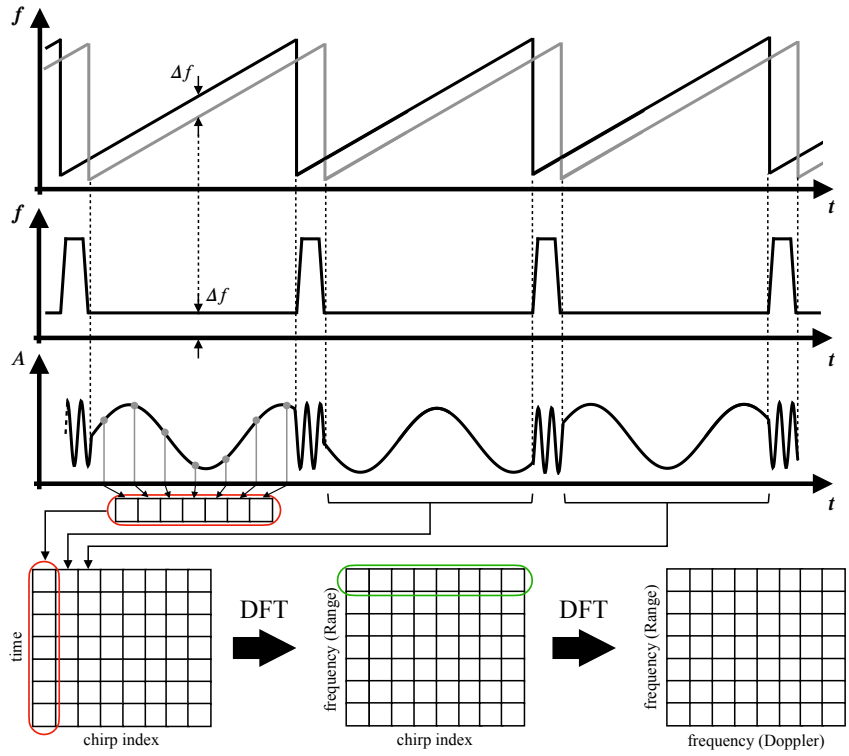
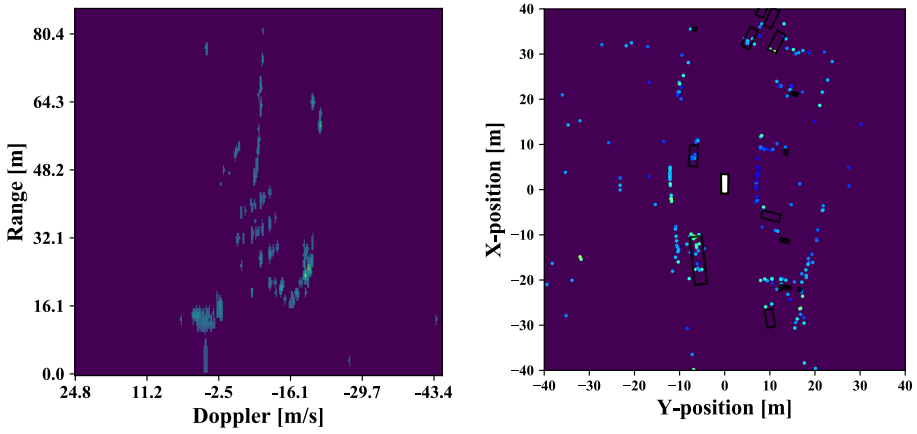


Figure 2.4: FMCW radar processing example. Tx (black) and Rx (gray) signals are down-mixed together generating a signal which oscillates at the beat frequency. This signal is discretized and arranged in a matrix where each column contains samples from the same chirp period. DFT across the columns of the matrix (red marked cells) extracts ranging information. A second DFT across the rows of the matrix which results from the first DFT (green marked cells) produces velocity information. The outcome of this operation is the RD-map.

In the previous section, it has been shown that the relative motion between sensor and target produces a phase shift. This, in turn, causes a residual fluctuation in the range spectrum across consecutive chirps, embedding information about the target radial velocity. Therefore, it is possible to use a DFT to recover the frequency-content and determine the Doppler shift. In particular, as shown in the bottom part of fig. 2.4, the DFT is performed across the rows of the data matrix in the center – i.e. produced by the first DFT. Finally, eq. (2.5) can be leveraged to obtain the relative radial velocity. In literature, range (column-wise) and Doppler (row-wise) DFTs are also known as **fast-time** and **slow-time** DFT, as they are performed, respectively, across samples of the same chirp and consecutive chirps. At this point, it is generated the so-called Range-Doppler map (RD-map), containing energy distribution across range and velocity plane. Figure 2.5A showcases an example of an RD-map.

As mentioned in the previous section, FMCW radars require several antenna elements to enable angle detection. Therefore, the processing described so far must be repeated for each sensing element. The samples of the same range-Doppler cell from different antennas constitute a new signal containing angular information, also known as beam-vector (BV). As the spatial displacement of the antennas results in a phase delay proportional to the AoA, it is possible to recover the angular information by conducting a spectral analysis on every BV. Though FFT represents a valid candidate, more sophisticated techniques are usually involved to solve this task. In automotive radars, commonly used methods are MUltiple SIgnal Characterization (MUSIC) [10], Estimation of Signal Parameters via Rotational Invariance Techniques (ESPRIT) [14] and their derivations [73, 36, 113].

In practice, the large number of cells in the RD-map makes unfeasible to process every BV with such demanding algorithms. However, as can be noted from fig. 2.5A, a RD-map generally contains many empty cells. Therefore, a common technique consists in filtering the input so that only BVs containing a signal will be processed with the AoA algorithm. In the radar domain, a widely adopted algorithm for discriminating relevant from noisy cells is the Constant False-Alarm Rate (CFAR)



(A) RD-map. (B) Point cloud.

Figure 2.5: Radar data. The same scene represented by RD-map (front-right sensor) and point cloud. Best viewed in colors.

Figure 2.6: Overview of FMCW radar digital processing chain. The discretized signal is processed with two dimensional DFT (fast- and slow-time) to produce RD-maps. The CFAR algorithm detects target signals from noisy ones. Finally, an AoA algorithm extracts angular information to generate a point cloud of detections.

[11]. It is an adaptive method that computes detections by comparing the energy of the cell-under-test (CUT) with the energy of the neighboring cells. More precisely, it compares the energy of the CUT with a threshold computed as the mean of the energy of the neighboring cells, scaled by a constant. This latter is proportional to the required false-alarm probability, thus enabling the user to set the strictness of the detection process. Note how the threshold dynamically changes for each cell, thus selecting local peaks that rise from the noise floor. Finally, every cell that has an energy higher than the threshold is considered a detection and the corresponding BV fed to the AoA algorithm that resolves the angle.

Figure 2.6 contains the whole radar processing chain. The outputs of this process are radar detections, i.e. a list of points determined by the measured range-Doppler-angle coordinates plus associated energy – a.k.a. radar-cross section (RCS). Since radars are often used in multi-sensor systems, it is convenient to map the polar-domain range-angle information to the Cartesian domain. In this way, indeed, by selecting a global reference system, it is possible to merge the measurements from different radars in a single data-list. As a result, the environment surrounding the sensors can be represented by means of a list of points: each point determines a physical target in the scene, located at the corresponding XY position. Furthermore, by means of Doppler and RCS measurements, it is possible to provide information about other properties of the targets like speed and exposure/material. This data-format is commonly referred to as **point cloud**. Figure 2.5B showcases an example of radar point clouds.

2.2 Deep Learning

Deep Learning (DL) is a branch of Machine Learning (ML). ML enables the solution of tasks which are too complex to be solved with programs of fixed rules designed by humans. However, ML algorithms are not powerful enough to effectively perform very complicated Artificial Intelligence (AI) tasks that involve high-dimensional data – e.g. speech recognition or object detection. DL has been designed to overcome the limitation of ML with high-dimensional data and the associated computational costs [49]. Yet, the basic concepts of ML applies to DL as well.

Essentially, ML algorithms are programs that can learn from data. According to Mitchell [19], "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks T, as measured by P, improves with experience E". In this definition, T is the goal the system aims to achieve, P a metric that quantifies how well the task is performed and E the process that enables the system to learn. For instance, a program learns to preform a classification task (T) if the classification accuracy (P) increases as it experiences the input-label pairs in the dataset (E).

Based on the type of experience steering the learning process, ML algorithms can be divided in **unsupervised** or **supervised**. The former learn by collecting useful properties from a dataset of many samples. For instance, clustering algorithms [5, 16] learn to group samples together based on their similarity in feature space. The latter, instead, experience a dataset composed of input-label pairs, such that the algorithm can learn the task by comparing its output with the corresponding target label. Random forest [20] is an example of supervised method, learning to threshold the input features to predict the proper output. In a nutshell, given a sample $\boldsymbol{x}$ of the dataset $\boldsymbol{\mathcal{X}}$, unsupervised methods attempt to learn the data distribution $p(\boldsymbol{x})$ by capturing interesting properties of the dataset. On the other hand, supervised methods are provided with the desired output y, therefore, they attempt to estimate the probability $p(y|\boldsymbol{x})$ by learning from the input-label association. The fact that the system has access to the target label y can be seen as a teacher instructing it on the task, hence the term supervised.

The next sections are dedicated to the key concepts of DL, describing neural network (NN) models and the algorithms used for learning.

2.2.1 Neural Network

Neural networks, also known as multi-layer perceptrons (MLP), are the building block of DL techniques. Ultimately, a NN aims at approximating a function f^* such that, when queried with an input sample $\boldsymbol{x}$, it produces an output $y = f(\boldsymbol{x};\boldsymbol{\theta})$ as close as possible to the target function output $f^*(\boldsymbol{x})$. The parameters $\boldsymbol{\theta}$ are learned to shape f as the most accurate approximation of the target function f^*.

NNs are called *neural* because they are inspired by neuroscience to resemble the brain: neuron-like units (or nodes) are connected to each other by means of synaptic-like weights. However, it is the term *network* which tells more about this algorithm. It derives from the structure of NNs, which is a composition of several functions. Indeed, the final NN function, $f(\boldsymbol{x};\boldsymbol{\theta})$, is generally constructed by chaining together several function:

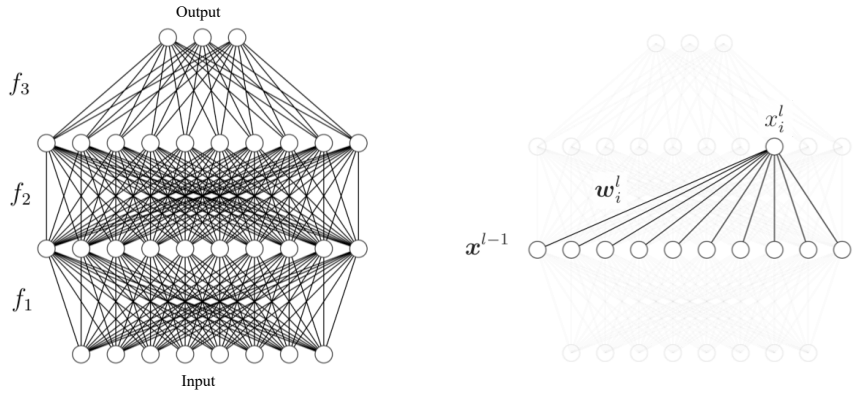
$$f(\boldsymbol{x};\boldsymbol{\theta}) = f_3(f_2(f_1(\boldsymbol{x};\boldsymbol{\theta^1});\boldsymbol{\theta^2});\boldsymbol{\theta^3}) \ . \tag{2.9}$$

In this case, f_1 is called the **input layer**, f_3 the **output layer** and f_2 the **hidden layer**. The more functions are chained, the **deeper** is the model. Figure 2.7A shows the graph related to the function in eq. (2.9) – implemented as a fully-connected (FC) network with three output nodes.

By inspecting a single unit in an FC model, it is possible to gather more information about the basic operation of NNs. FC nets are a kind of NN where each node in layer l receives as input the output of every node from the previous layer $l-1$. Let $\boldsymbol{x^{l-1}}$ be the input at layer l, it is possible to write the output (or **activation**) of the i-th node at layer l as

$$x_i^l = h(\boldsymbol{x^{l-1}};\boldsymbol{\theta_i^l}) = \sigma(\boldsymbol{w_i^l}\cdot\boldsymbol{x^{l-1}} + b_i^l) = \sigma(\sum_j w_{ij}^l \cdot x_j^{l-1} + b_i^l) \ , \tag{2.10}$$

where $\sigma(\cdot)$ is a non-linear function, also called **activation function**, and the parameters $\boldsymbol{\theta_i^l}$ are the **bias** b_i^l and the weights vector (or **kernel**) $\boldsymbol{w_i^l}$ – containing the weights that multiply every feature of the input $\boldsymbol{x^{l-1}}$. Figure 2.7B graphically depicts the operation in eq. (2.10): the edges connecting the previous layer to the respective node account for the weights $\boldsymbol{w_i^l}$, whilst the node is responsible for adding



(A) FC network. (B) Single unit operation.

Figure 2.7: Graphical illustration of a fully-connected NN.

the bias b_i^l and applying the non-linearity. Therefore, the core operation of NNs consists in a simple weighted combination of the previous layer outputs, filtered by a non-linearity. However, what defines this algorithm is its ability to learn the values of the weights that produce the desired final outcome.

Another convenient mathematical representation of NN involves matrix notation. By considering all the units in layer l, it is possible to express the whole layer output as

$$\boldsymbol{x}^l = h(\boldsymbol{x}^{\boldsymbol{l-1}}; \boldsymbol{\theta}^{\boldsymbol{l}}) = \sigma({\boldsymbol{W}^{\boldsymbol{l}}}^T \boldsymbol{x}^{\boldsymbol{l-1}} + \boldsymbol{b}^{\boldsymbol{l}}) \ , \tag{2.11}$$

where $\sigma(\cdot)$ operates element-wise, each column of $\boldsymbol{W}^{\boldsymbol{l}}$ contains the weights related to a single node and each row of $\boldsymbol{b}^{\boldsymbol{l}}$ the bias. Finally, by chaining eq. (2.11) from the input layer ($l = 1$) to the output layer ($l = L$), it is possible to compute the overall function f that represents the network.

During the course of the years, several models have spawned from FC networks, using eq. (2.10) as its core operation. Hereafter, it is provided a description of two very popular models.

Convolutional Neural Network Convolutional neural networks (CNNs) [13], are models specialized for processing structured inputs such as time-series data or images. The intuition behind CNNs relies in the different interaction between input-output nodes: each unit no longer receives as input every activation in the previous layer but only a localized subset of them. Specifically, convolution layers use kernels of predefined shape to span the input, thus resulting in its weights being shared across different regions of the input. This property is very useful as it allows them to scale well to data with large and variable size. Over the last decades, CNNs have proven tremendously effective on images and have been used in a broad variety of applications, notably image classification.

Recurrent Neural Network All the neural networks described so far belongs to the **feed-forward** family – i.e. they can be associated to an acyclic graph where the information flows in one direction: from the input to the output. Recurrent

neural networks (RNNs) [9], are models including feedback connections that inject information extracted from previous inputs into the processing of the current sample. This characteristic makes RNNs well-suited for processing sequential data, such as videos, containing temporal information. The core concept consists in the presence of a *state* $\boldsymbol{s}$ in the network which holds information describing the previous samples seen by the model. Then, the network output can be written as a function of both the current input $\boldsymbol{x^t}$ and the state from the previous sample $\boldsymbol{s^{t-1}}$, as shown in eq. (2.12a). At the same time, the network needs to update its state in order to incorporate information extracted from the current input, as shown in eq. (2.12b),

$$y^t = f(\boldsymbol{x^t}, \boldsymbol{s^{t-1}}; \boldsymbol{\theta^f}) \ , \tag{2.12a}$$

$$\boldsymbol{s^t} = g(\boldsymbol{x^t}, \boldsymbol{s^{t-1}}; \boldsymbol{\theta^g}) \ . \tag{2.12b}$$

During the years, several variants of RNNs have been developed to improve the ability to learn from temporal information [18]. Nowadays, RNNs are very widespread, finding application in a variety of ordinary tasks like language translation.

2.2.2 Stochastic Gradient Descent and Back-propagation

The previous section stated that the parameters of a NN can be learned. In a DL framework, the learning procedure is called **training**, as in this process the model is instructed on how to perform its task. Specifically, at the output layer is instantiated an **objective function**, commonly referred to as **loss** or **cost function**, which tells how bad the current parameters configuration is at performing the task. For example, in a supervised learning framework, the objective function is typically built by comparing the network output to a training sample with the associated label. The lower the objective function is, the closer output and target are. During the training process, it is addressed an optimization problem that allows the network to adapt its parameters in order to minimize the cost function. Hence, steering the output to be as close as possible to the ground-truth (GT) label.

Among the various optimization techniques, DL usually adopts gradient-based solutions. The optimization theory tells us that the local gradient is very informative in a minimization/maximization problem: it indicates the slope of the function at a specific point. Therefore, one could minimize a function by simply following the opposite direction of the gradient. Indeed, considering a scalar problem where f' indicates the derivative of f w.r.t. to x, for many ϵ small enough holds true that

$$f\big(x - \epsilon \ \text{sign}(f'(x)\big) < f(x) \ . \tag{2.13}$$

Let now $\mathcal{L}(\boldsymbol{\theta})$ be the objective function to minimize and $\boldsymbol{\theta}$ the network parameters. Bearing in mind that the gradient is a vector of partial derivatives, it is possible to rewrite an expression similar to eq. (2.13) in vector notation, i.e.

$$\mathcal{L}\big(\boldsymbol{\theta} - \epsilon \ \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})\big) < \mathcal{L}(\boldsymbol{\theta}) \ . \tag{2.14}$$

The training process attempts to find the configuration $\boldsymbol{\theta}^*$ of the network parameters that minimize the objective function, namely it tries to solve

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) \ . \tag{2.15}$$

Exploiting the property in eq. (2.14), it is possible to solve the optimization problem in eq. (2.15) by iteratively applying eq. (2.16) to follow the gradient direction downhill

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \epsilon \, \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}^{(t)}) \; . \tag{2.16}$$

The update rule in eq. (2.16) is known as **gradient descent**, or **method of steepest descent**, and ϵ is commonly referred to as **learning rate**. Note how this method requires the gradient to be computed at every update step.

In a DL framework, the cost function is usually built-up as sum of per-sample cost functions over the training dataset $\boldsymbol{\mathcal{X}}$. Equation (2.17) provides an example of cost function for a supervised learning model, where $L(\cdot)$ is a per-sample loss

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},y)\sim\boldsymbol{\mathcal{X}}}[L(\boldsymbol{x}, y, \boldsymbol{\theta})] = \frac{1}{N} \sum_{i=1}^{N} L(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}) \; . \tag{2.17}$$

By differentiating the loss function in eq. (2.17), it is possible to obtain the signal in eq. (2.18), which requires computation of the gradient for each sample in the dataset

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},y)\sim\boldsymbol{\mathcal{X}}}[\nabla_{\boldsymbol{\theta}} L(\boldsymbol{x}, y, \boldsymbol{\theta})] = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}) \; . \tag{2.18}$$

In practice, calculation of the gradient as in eq. (2.18) is hardly ever possible on DL datasets, as computational costs grow linearly with dataset dimension N – i.e. as $O(N)$. Therefore, the standard gradient descent algorithm – with update rule in eq. (2.16) – would take excessively long to perform a single training step.

To overcome this limitation, DL algorithm generally relies on a variant of gradient descent: stochastic gradient descent (SGD). SGD exploits the fact that the gradient signal is an expectation, as shown by eq. (2.18). Consequently, it can be approximated using a subset of the training samples. In more details, this algorithm requires to query the network with a small batch of inputs. Then, it performs the update step in eq. (2.16) using as gradient an estimate calculated as

$$\widehat{\boldsymbol{g}} = \mathbb{E}_{(\boldsymbol{x},y)\sim\boldsymbol{\mathcal{B}}}[\nabla_{\boldsymbol{\theta}} L(\boldsymbol{x}, y, \boldsymbol{\theta})] = \frac{1}{n} \sum_{i=1}^{n} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}) \; , \tag{2.19}$$

where $\boldsymbol{\mathcal{B}} = \{(\boldsymbol{x}^{(1)}, y^{(1)}), \ldots, (\boldsymbol{x}^{(n)}, y^{(n)})\}$ is the current batch of n input-label pairs uniformly sampled from the training set. Since n is often chosen such that $n \ll N$, $\boldsymbol{\mathcal{B}}$ is commonly known as **mini-batch**. Though the gradient estimated in eq. (2.19) is more noisy than the gradient in eq. (2.18) – it is computed on a limited amount of samples –, it makes possible to scale the problem to be $O(1)$ w.r.t. the dataset dimension. Nowadays, SGD is extensively used in the DL community, enabling networks to learn from huge datasets, composed of up to millions of samples.

SGD provides the instructions to analytically compute the gradient. However, DL networks are very deep, therefore, numerical calculation of such quantity can be exhaustive. The **back-propagation** algorithm provides the numerical rules for gradient computation throughout the network. It exploits the chain rule to decompose gradient computation of a composite functions into intermediate derivatives. Consider, without loss of generality, the scalar version of the network in eq. (2.9), where the NN function f is composed of three chained functions f_1, f_2 and f_3, with

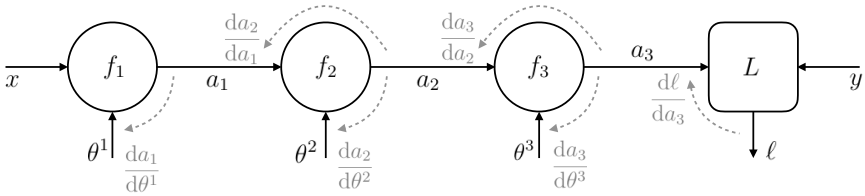


Figure 2.8: Computational graph of a network during training. The dashed gray arrows show the back-propagation path with corresponding derivatives.

scalar parameters θ^1, θ^2 and θ^3 and output activations a_1, a_2 and a_3, respectively. Figure 2.8 shows the computational graph of the network during training. Given the loss $\ell = L(f(x), y) = L(a_3, y)$, it is possible to compute the update term for the parameter θ^1 of the input layer as

$$\frac{\mathrm{d}\ell}{\mathrm{d}\theta^1} = \frac{\mathrm{d}\ell}{\mathrm{d}a_3}\frac{\mathrm{d}a_3}{\mathrm{d}\theta^1} \tag{2.20a}$$

$$= \frac{\mathrm{d}\ell}{\mathrm{d}a_3}\frac{\mathrm{d}a_3}{\mathrm{d}a_2}\frac{\mathrm{d}a_2}{\mathrm{d}\theta^1} \tag{2.20b}$$

$$= \frac{\mathrm{d}\ell}{\mathrm{d}a_3}\frac{\mathrm{d}a_3}{\mathrm{d}a_2}\frac{\mathrm{d}a_2}{\mathrm{d}a_1}\frac{\mathrm{d}a_1}{\mathrm{d}\theta^1} \ . \tag{2.20c}$$

Notice how the derivative has been broken down into several sub-derivatives. The term $\frac{\mathrm{d}\ell}{\mathrm{d}a_3}$ contains the error signal resulting from the comparison prediction-label. The term $\frac{\mathrm{d}a_1}{\mathrm{d}\theta^1}$ computes the derivative of the input function f_1 w.r.t. its parameter θ^1. Finally, the terms $\frac{\mathrm{d}a_3}{\mathrm{d}a_2}$ and $\frac{\mathrm{d}a_2}{\mathrm{d}a_1}$ are the derivative of the layer functions f_3 and f_2, respectively, w.r.t. their inputs. These two are responsible for propagating the error signal from the output layer, back into the network, towards the input layer.

The back-propagation algorithm allows the error signal to flow from output to input while it keeps track of the intermediate derivatives. In this way, it is not necessary to compute all the derivatives from scratch, thus reducing the computational complexity. Notice, indeed, how it is possible to reuse the first two terms of eq. (2.20b) to compute the update term for the parameter θ^2. The back-propagation paradigm is crucial as it enables to train very deep models with a reasonable amount of resources. Together with SGD, it is the core tool that allows DL to successfully perform advanced, complicated tasks and generalize to a broad variety of domains.

2.2.3 Tasks

Over the last years, DL has been gradually more involved in various applications, using different type of sensors [124, 115, 57, 51, 112, 89]. Its ability to generalize to unseen data with minimal problem modeling requirements, makes this family of algorithms particularly well suited to solve complex tasks. In this section, the most common tasks which can be tackled with DL solutions, especially NNs, are introduced.

Classification It is the most common task in the DL area. In this task, the model has to associate the input with one of d classes or categories. For instance, a

classification task might require the algorithm to label an input image as containing dogs or cats. Therefore, the model is asked to estimate a function $f : \mathbb{R}^m \rightarrow [0, 1]^d$ such that, when queried with a m-dimensional input, it outputs a score for every one of d classes. A common interpretation of the output score is as probability of the input to belong to the specific class. During the years, researchers have collected several datasets targeting the classification task [45, 26, 29]. As an example, CIFAR10 [26] provides image-label pairs of common objects like planes or ships.

Regression This kind of task challenges the algorithm to produce a continuous output: the model is asked to predict the exact value of a property of the input, such as the length or width of a vehicle. It can be formalized as the estimation of a function $f : \mathbb{R}^m \rightarrow \mathbb{R}$. Compared with classification, regression tasks are more difficult, as the network has to produce a precise value, rather than selecting within a pool of possible outcomes. Nowadays, there are several datasets providing regression tasks [42, 33]. For instance, Brown et al. [33] collected a dataset containing time series data of the Dow Jones Industrial Index, enabling future stock price predictions.

Semantic Segmentation It extends the classification task to the whole input domain. Specifically, it asks to classify every part of the input according to d semantic classes. Segmentation tasks are very common with point cloud data, where the task consists in the classification of every point. Formally, given an input with N points and m features, a model that performs a semantic segmentation task estimates a function $f : \mathbb{R}^{N \times m} \rightarrow [0, 1]^{N \times d}$ such that every point gets a classification score for every output class. Another task from this family is **instance segmentation**, which requires not only to classify every point as a class, but also to specify the different physical instances they belong to.

Object Detection It represents the ultimate solution to environment perception problems. It combines classification and regression tasks to locate, shape and categorize the objects of interest present in the input. In this task, the model is asked to estimate a function $f : \mathbb{R}^{N \times m} \rightarrow \mathbb{R}^{M \times k} \times [0, 1]^{M \times d}$, where M is the number of objects, k the parameters to regress and d the classes of objects of interest. For instance, the task can require to predict the position of vehicles and pedestrians in the input, along with their dimension – i.e. length, width and height. N can be both the number of points in a point cloud or the number of pixel in an image, while m represents the features associated to each point/pixel. Object detection is a very complex but informative task which can enable a variety of applications. For this reason, the research community is very active in this area, with several dataset available. As an example, NuScene [105] is an automotive dataset providing camera, lidar and radar data collected in urban scenes along with annotated objects.

2.3 Deep Learning on Radar

The limited availability of public datasets has significantly slowed down the adoption of DL in the radar domain. However, the recent years have witnessed an intensification of the research in this topic. Indeed, concerning the automotive industry, the unique properties of radar can play an important role in the arm-race to autonomous

driving solutions. Fueled by these ambitions, researchers have successfully engaged in the development of various DL techniques to effectively consume radar data. In this section, the main DL approaches for processing radar data are introduced, providing details about the state-of-the-art methods.

2.3.1 Classical Approaches

Historically, radar utilization in environment perception systems involved the generation of occupancy grid-maps (OGMs) [30, 35]. OGMs [12] integrate measurements over time to create a map representing the surrounding environment. The resulting grid contains the probability of obstacles being present at every cell location. During the years, researchers have leveraged the radar features to improve the OGM representation. For instance, amplitude grid-maps [48], which use the measured RCS values, confer to each cell fine details about the obstacles. Due to its structured nature, such data-format can be processed with conventional CNNs to perform tasks like semantic segmentation or object detection [61, 62, 99, 114]. However, though this approach results very successful on static objects, it struggles on dynamic ones, since their movement generates tails corrupting the map.

To overcome these limitations, scientists have proposed to process radar point clouds in a three-stage approach [78]. In the first stage, a clustering algorithm groups together detections with similar properties. Then, features of points belonging to the same group are leveraged to extract a cluster signature. Finally, a classification model determines the cluster class based on its signature. An example of this approach can be found in [67], where the authors used Density Based Spatial Clustering for Applications with Noise (DBSCAN) [16] at the clustering stage and a long-short term memory (LSTM) network [18] for classification.

Classical radar processing approaches use DL solutions only at the classification stage, therefore, they cannot exploit the full strength of such techniques. Yet, more recent works have proved that it is possible to consume radar data using end-to-end DL methods, thus leveraging their ability to learn from low-level data. The next sections provide an overview on state-of-the-art approaches that fully rely on DL to process radar data.

2.3.2 Point Cloud Approaches

Point cloud is a typical yet efficient data representation for radar. Indeed, contrarily to OGMs, point clouds do not directly represent void region of the space, as they are defined as a collection of point-measurements. However, the most successful DL methods do no scale-well to such unstructured data. As an example, CNNs – which are the state-of-the-art solution – owe their popularity to remarkable achievements in image processing. Therefore, new techniques have been designed to enable the adoption of DL on point clouds.

There are two main techniques for point cloud processing: grid-based and point-wise. Grid-based approaches convert the point cloud in a structured input, so that conventional methods, like CNNs, can be applied. Specifically, the point cloud is fitted into a grid and the point features are encoded into the cells. Based on the encoding process, it is possible to differentiate between 3D and multi-view methods. The former use 3D voxel-encoding to enable the deployment of 3D CNNs [88, 56].

The latter use 2D pixel-encoding to obtain images that can be processed with conventional CNNs [72, 87]. Multi-view methods repeat the encoding process several times in order to gather different snapshots of the input. Then, features from every view are fused to recover 3D information. Grid-based methods have proven dominating performance on tasks like shape recognition, semantic/part segmentation and object detection. Yet, the grid-resolution parameter results in a trade-off between computational demands (small cells) and information loss (big cells).

Point-wise processing approaches avoid the information-computation trade-off by consuming raw point clouds. However, since the input received by these methods is a list – whose entries are the point with associated features –, the network adopted are constrained to be order-invariant. Note, indeed, how it is possible to generate a different input by reshuffling the order of the list-entries. Yet, the new list represents the very same physical point cloud, therefore, the algorithm has to produce a similar outcome. Despite the order-invariance constraint hampers the design of point-wise architectures, the literature proves that they can achieve state-of-the-art performance on complex tasks like semantic/part segmentation and classification [65, 64, 74, 68]. Additionally, the processing performed by such methods is efficient, maintaining a low computational complexity without discarding input information.

A sensor which can be associated with radar is lidar (LIgth Detection And Ranging). Despite it possesses different intrinsic properties, indeed, it produces data in the form of point clouds that describe the surrounding environment. Due to the limited availability of public radar datasets and the wide popularity of lidar, both grid-based and point-wise approaches discussed so far have first found application on lidar-data. However, during the recent years, the research community has shown increasing interest in radar, thus pushing the development of DL-based solutions for radar point clouds. In particular, researchers have focused on point-wise processing approaches, mainly for twofold reasons:

- Radar is a unique sensor, intrinsically different than lidar. The point clouds produced by radar are very sparse, generally orders of magnitude sparser than lidar ones. This aspect severely affects grid-based methods, which would result in many empty grid-cells and/or resolution degradation [93].

- Radar already provides grid-data in the form of RD-maps. Therefore, rather than encoding point-information into grid-cells, a more effective processing would use RD-maps as input [89, 97]. Section 2.3.3 provides an overview on these approaches.

For these reasons, hereafter, it has been focused on point-wise processing techniques, describing the state-of-the-art in this area and their application on radar data.

PointNet PointNet [65] is a milestone among point-wise processing methods. The authors designed an order-invariant architecture by leveraging two building-blocks: shared FC layers and max-pooling. In more details, they noted that, by sharing the weights of the kernel across all points, it is possible to learn the same high level representation for every point, regardless of its position in the list. Additionally, they found out that a symmetric function can extract global information from the point cloud. Therefore, they adopted max-pooling to leverage the features of all the points. Then, the global signature can either be used for classification purposes

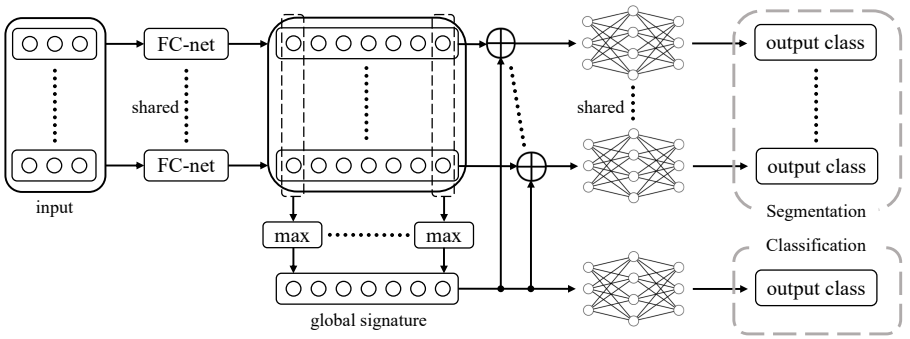


Figure 2.9: PointNet architecture. Shared FC layers ensure learning of order-invariant features. Max-pooling extracts a global descriptor from point-features. By means of a shared FC classification network it is possible to perform classification or segmentation. The $\oplus$ symbol defines the concatenation operation.

or appended to the features of each point to perform semantic/part segmentation. Figure 2.9 shows the architecture. PointNet has proven capable to consume raw point clouds, showing good performance in both classification and segmentation tasks. However, it fails to encode information of proximity between points, which represents a key aspect for the solution of more complex tasks.

PointNet++ PointNet++ [64] represents the current state-of-the-art in the field of point-wise processing. The authors noted that CNNs are so successful because of their ability to hierarchically encode local low-level structures into high-level patterns. Therefore, they designed an evolution of the PointNet architecture, by using a hierarchical encoder-decoder structure that can learn from geometric information. To maintain the ability to consume raw point clouds, they defined two building-blocks: set-abstraction (SA) and feature-propagation (FP) layers. SA layers are used in the encoder section to learn high-level representations, whilst FP layers propagate it back to the lower levels in the decoder section. In more details, SA layers perform three operations: sampling, grouping and feature extraction. The first process selects a sub-set of the input points by means of the farthest point sampling (FPS) algorithm, thus uniformly covering the input points. Then, the points which lie in the neighborhood of the selected ones are grouped together. Finally, an order-invariant feature extraction module encodes the neighborhood cluster into a single feature-vector. PointNet is used for this purpose. Figure 2.10 depicts a visual example. FP layers perform an inverse distance weighted interpolation to propagate the high level information into the original point locations. Skip connections from corresponding SA-FP pairs enrich the point signatures, boosting the information flow. PointNet++ is the most popular architecture among point-wise approaches.

Radar solutions The first works proposing the application of point-wise techniques to process radar point clouds adopt conventional methods. Schumann et al. [81] deployed a PointNet++ architecture to segment radar reflections according to six classes: pedestrian, pedestrian group, bike, truck, car and static. This work proved that point-wise architectures can generalize well to radar point clouds,

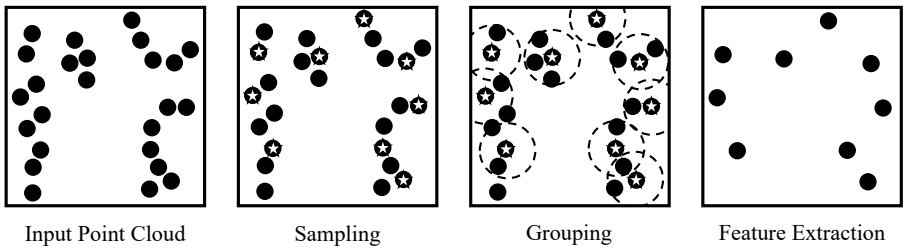


Figure 2.10: PointNet++ SA layer operations. The sampling process selects some of the input points (star-marked). Then, the points which lie nearby are grouped together, forming a local point cloud. Finally, the local point clouds are processed with PointNet to extract a signature for each group.

despite they have been designed for other domains. Other researchers enhanced the PointNet++ framework by introducing radar-related hand-crafted statistics like point density [93]. They targeted the semantic segmentation task between vehicles and guardrails. The authors proved that it is possible to improve the system performance by adapting the network to address the nature of radar data. Braun et al. [118] showed that it is possible to process radar point clouds with a different architecture. They adopted PointCNN [74] and addressed the task of detecting moving vehicle and pedestrian points. Other works, focused on solving more complicated tasks. Schumann et al. [114] addressed the whole scene understanding task. They used an OGM-based solution to gather insights about the stationary objects in the scene. Additionally, they designed a novel point-wise architecture to detect dynamic objects. The authors tampered the PointNet++ framework by introducing a memory point cloud. This allowed the network to infer the final output based on inputs seen in the past. The authors targeted the instance segmentation task and proved that point-based architectures can perform such complex tasks. Danzer et al. [92] addressed the object detection task with radar data. They used a PointNet model to detect the points that belong to a vehicle. Then, they estimated the corresponding oriented bounding-box (BB). This work proved that it is possible to predict 2D object boxes from radar point clouds. Successively, Dreher et al. [109] compared the model proposed in [92] with a grid-based approach, proving that the former detects objects more effectively on radar point clouds.

2.3.3 Range-Doppler Map Approaches

An interesting DL approach for radar consists in processing low-level data. Indeed, Range-Doppler maps are image-like, structured inputs easily consumed by neural networks. By concatenating the RD-maps generated by every antenna it is possible to achieve a full 3D radar cube. However, it is worth stressing out that radar signals live in the polar domain. Therefore, in order to process the radar cube with conventional, Cartesian-based DL solutions, customizations are required.

Major et al. [97] designed an architecture that can detect objects from the raw radar cube. Their implementation relies on CNNs to process 2D views of the input cube: the range-antenna-Doppler (RAD) cube is split into RA, RD and AD views by summing up the signal power over the excluded dimension. To address the

polar nature of the data, they resorted to the convolution implementation in [76], appending the pixel coordinates to each cell. As a result, the convolution output is conditioned on the location where the kernel is applied. The high-level features learned from the 2D views of the input are then merged together and propagated to the original location by means of a decoder 3D CNN. Finally, the latent features learned in the polar domain are interpolated bi-linearly in a Cartesian grid and processed with an 3D LSTM network to produce object box proposals. This work proved that it is possible to detect objects directly from the radar cube. Additionally, the authors concluded that it is extremely important to take into consideration the polar nature of the data.

In another work, Brodeski et al. [89] developed a DL framework to extract detection from the radar cube. The system consists in a 2-stage approach. The first stage segments the range-Doppler image – using antenna as channel axis – with a 2D CNN-based encoder-decoder architecture. As a result, every location in the RD-map is classified as object or non-object. Then, the second stage, receives as input the 3×3 crop of the RD-image, centered on the object cells, and estimates their angular content (azimuth and elevation) via a CNN. The global signature learned in the first-stage network is included at this stage to enrich the context information. The authors evaluated their system on calibration data collected in anechoic chambers, proving its ability to estimate the correct target position.

Chapter 3

RadarPCNN: an Innovative Architecture for Semantic Segmentation of Radar Point Clouds

Radar point clouds contain information about the location of targets around the sensor as well as the target themselves. In autonomous vehicle applications, common targets include, but are not limited to, vehicles, pedestrians and bikes. The semantic segmentation task allows the discrimination of points generated by different object categories, thus providing insights about the targets distribution in the scene. Furthermore, its output can be used to enrich the point cloud, enabling the solution of more complicated tasks like object detection or instance segmentation.

This chapter investigates the effectiveness of DL solutions on radar data. It begins with a brief presentation of the point cloud semantic segmentation task. Then, it introduces novel solutions, specifically designed for radar applications, that improve both efficiency and effectiveness of point-based processing schemes. Section 3.3 describes both the dataset and the settings used during the evaluation. Finally, the last section reports the results of the experiments, proving the validity of the proposed solutions.

The following own contributions are provided in this chapter:

- Development of RadarPCNN, a novel NN-based point-wise processing architecture for semantic segmentation of radar point clouds.
- Introduction of a pre-processing module, encoding radar-related information into point-descriptors.
- Development of an attention mechanism, enabling smart combination of multiple point-signatures.
- Adoption of the mean-shift (MS) algorithm [5] as sampling method in the PointNet++ framework.
- Assessment of the network operations such as grouping process and usage of radar Doppler feature.

- Analysis of the network errors through visual investigations.
- Study of the radar similarity between pedestrians and bushes.

The results in this chapter were previously published in [108] and [120].

3.1 Point Cloud Semantic Segmentation

Point clouds provide a detailed description of the space surrounding the sensor. By leveraging information about 3D structures and objects, it is possible to solve environment perception tasks, such as object detection, instance or semantic segmentation. This latter consists in the classification of each point, according to a set of classes, thus providing information about the location of the targets. It is possible to provide a formal definition of the semantic segmentation task. Let's consider an unordered collection of $N \in \mathbb{N}$ points $\boldsymbol{x^i} \in \mathbb{R}^m$, $i = 1, \ldots, N$, whose elements are the 2D/3D spatial coordinates with annexed features. The semantic segmentation task can be defined as the problem of estimating the function $f : \mathbb{R}^{N \times m} \rightarrow \mathbb{R}^{N \times d}$ such that every input point is associated with one of d semantic classes.

There exist several methods that can be used to semantically segment a point cloud. The most simple solution consists in using each point's features to infer the output class. For this purpose, it is possible to use algorithms like random forest (RF) [20] or Support Vector Machine (SVM) [15]. The former processes the points' features with a set of learned threshold to determine the class prediction. The latter performs its task by finding the hyperplane which best separates the input data-points, based on their features. Though very straightforward, however, these methods do not perform very well – as shown in section 3.4 –, due to the lack of context in the inference process. More advanced solutions make use of clustering algorithms – like DBSCAN [16] – to group together points with similar properties. Then, new features are computed from points belonging to the same cluster and used as input to either RF, SVM or NN models, which predict a class for the whole cluster [67, 78]. By propagating the cluster predicted class to every associated point, it is possible to semantically segment the whole point cloud. However, despite introducing context information in the decision process, these techniques still depend on the correctness of the clustering algorithm.

The most recent and efficient solutions involve NN-based architectures. As described in section 2.3.2, there are two main families. Grid-based methods resort to grids to exploit the strength of computer vision (CV) approaches. However, these methods can output a semantic segmentation grid rather than points. A workaround consists in assigning to each point which fall in a given cell the class prediction of the cell. Note that grid-based algorithms are strongly characterized by the cell resolution: the smaller the cells, the lower the minimum distance allowed between objects of different classes, but the higher the computational overhead – due to empty cells with no points. The second family of algorithms that can be used for semantic segmentation purposes are the point-wise processing methods. These methods consume raw point clouds, therefore, they produce as output a class for every input point.

Among the various existing point-wise processing methods, PointNet++ [64] represents the current state-of-the-art. It has been designed to replicate the operations of traditional CNNs, in particular, their ability to hierarchically abstract

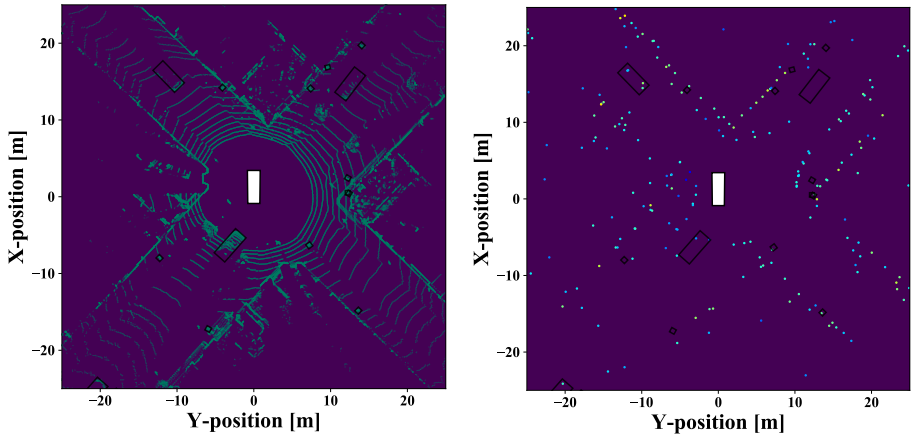
low-level structures into high-level patterns[1]. However, the original purpose of this architecture has been to process lidar point clouds, which differ substantially from radar ones. Indeed, radar has the ability to measure object-related attributes, which results in additional features like Doppler and RCS. On the other hand, lidar has higher spatial precision which results in high-density point clouds. Therefore, it is necessary to adapt the architecture before deploying it on radar data. Nevertheless, several works have demonstrated that PointNet++ can generalize well to radar point clouds [81, 93], achieving dominating semantic segmentation performance.

3.2 RadarPCNN

PointNet++ has proved to be an effective tool for consuming radar point clouds. It is characterized by its ability to use structural information. Indeed, by processing local groups of points, it enables the extraction of high-level local descriptors from low-level patterns. For instance, consider a scene containing the corner of a building, like in fig. 3.1. In order to recognize the whole object, the system must have a field-of-view (FoV) broad enough to asses both straight parts and the corner. By breaking down the input in local groups of points, it is possible to distribute the task. In this way, points locally scattered to form an L-like shape hints the presence of a corner. On the other hand, points in a straight line suggests the presence of an edge. Therefore, it is possible to encode this information in high-level signatures, providing a local descriptions of the environment.

Radar, however, provides weak structural information. Due to technology limitations, it usually generates point clouds with lower density than lidar ones. This effect is visible in fig. 3.1. It shows a point cloud recorded with lidar (left) and radar (right). Both images describe the same scene, where the ego-vehicle is turning left at a crossroad. Note how much denser is the lidar point cloud compared to the

[1]The interested reader is referred to section 2.3.2 for a detailed description of PointNet++.



(A) Lidar point cloud.

(B) Radar point cloud.

Figure 3.1: Lidar and radar point clouds describing the same scene where the ego-vehicle is turning left at a crossroad. Black rectangles represent objects of interest.

radar one, providing very detailed shape information. Back to the building's corner example, it is possible to observe that lidar accurately depicts the contours of the buildings. On the other hand, radar is not so precise: it provides a fairly good appearance of the corner in front of the ego-vehicle, but it is difficult to visualize the exact location of the other three corners. Consequently, since PointNet++ relies mostly on spatial structural information to perform its task, it is believed that, although effective, it represents a sub-optimal solution for processing radar data.

Radar has other advantages over lidar, like the ability to operate in hard weather conditions and to measure object-related attributes. This latter, in particular, can be leveraged to increase the efficiency of the processing scheme. Indeed, it is crucial that the underlying characteristics of the sensor are addressed and exploited by the network. This section introduces RadarPCNN, a novel point-wise processing architecture specifically-designed for semantic segmentation of radar point clouds. RadarPCNN uses a pre-processing module to leverage object-related features and enrich the input points signature. Moreover, it is proposed to use mean-shift (MS) [5] to improve the usage of the scarce structural information provided by radar. The PointNet++ scheme is adopted to enable consumption of raw point clouds. However, the hierarchical structure is discarded in favor of an object-oriented configuration. Finally, an attention mechanism is proposed to fuse information extracted at different levels within the network. Figure 3.2 contains the developed architecture.

3.2.1 Pre-processing Module

The most important radar characteristic that can be exploited during processing lies in its features. Radar provides two highly-informative target-related attributes: Doppler and radar-cross section (RCS). The Doppler feature contains dynamic information about the target. It generates as a frequency shift, proportional to the relative radial velocity between sensor and target. This attribute can be used to distinguish moving from stationary objects. RCS, instead, provides information about the nature and pose of the target. It can be interpreted as an energy-like attribute of the radar signal: metallic targets produce high RCS values while hybrid targets, like pedestrians, generate lower values. However, it is also dependent on the target pose, as it is proportional to the area irradiated by the radar signal. When

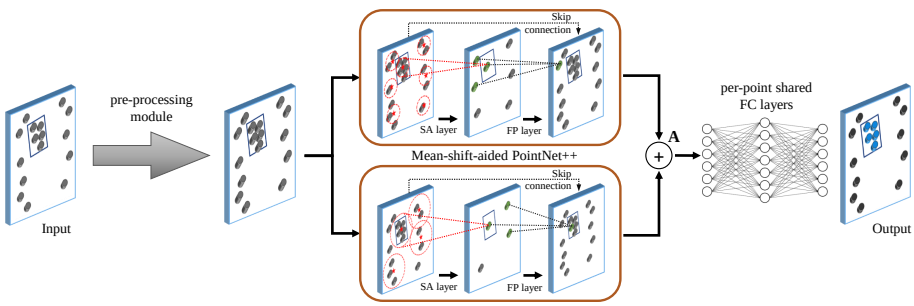


Figure 3.2: RadarPCNN architecture. The proposed pre-processing module learns features for each point. Mean shift is used to alter the sampling process of a single-stage PointNet++. An attention mechanism fuses point signatures and a shared FC-network produces the class scores. Best viewed in color.

combined together, Doppler and RCS build-up a signature for specific objects. For instance, pedestrians have a broad Doppler-spectrum and weak RCS values [31], while metallic poles share a static Doppler with fairly constant RCS values.

Classical radar processing methods usually rely on clustering algorithms to leverage this information [78]. First, they group together detections which share similar underlying statistics – e.g. spatial proximity. Then, they build a signature that describes the cluster, by collecting statistics over the features of the points in each group. However, these approaches are sub-optimal, as they rely on the clustering algorithm to group together points related to each other – e.g. generated by the same object. This opens-up to the possibility of outliers which can potentially corrupt the cluster signature and, eventually, lead to erroneous final predictions. Finally, it is worth noting that each radar reflection carries relevant target-related features. Since the ability to exploit all the available information represents a crucial aspect for an efficient processing, therefore, it is important that the algorithm possesses the means to leverage this data.

NNs are notoriously good at extracting high-dimensional descriptors from low-level features. Therefore, in a point-wise processing framework, it is possible to leverage this property to encode informative radar measurements into meaningful point-signatures. In this section, a novel data-learned pre-processing module is proposed. It adopts a FC-network to generate high-dimensional point-descriptors based on single-point radar features. In more details, given the generic i-th point with input features $\boldsymbol{x^i} \in \mathbb{R}^m$ – containing x, y, Doppler and RCS –, the proposed pre-processing module uses a shared FC-network – with trainable parameters $\boldsymbol{P_p}$ –, to build a new point-signature $\boldsymbol{x_p^i} \in \mathbb{R}^{m'}$, i.e.

$$\boldsymbol{x_p^i} = f_p(\boldsymbol{x^i}; \boldsymbol{P_p}) = \text{ReLU}\big(\boldsymbol{P_3}^T \text{ ReLU}\big(\boldsymbol{P_2}^T \text{ ReLU}\big(\boldsymbol{P_1}^T \boldsymbol{x^i} + \boldsymbol{p_1}\big) + \boldsymbol{p_2}\big) + \boldsymbol{p_3}\big) \ , \quad (3.1)$$

where $\boldsymbol{P_p} = \{\boldsymbol{P_1}, \boldsymbol{P_2}, \boldsymbol{P_3}, \boldsymbol{p_1}, \boldsymbol{p_2}, \boldsymbol{p_3}\}$ and the Rectified Linear Unit (ReLU) is used as activation function – i.e. $\text{ReLU}(x) = \max(x, 0)$. Note how the new features computed for every point are based uniquely on the radar measurements of the specific point. In this way, the module is forced to learn the task of extracting knowledge from the radar features, as it cannot rely on any other information source. Moreover, neighboring points, with dominating features, cannot eclipse the radar measurements of the current point. Notice also, in eq. (3.1), how the parameter-set $\boldsymbol{P_p}$ is shared across all points. Nevertheless, every point receives a unique signature, steered by its own input features. Figure 3.3 shows the operations of the proposed pre-processing module. Since it has been designed to produce an output that preserves the spatial structure of the input point clouds, the original point locations result unaltered by this operation.

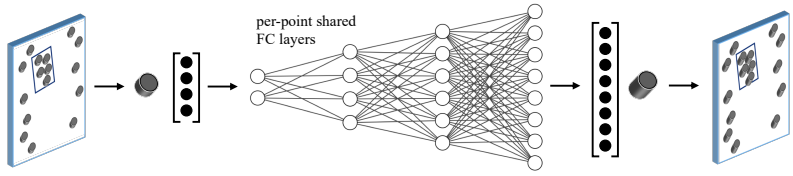


Figure 3.3: The proposed pre-processing module. The input features of every point are combined, by means of a shared FC-network, to obtain a richer feature-set.

The pre-processing module described in this section enables the network to exploit the information embedded in the features of every input point. It provides as output a point cloud with a deeper feature-set, encoding the radar perspective of the objects. As a result, the deeper, latent point cloud retains the radar structural information, but has new point-descriptors that increases the distance between points with diverse radar properties, thus facilitating the classification task. Finally, due to the simplicity of the process, it is possible to easily plug the proposed solution to every point-wise processing architecture, improving their classification performance, as shown in section 3.4.

3.2.2 Altered PointNet++

RadarPCNN adopts the pre-processing module described in section 3.2.1. It generates deeper point clouds, abstracting the radar perception of the targets, however, it does not use any context information. Despite radar provides sparser point clouds than other sensors, to achieve an optimal processing scheme, it is important to exploit structural information. Indeed, by analyzing the features of points close to each other, it is possible to collect useful insights. For instance, consider a point with the common properties of a pedestrian – bearing in mind that pedestrians have specific radar characteristics [24, 31]. Since pedestrians are often located besides static objects – e.g. buildings, traffic-lights or stationary vehicles – additional details can be collected by studying the neighboring points. In particular, if detections of stationary objects lie nearby, it is possible to encode this knowledge and strengthen the point signature. The next stage of the network has been designed to extract context information. Specifically, it has the goal to generate descriptive classification features that enable the detection of the most interesting objects in the scene.

Among various possible solutions, it has been decided to adopt the PointNet++ framework for two reasons. In this way, indeed, RadarPCNN can process raw point clouds and, at the same time, inherits the ability to extract information from local neighborhoods. However, it has been decided to alter the standard architecture in order to take into account the lower density of radar point clouds. In particular, the sampling process in the SA layers has been altered by means of mean-shift (MS) and the hierarchical structure is discarded in favor of an object-oriented approach.

The PointNet++ framework consists in an encoder-decoder structure. Its operations are depicted inside the orange rectangles in fig. 3.2. The encoder section has the task of embedding context information into local descriptors. SA layers are used as building blocks. They perform three operations: sampling, grouping and feature extraction. Formally, consider an input point cloud $\boldsymbol{\mathcal{X}} = \{\boldsymbol{x^i} \in \mathbb{R}^m,\ i = 1, \dots, N\}$. The sampling process selects a collection of points $\boldsymbol{\mathcal{S}_{SA}} \subset \boldsymbol{\mathcal{X}}$ as center of the local descriptors. Then, for each representative point $\boldsymbol{x^j} \in \boldsymbol{\mathcal{S}_{SA}}$, the grouping process clusters together all the points $\boldsymbol{x^i} \in \boldsymbol{\mathcal{X}}$ that lie in its neighborhood. A distance function and a search range define the neighborhood area. Therefore, the outcome of the grouping process is a collection of $|\boldsymbol{\mathcal{S}_{SA}}|$ local point clouds

$$\boldsymbol{\mathcal{G}^j_{SA}} = \{\boldsymbol{x^i} \in \boldsymbol{\mathcal{X}} \mid d(\boldsymbol{x^i}, \boldsymbol{x^j}) < r\} \quad \forall\ \boldsymbol{x^j} \in \boldsymbol{\mathcal{S}_{SA}}\ , \tag{3.2}$$

where $|\cdot|$ is the set-size operator – returning the number of elements in the set –, $d(\cdot, \cdot)$ a suitable distance function and r the search range. Finally, the feature extraction

process abstracts the local point clouds, by encoding their structural information into local descriptors. In order to perform this task, a local shared PointNet [65] network is deployed, using shared FC-networks and the max-pooling operator – as illustrated in fig. 2.9. As a result, the output of an SA layer consists in a point cloud $\boldsymbol{\mathcal{X}_{SA}}$, with locations defined by the points $\boldsymbol{x^j} \in \boldsymbol{\mathcal{S}_{SA}}$ and features calculated as

$$\boldsymbol{x^j_{SA}} = \text{MaxPool}\big(\text{ReLU}\big(\boldsymbol{W_{SA}}^T \boldsymbol{G^j} + \boldsymbol{B_{SA}}\big)\big) \quad \forall\, \boldsymbol{x^j} \in \boldsymbol{\mathcal{S}_{SA}}\ , \tag{3.3}$$

where $\boldsymbol{G^j}$ is the matrix whose columns contain the points $\boldsymbol{x^i} \in \boldsymbol{\mathcal{G}^j_{SA}}$ – the set of neighbors of $\boldsymbol{x^j}$ – and $\boldsymbol{W_{SA}}$ and $\boldsymbol{B_{SA}}$ the learnable parameters of the FC-network – weights and biases, respectively. MaxPool is the max-pooling operator used to return the maximum value in each row. Note that, since the output of an SA layer is still a point cloud, it is possible to stack multiple SA layers by feeding the output of one layer as input to the next one.

The decoder section of the PointNet++ framework, instead, propagates the high-level signatures back at the input point locations. FP layers are the building blocks used to perform this task. They execute an inverse-distance weighted interpolation followed by a learning stage. Consider, without loss of generality, a framework with a single encoder-decoder layer. In this case, $\boldsymbol{\mathcal{X}}$ is the input of the SA layer and $\boldsymbol{\mathcal{X}_{SA}}$ represents both the output of the SA layer and the input of the FP layer. Therefore, the FP layer's objective consists in propagating the information encoded in $\boldsymbol{\mathcal{X}_{SA}}$ into the point locations in $\boldsymbol{\mathcal{X}}$. The first step consists in grouping together, for each point $\boldsymbol{x^j} \in \boldsymbol{\mathcal{X}}$, the three closest points $\boldsymbol{x^i_{SA}} \in \boldsymbol{\mathcal{X}_{SA}}$ – a.k.a. nearest-neighbors – according to a given distance function $d(\cdot,\cdot)$, i.e.

$$\boldsymbol{\mathcal{G}^j_{FP}} = \{\{\boldsymbol{x^{i_1}_{SA}}, \boldsymbol{x^{i_2}_{SA}}, \boldsymbol{x^{i_3}_{SA}}\} \in \boldsymbol{\mathcal{X}_{SA}} \mid \max_{i \in \{i_1,i_2,i_3\}} d_{ji} < d_{jk}\ \ \forall\, \boldsymbol{x^k_{SA}} \in \boldsymbol{\mathcal{X}_{SA}} \backslash \boldsymbol{\mathcal{G}^j_{FP}}\}\ , \tag{3.4}$$

where $\cdot \backslash \cdot$ is the set-difference operator[2] and $d_{ji} \triangleq d(\boldsymbol{x^j}, \boldsymbol{x^i_{SA}})$ for notation convenience. Then, the high-level features of the points in $\boldsymbol{\mathcal{G}^j_{FP}}$ are propagated by means of an inverse-distance weighted interpolation. This procedure results in a point cloud $\boldsymbol{\mathcal{P}_{FP}}$, whose points have the same spatial location of the points $\boldsymbol{x^j} \in \boldsymbol{\mathcal{X}}$, but features computed as

$$\boldsymbol{x^j_{feat}} = \Big(\sum_{\boldsymbol{x^i_g} \in \boldsymbol{\mathcal{G}^j_{FP}}} \frac{1}{d(\boldsymbol{x^j}, \boldsymbol{x^i_g})} \Big)^{-1} \cdot \sum_{\boldsymbol{x^i_g} \in \boldsymbol{\mathcal{G}^j_{FP}}} \frac{1}{d(\boldsymbol{x^j}, \boldsymbol{x^i_g})} \cdot \boldsymbol{x^i_g} \quad \forall\, \boldsymbol{x^j} \in \boldsymbol{\mathcal{X}}\ . \tag{3.5}$$

Notice that the first term on the right-hand side of eq. (3.5) operates as a normalization factor, as it contains the sum of the inverse distances. In this way, the closer the point $\boldsymbol{x^i_g}$ is to the new point $\boldsymbol{x^j}$, the higher its impact on the resulting signature will be. Next, the features of the points $\boldsymbol{x^j} \in \boldsymbol{\mathcal{X}}$ – i.e. the SA layer input – are concatenated with the features of the points in the interpolated point cloud $\boldsymbol{\mathcal{P}_{FP}}$. This skip-connection ensures propagation of the single-points low-level signatures. Finally, new point descriptors are calculated by sharing an FC-network across all points, i.e.

$$\boldsymbol{x^j_{FP}} = \text{ReLU}\big(\boldsymbol{W_{FP}}^T \cdot [\boldsymbol{x^j_p}^T, \boldsymbol{x^j}^T]^T + \boldsymbol{b_{FP}}\big) \quad \forall\, (\boldsymbol{x^j_p},\ \boldsymbol{x^j}) \in (\boldsymbol{\mathcal{P}_{FP}},\ \boldsymbol{\mathcal{X}})\ , \tag{3.6}$$

[2]The set-difference operator is defined such that $\mathcal{A} \setminus \mathcal{B} = \{x \in \mathcal{A} \mid x \notin \mathcal{B}\}$.

where $\boldsymbol{W_{FP}}$ and $\boldsymbol{b_{FP}}$ are learnable weight and bias parameters of the FC-network. Note that the point cloud produced as output of the framework, $\boldsymbol{\mathcal{X}_{FP}}$, has points at the same location of the input point cloud $\boldsymbol{\mathcal{X}}$. However, as the point-descriptors are computed using eq. (3.6), they combine single-point and local structural information. The same derivation can be extended to a multiple SA-FP layers architecture, by coupling the corresponding encoder-decoder layers.

RadarPCNN uses an altered version of the PointNet++ framework to address the properties of the radar sensor. Standard SA layers use the farthest point sampling (FPS) algorithm [7] in the sampling process. FPS works by first selecting a point at random. Next, it selects the point which lies the farthest from the first point. It continues to iteratively select the point that has the largest cumulative distance from all the points already present in the list, until it is filled with the number of points requested as input. Due to its ability to sample uniformly from the input space, FPS is well-suited for very dense point clouds, where the objective is to represent the same information with a lower amount of points. Radar, however, provides sparse data with very localized information. Therefore, it is possible to optimize the usage of structural information by exploring regions with high point-density. For this reason, it has been decided to tamper the sampling process, replacing the standard FPS method with the mean-shift (MS) clustering algorithm [5]. MS allows the localization of local maxima of the distribution defined by a set of observations. It operates by applying a given kernel function $k(x - x^i; h)$ to each data-point x^i. In this way, it is possible to determine the value of the distribution $f(x)$ at the nearby locations x around x^i. Then, by summing-up the contribution of every data-point, MS builds-up the data distribution $f(x)$, i.e.

$$f(x) = \sum_i k(x - x^i; h) \ , \tag{3.7}$$

where $h > 0$ is the bandwidth parameter, controlling the kernel's width. Finally, it is possible to use an iterative optimization algorithm to find the locations of the local maxima in $f(x)$. Figure 3.4 contains a graphical representation of the procedure.

In RadarPCNN, it is proposed to use MS to sample locations of the space where there is a high concentration of points. Therefore, point locations $\boldsymbol{x^i} \in \boldsymbol{\mathcal{X}}$ are

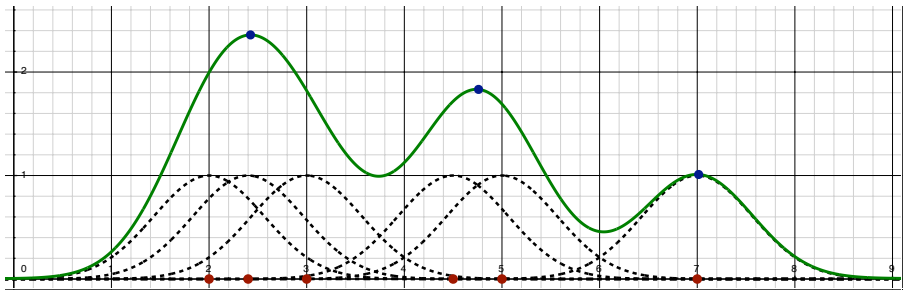


Figure 3.4: Mean shift algorithm applied to a one-dimensional problem. The red points represent the data observations x_i. The black-dotted lines are the kernels applied to each data-point. Gaussian kernels in eq. (3.8) are used, with $h = 0.8$ and $c = h\sqrt{\pi}$. The green curve is the distribution $f(x)$ computed as in eq. (3.7). The blue points are the local maxima selected by the algorithm. Best viewed in color.

used as observation of the distribution $f(\boldsymbol{x})$. In particular, kernels are centered at each point location and summed-up to build $f(\boldsymbol{x})$. Then, the local maxima of the resulting distribution are used as representative points in the SA layer – i.e. to define $\boldsymbol{\mathcal{S}_{SA}}$. The proposed solution adopts multidimensional Gaussian kernels, defined as

$$k(\boldsymbol{x} - \boldsymbol{x^i}; h) = \frac{c}{h\sqrt{\pi}} e^{-\frac{\|\boldsymbol{x}-\boldsymbol{x^i}\|_2^2}{h^2}} \ . \tag{3.8}$$

In eq. (3.8), $\|\cdot\|_2$ is the L_2-norm operator, therefore, $\boldsymbol{x^i}$ acts as the mean parameter of the Gaussian function, while the bandwidth parameter h is proportional to the standard deviation σ – i.e. $h^2 = 2\sigma^2$. The constant c is used to normalize the function – e.g. by setting $c = h\sqrt{\pi}$, the kernel output is constrained in the range $(0, 1)$. It is also worth noting how the bandwidth parameter enables control over the total amount of points produced by MS: the lower it is, the smaller the width of the kernel is, thus the closer two points must be to sum-up their contributions. To understand this effect, it is possible to study the behavior of the algorithm at the two extremes. Large values of h produce wide kernels. As it approaches infinity, the kernel becomes flat, i.e.

$$\lim_{h\to\infty} \frac{c}{h\sqrt{\pi}} e^{-\frac{\|\boldsymbol{x}-\boldsymbol{x^i}\|_2^2}{h^2}} = 0 \ , \tag{3.9}$$

thus, every point is affected by any other point, regardless of their relative position. This configuration, however, sums-up to a constant distribution function, hence producing no local maxima solution. Therefore, it produces the same output of the FPS algorithm when 0 points are requested. On the other hand, as h gets smaller, the kernel starts to shrink, until it becomes an impulse. Formally, indeed,

$$\lim_{h\to 0} \frac{c}{h\sqrt{\pi}} e^{-\frac{\|\boldsymbol{x}-\boldsymbol{x^i}\|_2^2}{h^2}} = \delta(\boldsymbol{x} - \boldsymbol{x^i}) \ , \tag{3.10}$$

where $\delta(\boldsymbol{x} - \boldsymbol{x^i})$ is the Dirac delta centered at $\boldsymbol{x^i}$, namely,

$$\delta(\boldsymbol{x} - \boldsymbol{x^i}) = \begin{cases} \infty, & \text{if } \boldsymbol{x} = \boldsymbol{x^i}, \\ 0, & \text{otherwise} \ . \end{cases} \tag{3.11}$$

In this configuration, contributions of different points cannot sum-up, as the kernel has infinitesimal width. Therefore, eq. (3.7) generates a distribution which has peaks at the location of the data-observations and is null elsewhere. Since this function has local maxima located at all the input point locations, this is tantamount to the FPS algorithm, when it is requested to sample all the input data-points.

Although, under special configurations, MS can be equivalent to FPS, the two algorithms are significantly different. Indeed, in normal settings, MS samples representative points from high-density regions, respecting the distribution of the input data. On the other hand, FPS performs a uniform sampling, covering the whole input as much as possible. Moreover, note that MS can produce as output clusters, assigning the input points according to the cluster they contributed to. Yet, RadarPCNN does not use this information to avoid incurring in the limitation of the clustering algorithm. The grouping process, therefore, remains unaltered. Here, the Euclidean distance is used to define the neighborhood area. Formally, given the

coordinates (x^i, y^i, z^i) of the i-th point $\boldsymbol{x^i}$ and the coordinates (x^j, y^j, z^j) of the j-th point $\boldsymbol{x^j}$, it is defined as

$$d^E(\boldsymbol{x^i}, \boldsymbol{x^j}) = \|\boldsymbol{x^i} - \boldsymbol{x^j}\|_2 = \sqrt{x^2 + y^2 + z^2}\ , \tag{3.12}$$

where $x = x^i - x^j$ and y, z are computed accordingly. Although other solutions have been implemented to improve the grouping stage, the Euclidean distance provides the best performance – further details in section 3.4.2. About the decoder section of the framework, unaltered FP layers are used, maintaining all the original operations.

Finally, it has been observed that SA layers work by encoding dense, low-level data into sparser high-level point clouds. In order to avoid further sparsification of the input in the hidden-layers, it has been decided to abandon the hierarchical PointNet++ structure. In this way, indeed, it is possible to avoid sub-sampling from a sub-set of the input points, operation that might prove fatal on sparse radar data. Instead, it has been used two independent, object-focused, single layer mini-PointNet++ – as illustrated in top and bottom branch of fig. 3.2. In particular, one network is configured to focus on small objects such as pedestrians, bikes, etc (top branch). It uses a dense sampling strategy (MS with bandwidth 2.0) and groups points in small neighborhood areas (below 2 meters around the representative point). The other network (bottom branch), instead, is set-up to extract information from larger objects such as cars, trucks, etc. Therefore, it uses a higher MS bandwidth value (i.e. 8.0) to produce sparser representative point clouds. Then, it uses larger neighboring areas to group together nearby points (4 to 8 meters). As a result, the output of this stage of RadarPCNN consists in two point clouds $\boldsymbol{\mathcal{X}_1}$ and $\boldsymbol{\mathcal{X}_2}$, sharing the same point locations. Yet, they provide different point descriptors, as they use diverse settings to encode context-related information.

3.2.3 Attention Mechanism

In order to semantically segment a point cloud, it is necessary to produce a signature for each point, containing relevant information for the classification task. The processing scheme of RadarPCNN presented so far, has been designed to extract both single-point and context-based information. However, since it uses two altered PointNet++ networks, it generates two signatures for every input point. Therefore, RadarPCNN requires a module that has the ability to combine different informative patterns into a single point-descriptor. Note that it is possible to accomplish this task using very simple methods. For instance, a suitable solution consists in the addition operation. Indeed, by summing element-wise two equally-sized vectors, it is possible to merge them into a single one. However, this operation can potentially destroy part of the information present in the vectors. Another simple solution is represented by the concatenation operation. As this method chains together the two vectors, it does not alter its elements, hence preserving all the information. Yet, it increases the dimensionality of the output vector, thus raising the computation complexity of the algorithm. Moreover, as it just propagates the two vectors, it leaves the filtering task entirely to the final classification network.

Similarly to [87], RadarPCNN has the ability to decide the best combination of the two point-signatures. Indeed, it is equipped with the attention mechanism in fig. 3.5. This module collects, for each point, the different signatures extracted by top (green point) and bottom (red point) branch. Then, it fuses their features

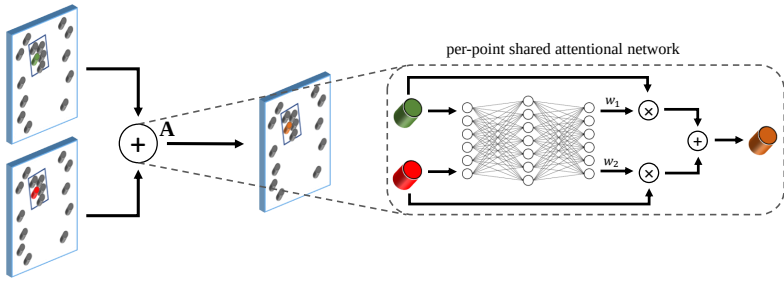


Figure 3.5: Attention mechanism. It fuses together different signatures of the same point through a learned weighted combination. Best viewed in color.

together by means of a learned weighted combination. An FC-network is used to compute the weights, sharing its parameters across the different signatures of every point. Formally, given the i-th point and the features $\boldsymbol{x_1^i} \in \boldsymbol{\mathcal{X}_1}$ and $\boldsymbol{x_2^i} \in \boldsymbol{\mathcal{X}_2}$ learned in the two altered PointNet++, it calculates the scalar attention weights w_1^i and w_2^i as

$$w_1^i = f_a(\boldsymbol{x_1^i}; \boldsymbol{A}, \boldsymbol{a}) = \sigma\Big(\boldsymbol{a}^T \ \mathrm{ReLU}\big(\boldsymbol{A}^T \boldsymbol{x_1^i}\big)\Big) \quad \text{and} \quad w_2^i = f_a(\boldsymbol{x_2^i}; \boldsymbol{A}, \boldsymbol{a}) \ , \tag{3.13}$$

where $\boldsymbol{A}$ and $\boldsymbol{a}$ are learnable parameters of the FC-network and $\sigma(\cdot)$ the sigmoid activation function – i.e. $\sigma(x) = (1+e^{-x})^{-1}$. Notice how the same network parameters are used to compute the attention weights of both vectors. Moreover, as they are independent from the index i, they are shared across all points in the point cloud. In this way, the network cannot be biased towards signatures from a specific branch. Therefore, it learns to compute the weights based on the saliency of the patterns present in the vectors. Finally, the signatures $\boldsymbol{x_1^i}$ and $\boldsymbol{x_2^i}$ are merged into a single classification vector $\boldsymbol{x_{ATT}^i}$ (orange point) by means of a weighted combination, i.e.

$$\boldsymbol{x_{ATT}^i} = w_1^i \cdot \boldsymbol{x_1^i} + w_2^i \cdot \boldsymbol{x_2^i} \ . \tag{3.14}$$

The proposed attention mechanism enables the network to perform an intelligent fusion. Similarly to the addition operation, it performs a linear combination of the two vectors. However, it explores the features of the signatures to generate the weights. Therefore, it can decide to increase the contribution of the vector with more relevant patterns for the final task, thus filtering out information.

3.2.4 Classification Network

The attention mechanism outputs a point cloud $\boldsymbol{\mathcal{X}_{ATT}}$ with points located as in the input point cloud $\boldsymbol{\mathcal{X}}$. Each point, has a descriptor vector, computed as in eq. (3.14), encoding radar-specific point-properties as well as context information gathered from small and large objects. This point cloud is fed to a classification network to produce the final semantic segmentation scores. It consists in a multi-layer FC-network shared across the points, i.e.

$$\boldsymbol{x_o^j} = \boldsymbol{W_{O_3}}^T \ \mathrm{ReLU}\big(\boldsymbol{W_{O_2}}^T \ \mathrm{ReLU}\big(\boldsymbol{W_{O_1}}^T \boldsymbol{x^j} + \boldsymbol{b_{O_1}}\big) + \boldsymbol{b_{O_2}}\big) \quad \forall \ \boldsymbol{x^j} \in \boldsymbol{\mathcal{X}_{ATT}} \ , \tag{3.15}$$

Notice that no activation function is used at the last layer of the classification network. In this way, it is possible to cast either a multi-class classification problem or several – dependent or not – binary ones.

The output point cloud $\boldsymbol{\mathcal{X}_O}$ is made-up of points which maintains the input point locations, but features $\boldsymbol{x^i} \in \mathbb{R}^d$, where d represents the number of output semantic classes. Therefore, every feature of the points in $\boldsymbol{\mathcal{X}_O}$ holds a non-normalized score associated with one output class. Generally, the higher the score is, the more certain the network is about predicting a specific class. Finally, it is worth noting that, as every layer entails a differentiable operation, the whole network can be trained in an end-to-end fashion. Therefore, the output segmentation loss can be used to train the network, without the need of any intermediate loss.

3.3 Experimental Setup

This section reports the experimental setup adopted during the evaluation process. The proposed solutions are tested on a two-class semantic segmentation task. In particular, RadarPCNN has been trained to recognize detections from moving pedestrians or moving vehicles. Points generated by other objects in the scene or clutter are assigned to the negative background class. Results are reported in terms of precision, recall and F_1 score on the two positive classes. Recall measures the ability of the network to recognize all points of a given class, while precision quantifies the correctness of the predictions. F_1 score provides a summary of the network performance, as it calculates the harmonic mean between precision and recall. Formally, it is possible to define precision (P_r), recall (R_e) and F_1 score as

$$P_r = \frac{TP}{TP + FP} \quad , \quad R_e = \frac{TP}{TP + FN} \quad \text{and} \quad F_1 = \frac{2}{\frac{1}{P_r} + \frac{1}{R_e}} = 2 \cdot \frac{P_r \cdot R_e}{P_r + R_e} \ , \tag{3.16}$$

where TP, FP and FN are, respectively, true positive, false positive and false negative samples[3]. Note that the denominator of the precision formula corresponds to the total amount of positive predictions, while in the recall formula, it contains the total amount of positive samples. In order to extract the results, the multi-class task has been cast into two binary problems. For instance, the results collected on the pedestrian class consider pedestrian points as positive, and both vehicles and other points as negative. In addition, the macro-average version of the metrics are reported, to provide an overview of the network performance on the multi-class task. They are computed by averaging the single-class results, without considering the amount of positive samples – i.e. each class provides an equal contribution. For completeness, confusion matrices are also provided, showing the distribution of the network predictions on all three classes – i.e. pedestrian, vehicle and other. Finally, as part of a visual investigation, semantic segmentation predictions on real-world radar point clouds are included.

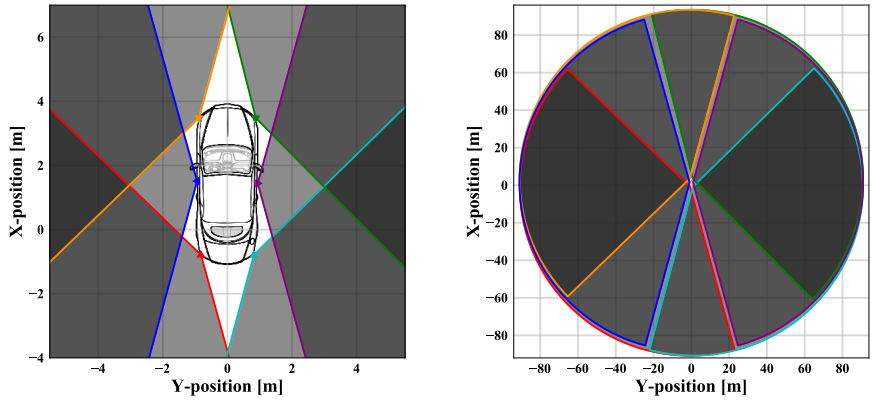
3.3.1 Dataset

The evaluation of the proposed solutions suffers from the shortages of publicly available radar datasets. To circumvent this obstacle, a customary solution consists in the adoption of simulated (or synthetic) data. By means of a computer-run simulation, indeed, it is possible to reproduce the environment where the system is supposed to

[3]TP are positive samples correctly predicted, FP are negative samples erroneously predicted as positive and FN are missed positive samples, predicted as negative.

operate as well as the resulting sensor readings. For instance, an urban scene can be reconstructed around the ego-vehicle and the corresponding radar measurements can be simulated. However, although synthetic results can provide significant insights to steer the early stages of development, they are not sufficient to fully assess the performance of a system. Indeed, simulations provide a simplified version of the real-world environment, neglecting complex, non-linear effects that would influence the sensor operations in the real-world. This holds particularly true for radar, which is exposed to several factors hard to implement in a simulated environment. For instance, the radar signal is significantly affected by marginal details such as material of the target and/or roughness of the reflective surface. This introduces a gap between real-world and synthetic data, leading to biased evaluation results.

For these reasons, it has been decided to collect a proprietary dataset. In particular, a fleet of vehicles were equipped with six FMCW 77GHz radars and deployed to record real-world radar reflections in urban scenes under clear, rainy and cloudy weather conditions. The sensors were mounted uniformly on the vehicles. Figure 3.6A shows the specific configuration, which consists in two front radars off-set by $\pm 60°$, two lateral sensors and two rear sensor off-set by $\pm 120°$. Note that this mounting setting results in blind spots in front and rear of the vehicle, extending for 2.7 m circa. Yet, as shown in fig. 3.6B, it enables coverage of most of the scene around the ego-vehicle with at least two sensors. In this way, it is possible to introduce the necessary redundancy to guarantee operational safety of the system even under the scenario of a malfunctioning sensor. Each radar collected reflections from targets up to 90 meters, in a field-of-view of $\pm 75°$. The total perception area of the vehicles and its distribution across sensors is shown in fig. 3.6B. The highest Doppler resolution is 0.13 m/s, while in range it is 30 cm. Azimuth resolution is fixed at



(A) Sensor mounting configuration.

(B) Total covered area.

Figure 3.6: Sensor mounting configuration and coverage. The ego-vehicle is shown at the origin. Different sensors are represented by triangle markers with different colors, oriented according to the sensor. Colored lines indicate the boundaries of the FoV. Gray regions shows the area covered by the sensors. Darker gray areas result from the superposition of multiple sensors. Best viewed in color.

$\pm 7.5°$. The recording frequency is 20 Hz. In total, the collected dataset counts 84 video sequences and more than 40 million points, annotated as pedestrian, vehicle or other/clutter. BB labels are used to represent objects in the scene and associate every point with the corresponding class. In particular, every point is located in the scene: if it falls inside a BB, it is labeled as the corresponding object class; otherwise it is labeled as other/clutter. Since pedestrians have very small sizes – the average BB size is 70 cm –, their BB boundaries are extended by 30 cm to avoid mislabeling of points just outside them. Manual-human labels as well as computer-generated ones are adopted.

Automotive radar point clouds generally contain weak stationary object information. Although OGM algorithms can enable the extraction of static context, the detection of stationary objects is a very challenging task. Indeed, they do not have a unique radar signature, as they share the same Doppler of other still instances (e.g. buildings). In addition, the sparsity of radar point clouds prevents the accumulation of enough local points, thus hindering the extraction of shape patterns. For this reason, is has been decided to focus on moving objects. In this setup, a critical decision consists in the definition of moving object. As vehicles and pedestrians have significantly different velocity profiles, it has been decided to use different thresholding configurations. In particular, a pedestrian is considered moving if it has a ground-truth velocity greater than 0.5 m/s, while for vehicles is used a threshold value of 2.5 m/s. Note that the velocity threshold is a very sensitive parameter, as wrong settings can either create a trivial problem or mislabel stationary objects as moving. However, its configuration affects the evaluation mostly from a numerical perspective. Indeed, regardless of the threshold values, the network can correctly classify points from objects with low velocity as moving – as shown in section 3.4.3. Consequently, reflections from objects with velocity values below the selected thresholds are forced to be transparent during both training and evaluation process. Yet, points from objects that do not move at all in the scene – e.g. parked vehicles, poles, trees – are assigned to the other/clutter class. Table 3.1 contains the point distribution in the dataset. Note that both train and test set have similar distributions of class-points and rainy/cloudy/clear scenes.

3.3.2 Settings

The video sequences in our dataset are divided in frames. Since each frame contains only a few hundred radar-detections, similarly to [81], it has been decided to enrich the input by aggregating points from previous frames. A window of 200 ms is used and points from past frames are shifted to compensate for the ego-vehicle movement. Note that, despite this solution does not introduce shape patterns, it increases the input point cloud context representation. Due to the low elevation resolution of the

Table 3.1: Distribution of points in the dataset. Percentage and total number of points are reported.

Set	Moving Pedestrian	Moving Vehicle	Other & Clutter
Train	2.26% (721 865)	6.12% (1 955 937)	91.62% (29 282 877)
Test	1.40% (142 647)	5.63% (574 321)	92.98% (9 493 667)

sensor, it has been decided to discard this point feature. Therefore, the input point cloud has four features: x and y spatial coordinates, RCS and Doppler. This latter is used as z-spatial-coordinate to increase the geometric distance between objects with different velocity profiles. Specifically, the ego-compensated Doppler value is adopted, containing the measured radial relative target's velocity, compensated by the ego-velocity – i.e. absolute radial velocity. RCS is the only point-feature. The input is set to have a fixed amount of 1200 points. Since the primary goal is the detection of moving objects, the ego-compensated Doppler feature is used to sample with importance: i.e. reflections with large Doppler values are prioritized by receiving higher sampling probabilities. Consequently, if the input has more than 1200 points, it is more likely to retain radar-reflections from moving instances[4]. Conversely, if less than 1200 points are available, reflections are duplicated in the same fashion. In this way, it is possible to increase the positive classes population, hence reducing a critical aspect such as the class imbalance of the task.

RadarPCNN has been designed to process raw point clouds. Therefore, it receives as input the list of 1200 points with four features $(3 + 1)$. The pre-processing module uses three FC layers with 8, 16 and 32 channels, respectively. As a result, its output consists of a point cloud with 1200 points and $3 + 32$ features. The two altered PointNet++ use different SA-layer configurations. One uses 500 representative points, three grouping areas (0.5, 1 and 2 meters) and outputs a point cloud with $3 + 192$ channels. The other uses 150 representative points, grouping areas of 4, 6 and 8 meters and $3 + 384$ output channels. The representative points are produced using MS. If more than the requested points are generated, FPS is used to select among them. Conversely, if less points are available, FPS is used to fill the gaps, selecting from the input points. Both altered PointNet++ use the same FP-layer configuration, producing a list of 1200 points with $3 + 128$ features. The attention mechanism adopts three FC-layers of size 8, 4, and 4 to produce the attention weights. Another FC-layer is then used to increase the point-features to 256. Finally, the classification network has two layers of 64 and 32 channels – interleaved by dropout with rate 0.5 –, before the output layer.

As shown in table 3.1, the points in the dataset are not evenly distributed across the three classes of the task. Note, for instance, how the negative class has much more samples than the two positive ones. This situation, a.k.a. class imbalance, poses serious difficulties in the learning process. Indeed, the lower availability of samples makes it more complicated for the network to learn the properties specific for a class. Therefore, it will develop a bias towards the class with the highest amount of observations – i.e. majority class. It is not uncommon, however, to have imbalanced datasets, especially for semantic segmentation tasks, where the minority class often includes the objects of interest. Consider, for example, an autonomous vehicle in an urban environment. Here, it is reasonable to assume that pedestrians will contribute to a lower amount of observations compared with other objects, e.g. cars or buildings. Nevertheless, pedestrians detection is a critical task to ensure safety operations of the system. Dataset imbalance represents a well-established problem in the DL community, so that researchers have developed various countermeasures, applicable at different stages, to mitigate its effects [116, 94]. In this work, it has been decided to operate in both design and training stages.

[4]Bear in mind that Doppler represents a radial velocity measure. Therefore, objects that move in tangential direction result in stationary Doppler values.

During the design phase, the output layer of the network has been customized to reduce bias toward the negative class (other/clutter). Specifically, the multi-class classification problem has been discarded in favor of multiple binary tasks. Though the dataset provides three classes, only two binary tasks are addressed, targeting the positive classes – i.e. – pedestrian vs non-pedestrian and vehicle vs non-vehicle. Sigmoid is used as output activation function to constrain the prediction scores within the $[0, 1]$ interval. Therefore, two scores are calculated for every point, holding the probability of pedestrian and vehicle class prediction. Since the semantic segmentation task at hand is mutually exclusive[5], the final class prediction will be the one achieving the highest score. However, in case the highest score does not exceed the threshold of 0.5, the negative class will be predicted. In this way, it is possible to train the negative class indirectly from both binary tasks. As a consequence, the network does not need to learn properties of the negative class. Rather, it has to recognize when a sample does not have the properties of a positive class. Finally, note that the two binary tasks are even more imbalanced than the original task. However, during training, it is easier to counteract a binary imbalanced problem than a multi-class one. To do so, the two tasks are trained using different, independent objective functions. Specifically, focal loss [110] has been used, as it is well-known for dealing with extreme class imbalance configurations. It addresses an objective function that enables down-scaling of the gradient contribution from member of the majority class. At the same time, it forces the network to learn from hard examples, by increasing its impact in the training procedure. Formally, given the output prediction score $p \in [0, 1]$ with associated ground-truth value $y \in \{0, 1\}$, focal loss is computed as

$$\mathrm{FL}(p, y) = \mathrm{FL}(p_t) = \alpha_t (1 - p_t)^{\gamma} \mathrm{CE}(p_t) \ , \tag{3.17}$$

where $\gamma \geq 0$ is the focusing parameter, $\mathrm{CE}(p, y) = \mathrm{CE}(p_t) = -\log(p_t)$ the cross-entropy loss function [2] and p_t, α_t defined, for convenience, as

$$p_t = \begin{cases} p, & \text{if } y = 1, \\ 1 - p, & \text{otherwise,} \end{cases} \quad \text{and} \quad \alpha_t = \begin{cases} \alpha, & \text{if } y = 1, \\ 1 - \alpha, & \text{otherwise .} \end{cases} \tag{3.18}$$

The parameter $\alpha \in [0, 1]$ weights the contribution of samples from different classes. Instead, the factor $(1 - p_t)^{\gamma}$ modulates the standard cross-entropy loss by reducing the contribution from easy examples. For the proposed dataset, α has been set to 0.85 for the vehicle task and 0.9 for pedestrians, while γ has been set to 2 for both.

In order to set the hyper-parameters of the model, five-fold cross-validation is performed. That is, the train-set is split in five non-overlapping sets and, in turn, one is used as validation-set and the remaining to train the model. The network parameters are changed until they do not lead to performance improvements any longer. During the evaluation process, the networks are trained on the whole train-set for 20 epochs. The network configuration achieving the best performance is then selected and tested 10 times on the test-set to average out the sources of randomness. Mean values are finally reported, while standard deviations – always below 0.1% – are omitted for clarity.

[5]Points cannot have multiple class predictions as one excludes the others.

3.3.3 Benchmarking Solutions

In this work, it is addressed the task of detecting moving instances from classes such as vehicle and pedestrian. However, radar has the ability to measure dynamic properties of the objects. In particular, the Doppler measurement provides information about relative radial velocity. This aspect raises logical questions about the potential triviality of the task. Indeed, one might think that a simple thresholding technique, using Doppler as input, would suffice to separate moving from stationary object-points. For this reason, it has been decided to use a random forest (RF) model [20] to perform the proposed semantic segmentation task. RF is an ensemble-based, advanced thresholding ML technique using a collection of decision trees [6, 8] to infer the classification output. After experiencing a labeled dataset, decision tree models can classify a sample by comparing its input features with a set of learned thresholds. RF leverages the output of several decision trees to compute a low-variance prediction. Further details are provided in section 4.2.4. Since its processing include thresholding of the Doppler feature, the complexity of this algorithm should suffice to solve a trivial problem.

In order to establish the validity of RadarPCNN and the solutions proposed herein, it is necessary to compare its performance with a benchmark. In the field of point-wise processing, PointNet++ represents the state-of-the-art static architecture for consuming radar point clouds. In particular, Schumann et al. have recently tested its ability to perform a multi-class semantic segmentation task [81]. Therefore, it has been decided to use this architecture implementation as benchmark. Minor modifications have been taken to adapt both input and output model interfaces to the task at hand. Specifically, it has been accounted for the reduced number of input points – halving the number of representative points in the SA-layers – and output classes – halving the size of the FC-classification-network. In details, the model consists of 3 encoder-decoder layers. The SA-layers use, respectively, 512, 256 and 128 representative points and outputs 192, 192 and 256 features. All layers use two grouping areas, with increasing radii: i.e. the first 1 and 3 meters, the second 2 and 4 meters and the last layer 3 and 6 meters. Standard FPS is used as sampling strategy. The FP-layers use 256, 128 and 128 output channels, respectively. Finally, the classification FC-network has two layers of size 128 and 64 channels, interleaved by dropout with rate 0.5. The final output layer has 2 nodes, associated with the positive classes of the task, using the same configuration of RadarPCNN – described in section 3.3.2.

During experimentation, it has been found out that the PointNet++ network used as benchmark has a much higher capacity than RadarPCNN. Indeed, as shown in table 3.2, it has more than 2× the amount of parameters of RadarPCNN and uses 1.5× its number of floating-point operations (FLOPS). However, in order to enable fair comparison, it is crucial to use models with similar capacity, as bigger models usually lead to better generalization results[6]. For this reason, it has been designed a shallow version of PointNet++. In particular, the depth of the model has been reduced, removing one layer in both encoder and decoder sections. Therefore, the model has two SA-layers, set similarly to RadarPCNN. In details, the first SA layer uses 500 representative points and 128 output features, while the second uses

[6]The model capacity must be appropriate for the problem, as too big models suffer from overfitting. A typical example is a high-degree polynomial model fit to noisy observations of a linear distribution.

150 points and 256 output channels. The search areas are set as in RadarPCNN. The hierarchical processing structure proper of PointNet++ has been retained, as well as the standard FPS sampling strategy. The decoder layers take as input the output of the corresponding encoder layer and produce, respectively, point clouds with 500/1200 points and $3 + 128/64$ features. Finally, the classification network is configured as in RadarPCNN. As shown in table 3.2, this shallow PointNet++ architecture has a capacity that matches RadarPCNN, thus enabling fair comparison.

Ultimately, it has been decided to evaluate the shallow version of PointNet++ equipped with the proposed pre-processing module – (sh. + pr.) in table 3.2. Though this model has a capacity slightly higher than RadarPCNN, it enables investigation of the pre-processing module's properties. Indeed, by analyzing the impact on the performance of the model, it is possible to evaluate its ability to generalize to other point-wise processing architectures. The same configuration of the pre-processing module used in RadarPCNN has been adopted.

3.4 Results

Table 3.2 reports the semantic segmentation results of the evaluated models. The first row shows the performance of RF, providing information about the potential triviality of the task. Note how, compared with other more advanced solutions, it achieves considerably inferior performance. This proves that the problem is not trivial, as moving objects cannot be detected by a thresholding operation on the radar features. Analyzing the results in more details, it is possible to note that RF can achieve competitive performance on the moving vehicle class, while struggling to detect pedestrians. This is attributed to the different velocity profiles of the two classes. Indeed, vehicles present a strong velocity signature: when moving, they often assume high speeds, thus neatly differing from stationary targets. Conversely, pedestrians exhibit lower velocities, therefore, it is more complicated to characterize them through this property. Table 3.2 also shows that RF's performance suffers from poor precision. This is attributed to the simplicity of the algorithm, which uses only thresholding operations to infer the output class. Indeed, this prevents the model from using context information, thus failing to filter out potential outliers – e.g. the result of intrinsic radar effects like multi-path propagation. As an example, consider a single point, with high Doppler, immersed in a cluster of stationary points. Since it shares the same characteristics of moving points, the model cannot infer the correct point-class without using context information. Finally, bear in mind that recall and precision are strongly correlated metrics. Therefore, to obtain appreciable recall values, it is necessary to tune down the precision, thus allowing a higher number of false positive predictions. In thresholding-based systems, this trade-off has a straightforward interpretation. Indeed, it is possible to increase the precision by using a stricter threshold-configuration for the positive classes. For the problem at hand, however, a low-recall high-precision model would be useless, as most of the objects would go by undetected. Nevertheless, note how both precision and recall scores are consistently below those produced by the best models in the table. Conclusively, though RF proves the non-triviality of the problem, its performance suggest the presence of a dominant feature – most likely the radar Doppler. This aspect increases the importance of an efficient processing scheme which captures, along with structural context, the information provided by the sensor.

Table 3.2: Semantic segmentation results on the radar dataset (%). Inference time is computed on Nvidia GeForce RTX 2080 (ms).

	Moving Pedestrian			Moving Vehicle			Average					
Method	Prec.	Recall	F_1	Prec.	Recall	F_1	Prec.	Recall	F_1	Param.	FLOPs	Time
Random Forest	16.92	70.43	27.29	54.49	84.09	66.13	35.71	77.26	46.71	—	—	*31.4
PointNet++ [81]	46.10	75.77	57.33	72.77	82.89	77.50	59.44	79.33	67.41	418.6K	1.55G	31.2
PointNet++ (shallow)	43.82	75.31	55.40	71.18	84.01	77.06	57.50	79.66	66.23	191.2K	**0.95G**	28.5
PointNet++ (sh.+pr.)	47.70	**76.77**	58.84	73.16	**93.24**	81.99	60.43	**85.01**	70.42	198.8K	1.02G	30.6
RadarPCNN (ours)	**48.64**	75.82	**59.27**	**75.78**	91.44	**82.88**	**62.21**	83.63	**71.07**	**175.3K**	1.02G	**26.8**

* CPU time.

Among NN-based solutions in table 3.2, RadarPCNN achieves the best semantic segmentation results. It shows superior performance on both vehicle and pedestrian classes. Interestingly, despite its higher capacity, the PointNet++ implementation from [81] exhibits poorer performance than RadarPCNN. It shows competitive results on the pedestrian-point detection task, but underperforms on the vehicle class. This suggests that RadarPCNN executes a more efficient radar point cloud processing, achieving better performance with less capacity. The third row contains the results of the shallow PointNet++ architecture. It experiences an expected performance degradation when compared with the original network. Both classes are affected, though the pedestrian class sustains a more noticeable drop. This effect is attributed to the reduced model capacity. Indeed, since the network uses less parameters – with the same processing scheme –, it can store less information-patterns, thus resulting in loss of generalization power. Yet, these results rule out the overfitting problem on the original PointNet++ network. Otherwise, the shallow model would have produced either improved or equal performance, as reduction of the model capacity is a common countermeasure against overfitting. Finally, note how this architecture and RadarPCNN use a similar amount of parameters/FLOPS. Therefore, the shallow PointNet++ network represents a fair comparison for validation of the proposed architecture. Since the results in table 3.2 show that RadarPCNN significantly outperforms the shallow PointNet++ network, they prove that the proposed solution exceeds its benchmark on semantic segmentation of radar point clouds.

The fourth row in table 3.2 contains the results of the shallow PointNet++ network equipped with the proposed pre-processing module. This architecture shows very competitive performance, challenging the superiority of RadarPCNN. However, the most interesting insights can be gathered by comparing its results with the ones from the previous PointNet++ models. Note, here, the performance boost produced by the pre-processing module over the shallow PointNet++ network, dramatically improving precision and recall scores on both classes. This proves the good transferability properties of the pre-processing module: it can be successfully plugged to other point-wise processing architectures, enhancing their ability to learn from radar features. In addition, these results suggest that the proposed module succeeds in abstracting the radar perspective of the object, producing point descriptors that facilitates the semantic segmentation task. Other interesting observations can be collected by examining the capacity of the networks. Indeed, the introduction of the pre-processing module increases both number of parameters and FLOPS. However, this configuration still shows lower complexity than the original PointNet++ model – it has less than half parameters. Nevertheless, it results in better performance, thus proving that the proposed module is not only effective but also efficient. Finally, it is worth noting how, despite using circa 15% more parameters, the improved version of shallow PointNet++ cannot outperform RadarPCNN. This hints that, though the pre-processing module plays a major role, all the proposed solutions are effective, optimizing the usage of radar point cloud information. Therefore, RadarPCNN performs an efficient processing, requiring a lower model capacity to obtain improved performance.

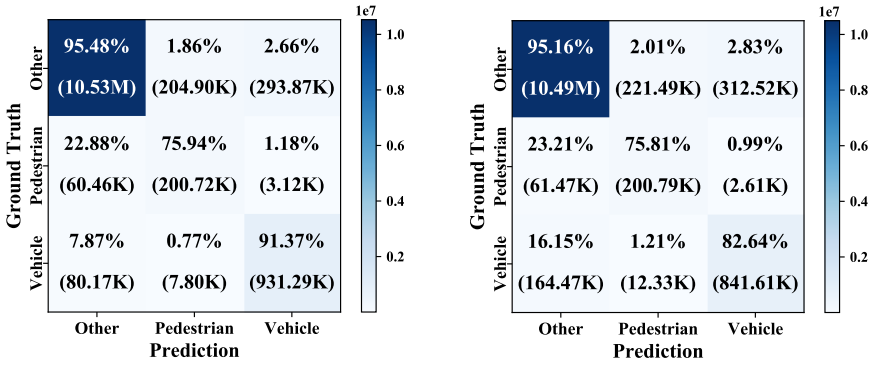
Another metric provided in table 3.2 is the inference time – i.e. the amount of time required by the model to perform the prediction on a single input. The results reported for RF cannot be compared, as they have been measured on CPU. Yet, this model produces much poorer semantic segmentation performance than the

NN-based solutions. Note how the inference time performance of the networks are directly correlated to their capacity: i.e. bigger models require more time to produce an output. However, this observation does not always hold true. Indeed, when a model is deployed on GPUs, its operations are not all executed in a sequential fashion, but they are parallelized based on the architecture structure. Among the tested models, RadarPCNN is the fastest approach. This is attributed to the non-hierarchical network structure. Indeed, given enough computational resources, the two altered PointNet++ mini-networks can be run in parallel, due to their mutual independence. Interestingly, the shallow PointNet++ network is almost 2 ms slower, despite its lower amount of FLOPS. That is because the encoder-decoder operations cannot be parallelized: every layer must wait for the output of the previous layer. Finally, note how the pre-processing module increases the inference time of the shallow PointNet++ model. Since it has to be run before the main network, it adds sequential operations, accounting for 2 ms circa. Nevertheless, this increment is justified by the huge improvement in terms of semantic segmentation performance.

3.4.1 Confusion Matrices

A confusion matrix (CM) provides quantitative information about the predictions of a classification algorithm. It is useful to detect prediction errors, such as confusion across different classes, and check the presence of eventual biases – i.e. the tendency to predict a given class. Figure 3.7 displays the CMs produced by both RadarPCNN and PointNet++ [81]. Each row of the matrices contains all the samples of a single ground-truth class, while each column contains all the samples predicted as a given class. As a result, each column of a given row shows how samples of the same class are predicted by the network. Likewise, each row of the same column reports the ground-truth class of all points predicted as the same class. For instance, the cell identified by the second-row third-column refers to pedestrian-points, classified as vehicle-points. The number in brackets shows the total amount of pedestrian samples that are classified as vehicles. Instead, the percentage refers to the proportion of ground-truth pedestrian-points for which the vehicle class has been predicted. Therefore, the diagonal of a matrix displays information about correct predictions, while all the other cells provides quantitative details about the model errors. Note that a perfect model would produce an identity matrix, with 1 – or 100% – on the diagonal and 0% everywhere else. Moreover, note that the percentage values reported on the diagonal correspond to the recall rates.

The CMs showed in fig. 3.7 confirm the insights collected from table 3.2. Notice, indeed, how RadarPCNN outperforms PointNet++ [81], producing higher percentages on the diagonal as well as lower percentages on the non-diagonal cells. By analyzing more accurately the scores in the error-related cells, it is possible to gather further insights about the network. Hereafter, it has been focused on the CM from RadarPCNN. Note how, in total, only 6.5% circa of the points in the test-set are erroneously classified by the proposed approach. In the 2×2 lower-right sub-matrix, it is possible to examine the error distribution over positive classes. Note how the confusion between them accounts only for a limited amount of errors. Indeed, they generate confusion factors below 2%, indicating that only a small portion of samples from the positive classes are mistakenly classified as the wrong positive class. In addition, less than $11K$ of points account for such errors, representing only 0.1%



(A) RadarPCNN. (B) PointNet++ [81].

Figure 3.7: Confusion matrices of RadarPCNN and PointNet++ [81]. The color-map shows the points count: the darker the color, the higher the points number (in brackets). Best viewed in color.

of the test-samples. This suggests that the network has a good knowledge about the difference between the two positive classes. The same observations cannot be repeated for the other error-sources. Indeed, almost 5% of errors committed by the network consist of positive class predictions of samples from the negative class. Though these errors do not affect the network recall rates, they severely harm the positive class precision rates. The remaining 1.4% circa of the errors are negative class predictions of positive samples. Note that they account for a consistent portion of the positive ground-truth samples – almost 23% and 8%, respectively, for pedestrians and vehicles. Yet, they hardly affect the precision rate of the negative class, as they constitute as low as 1.3% of the predictions. The strong class-imbalance present in the dataset plays a central role in the latter two error-sources. Indeed, the negative class is exceedingly overpopulated, with only less than 10% of the samples contributing to the positive classes. This disparity severely hardens the learning process, leaving the network marginally biased towards the negative class, despite the countermeasures implemented. As support to this thesis, note how the network is very confident about the negative class. It can achieve 95% recall and 98% precision, though this task has been learned indirectly. Similar considerations can be repeated for the CM from PointNet++ [81]. However, a significant difference can be observed in the vehicle ground-truth row. Here, note how, compared with RadarPCNN, twice as many points are misclassified as negative samples – i.e. $160K$ vs $80K$. This dramatically reduces the recall score on the vehicle class, leading PointNet++ [81] to worse performance, as can be also noted in table 3.2. Finally, note that there exists some discrepancy between the CMs in fig. 3.7 and table 3.2 in terms of numerical results. This is due to the evaluation method, as the results reported in the table are averaged across 10 test iterations, while CMs are computed over a single evaluation.

3.4.2 Ablation Studies

The previous sections have proved the ability of RadarPCNN to efficiently process radar point clouds. Additionally, it has been possible to assess the validity of the

proposed pre-processing module, by plugging it to a PointNet++ architecture. The main objective of this section is to evaluate the effectiveness of the other proposed solutions. In particular, results collected over ablation studies are reported and analyzed to determine the impact of every solution on the network performance.

Fusion method. RadarPCNN uses two parallel branches to extract multiple object-related point-descriptors. Then, as described in section 3.2.3, it uses an attention mechanism to combine them into single point-signatures. However, this is not the only method capable to accomplish this task. In this section, the proposed attention mechanism is compared with two different approaches: addition and concatenation. The former sums-up the features learned for the same point, while the latter chains them together, producing a double-sized point-signature. Note how both approaches are less smart than the attention mechanism. Indeed, they do not involve any form of learning. Yet, all methods are well-suited to fuse multiple point-signatures. Table 3.3 contains the results of the experiment, where RadarPCNN has been trained and evaluated with the different fusion methods. The addition method proves to be the weakest solution among the configurations tested, showing the poorest semantic segmentation scores. This is attributed to the extreme simplicity of the method. Indeed, it performs a plain sum over the features, thus destroying potentially relevant information. Conversely, the concatenation approach shows very competitive results, achieving performance comparable to the attention mechanism on both classes. From a complexity point-of-view, addition proves to be the cheapest solution. The proposed attention mechanism follows very closely, using only $1.2K$ more parameters. Finally, note how the concatenation method significantly increases the network capacity. This is attributed to the increased dimensionality of its output. Indeed, in order to maintain all the information of the input vectors, it produces double-sized point signatures. As a consequence, the classification network requires more parameters/operations to reduce the dimensionality of the latent point cloud. Note that, since the operations introduced can be parallelized, the inference time is not affected in the same fashion – it increases by only 0.2 ms w.r.t. the addition method, though there are circa $30K$ more parameters. Yet, despite the increased capacity, this model configuration cannot outperform the original RadarPCNN implementation. Furthermore, note how the complexity increases even more severely when multiple point-signatures are fused by concatenation, thus undermining the ability to deploy this method. Conclusively, the results in table 3.3 prove that the attention mechanism is the best fusion solution for RadarPCNN, producing the highest F_1 scores with a reasonable amount of parameters. In particular, they suggest that RadarPCNN benefits from the filtering performed by the attention mechanism, which retains the most relevant information for the semantic segmentation task.

Sampling method. In order to improve the usage of radar spatial context information, RadarPCNN modifies the sampling process in the SA layers. Specifically, it replaces the standard farthest point sampling (FPS) algorithm with mean-shift (MS). This section proposes an investigation to assess the effectiveness of this solution. In particular, it provides a comparison on the performance of RadarPCNN using FPS and MS as sampling strategy. Several SA layer configurations are tested, varying the number of representative points used in the two altered PointNet++.

Table 3.3: RadarPCNN performance using different fusion methods (%). Inference time is computed on Nvidia GeForce RTX 2080 (ms).

	Moving Pedestrian			Moving Vehicle			Average				
Method	Prec.	Recall	F_1	Prec.	Recall	F_1	Prec.	Recall	F_1	Param.	Time
Concat	48.27	**76.50**	**59.19**	74.98	91.78	82.54	61.63	**84.14**	70.87	206.9K	25.7
Add	47.11	75.87	58.88	74.07	**92.13**	82.12	61.09	84.00	70.50	**174.1K**	**25.5**
Attention	**48.64**	75.82	**59.27**	**75.78**	91.44	**82.88**	**62.21**	83.63	**71.07**	175.3K	26.8

The bandwidth parameter, h in eq. (3.8), has been changed to achieve the desired amount of points with MS. FPS, instead, expects as input a parameter specifying the amount of points to sample, therefore, no further tuning of the method has been necessary. All model configurations have been trained from scratch and evaluated as specified in section 3.3.2. Figure 3.8 contains the results of the experiment, plotting the F_1 scores collected on the positive classes with four different model configurations. Macro-average scores are also plotted to provide insights about the overall performance. Details about the model configurations are reported on the abscissa axis. They specify the total number of representative points sampled in the two altered PointNet++ – i.e. top/bottom branch of fig. 3.2, respectively. For instance, 500/150 indicates the RadarPCNN configuration described in section 3.3.2, where one branch samples 500 points and the other 150.

The graph shows that MS (solid line) constantly outperforms FPS (dashed) in all the model configurations tested. In the best performing setting (500/150), the difference between MS and FPS proves to be relatively small, accounting for less than 1%, on average. This is attributed to the strong similarity between the two algorithms when used to sample a large amount of points. Indeed, as the the number of representative points increases, the methods have less degrees-of-freedom, thus resulting in similar outputs. This behavior becomes even more tangible when the two algorithms are forced to operate with singular settings. Bearing in mind that, in MS, a lower bandwidth produces more points, by setting this parameter to an infinitesimal value it is possible to generate a very localized kernel, as derived in eq. (3.10). Consequently, MS is forced to produce exactly the same output of FPS when asked to sample all the points in the input. Remarkably, however, MS shows very impressive results when the number of representative points decreases.

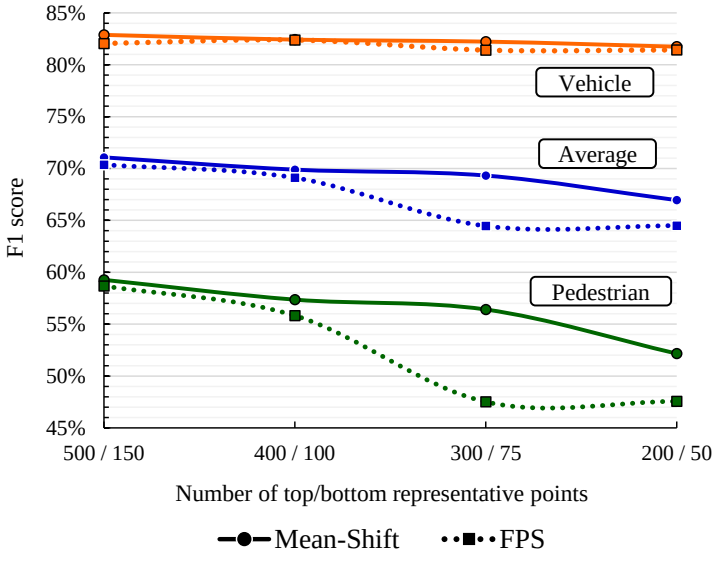
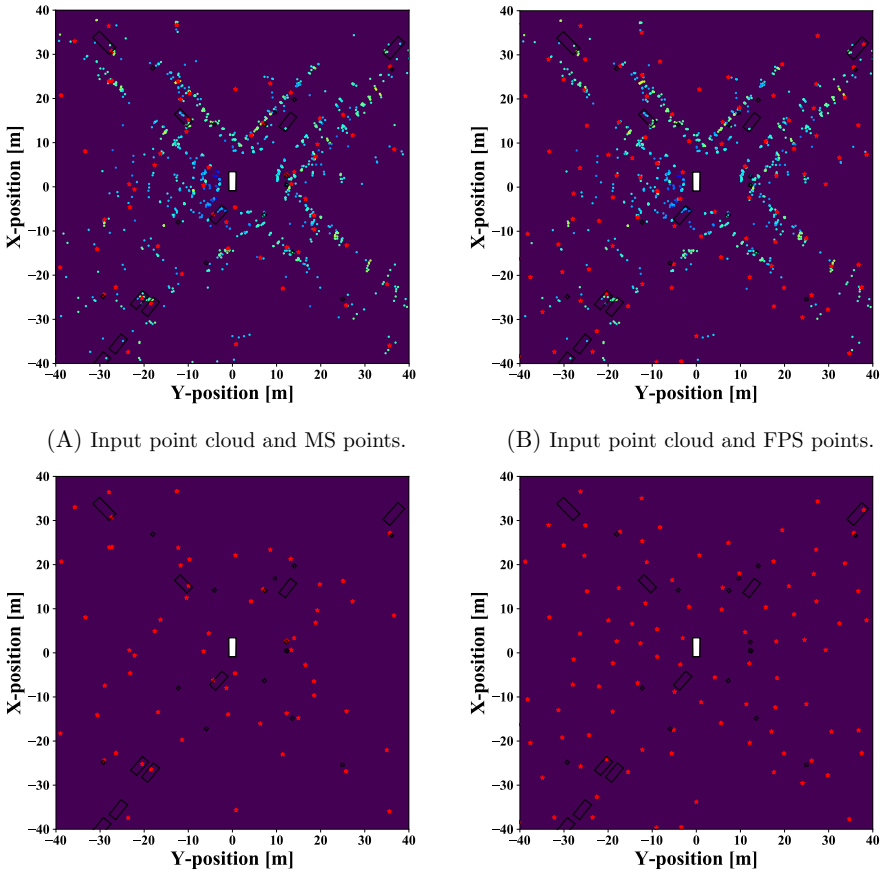


Figure 3.8: Comparison between MS and FPS under various configurations of the number of representative points. Best viewed in color.

In this configurations, it has a significant impact on the performance, limiting the drop caused by sparser sampling strategies. Note, indeed, how the solid blue curve remains almost flat until the 300/75 configuration: here, MS experiences a degradation lower than 2%, compared with the nearly 6% drop shown by FPS. This suggests that, by attending highly populated locations, MS optimizes the usage of spatial relationship in sparse radar data. Conversely, FPS proves to be better-suited when used in dense sampling strategies, due to its ability to uniformly cover the input space. Indeed, it shows good performance with high number of representative points, but misses relevant scene information when deployed in sparse sampling settings, thus inevitably leading to wrong point-class predictions. Similar observations can be collected by analyzing the curves of the positive classes separately. They show distinct behaviors when sparse sampling strategies are adopted: vehicles are hardly affected, while the performance on pedestrians is severely harmed. This is attributed to the considerably different sizes between instances of the two classes, resulting in diverse object-related point-density signatures in the input. Indeed, pedestrians are often represented by one or few reflections. Consequently, a sparser sampling strategy critically decreases the probability of selecting points for pedestrian instances. Though this consideration holds true for both methods, however, the ability of MS to respect the input data distribution allows it to consider small clusters, thus sampling representative points that describe the space around pedestrians. Contrarily, vehicles generally produces few dozens of reflections, especially when fast moving, Hence, they do not suffer from the same effect, resulting almost unaffected by changes in the sampling density. Finally, it is worth noting that the number of representative points strongly affects the model complexity. Therefore, the ability of MS to operate with sparse sampling settings represents a critical property, especially in resource-limited embedded systems.

To conclude the comparison between MS and FPS, it has been decided to examine their output and collect qualitative insights. Figure 3.9 contains an example of the representative points sampled with both MS and FPS on the same radar scene. FPS samples 150 points, while MS uses a bandwidth $h = 8.0$. Figures 3.9A and 3.9B display the radar point cloud used as input to the network (colored dots) as well as the representative points (red stars). Instead, figs. 3.9C and 3.9D showcase only the representative points. Note how FPS samples uniformly from the input: fig. 3.9D has points scattered around to cover most of the space, regardless of the shape present in the scene. Conversely, MS is more object-centered, as it provides points for almost every instance present in the scene (black boxes) – fig. 3.9C. Note how, despite generating more points, FPS cannot sample points for every object. In addition, note how the the output of MS results more clean than the one from FPS. Indeed, fig. 3.9C enables the visualization of the road boundaries, while in fig. 3.9D they are corrupted by the points evenly sampled. Further visualization results are provided in appendix A.1.

Grouping method. RadarPCNN adopts two altered PointNet++ networks to extract context information. Their encoder sections rely on SA layers, which perform three operations: sampling, grouping and feature extraction – as described in section 3.2.2. In this section different grouping techniques are investigated to gather insights about this process. In particular, the experiment proposed herein consists in tampering the the SA layers in RadarPCNN by changing the distance function



(A) Input point cloud and MS points.

(B) Input point cloud and FPS points.

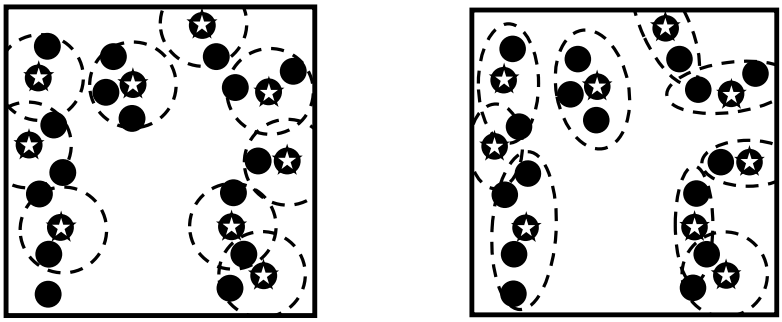
(C) MS representative points only.

(D) FPS representative points only.

Figure 3.9: Examples of representative points sampled with MS and FPS. Radar reflections are represented with points, colored based on their RCS value. Red stars show the space location sampled. Objects present in the scene are marked with black boxes. Best viewed in color.

$d(\cdot, \cdot)$ in eq. (3.2). The different model configurations are then trained and evaluated following the guidelines in section 3.3.2.

Figure 3.10 displays the grouping methods used in this experiment. The standard method proposed in [64] is used as benchmark. It consists in a spherical-based grouping process, gathering neighbors which lie in a sphere around the representative point. The Euclidean distance defined in eq. (3.12) is used to determine the boundaries of the search area. Note that, in this implementation, every spatial axis is considered equally important. Figure 3.10A shows a 2D example of the standard spherical-based grouping method, where circles represent the 2D projections of sphere boundaries. In order to evaluate the ability of the network to learn at the grouping stage, it has been decided to propose an advanced ellipsoidal-based grouping technique. It represents a generalization of the standard spherical-based approach, as it extends eq. (3.12) with the introduction of additional parameters.



(A) Spherical-based. (B) Ellipsoidal-based.

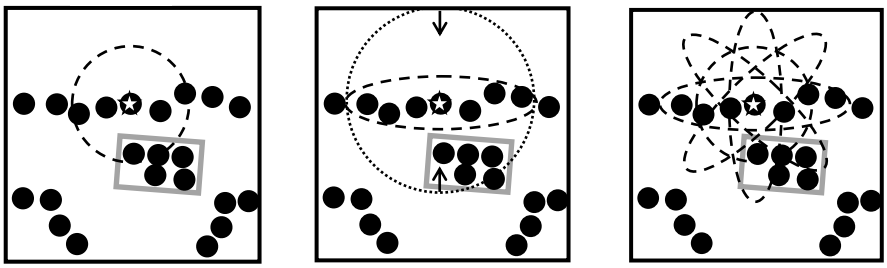
Figure 3.10: 2D examples of the proposed grouping techniques. Black-dots represent the data-points, while white stars indicate representative points. Dashed-shapes represent the neighborhood boundaries. Circles account for 2D projections of 3D spheres, while ellipses illustrate 2D projections of ellipsoids.

Specifically, it computes the distance as

$$d^{Ell}(\boldsymbol{x^i}, \boldsymbol{x^j}) = \sqrt{\alpha_e(x \cdot \cos\phi + y \cdot \sin\phi)^2 + \beta_e(x \cdot \sin\phi - y \cdot \cos\phi)^2 + \gamma_e z^2} \ , \quad (3.19)$$

where $\alpha_e \geq 0$, $\beta_e \geq 0$, $\gamma_e \geq 0$ and $\phi \in [0, \frac{\pi}{2}]$ are custom scalar parameters. The notation conventions in eq. (3.12) are adopted – i.e. x, y and z computed as the distance between the i-th and j-th point, along the corresponding axis. Note that eq. (3.19) describes the boundaries of a rotated 3D ellipsoid. In particular, the parameters α_e, β_e and γ_e allow scaling along the respective axis, while ϕ rotates the search space in the XY plane. It has been purposefully decided to prevent rotation along the z axis. This is because it is not expected a linear relation between point location and radial velocity – bear in mind that the input point clouds adopt the Doppler feature as z-coordinate. A side effect of this design choice consists in the reduction of the grouping process complexity. Figure 3.10B depicts a 2D example of the ellipsoidal-based method, where 3D ellipsoids are replaced with ellipses. Note how this technique enables the network to favor the aggregation of points along a specific, custom direction of the space. Moreover, note how eq. (3.12) represents a special case of eq. (3.19), where $\alpha_e = \beta_e = \gamma_e = 1$ and $\phi = 0$.

During this experiment, three model configurations are compared. Figure 3.11 contains details about the implementation of the associated grouping processes. One model uses the standard spherical-based grouping with the Euclidean distance in eq. (3.12) – fig. 3.11A. The other two use eq. (3.19) to set-up a dynamic and a static ellipsoid grouping method. The dynamic method defines the ellipsoid parameters α_e, β_e, γ_e and ϕ as trainable and regresses them using a NN. As depicted in fig. 3.11B, it first groups the neighboring points using the spherical-based method (dotted circle). Specifically, given the search range value r_i, it uses eq. (3.2) with $r = 1.5 \cdot r_i$ and the Euclidean distance function in eq. (3.12). Then, the local neighborhood is fed to a NN that estimates the values of the four parameters. Finally, it performs the grouping in eq. (3.2) with $r = r_i$ and the ellipsoid distance function in eq. (3.19), using the learned parameter-setup. This neighborhood (dashed ellipse) is the one used to encode local context information in the SA layer. To regress



(A) Spherical. (B) Dynamic Ellipsoid. (C) Static Ellipsoid.

Figure 3.11: 2D examples of the grouping configuration implemented. The same color coding of fig. 3.10 is adopted. The gray box represents a vehicle. The spherical method uses a simple sphere. The dynamic ellipsoid method has the ability to learn the parameters of the ellipsoid. The static method uses five different configurations and learns the best combination through an attention mechanism.

the parameter values, it is adopted a shallow PointNet architecture, followed by an FC-network. In more details, after computing 16 and 8 features for every point in the neighborhood, max-pooling is applied to obtain an 8-features-sized signature. Then, three FC-layers of size 8, 4 and 4 channels, respectively, are used to generate the parameter scores. The output layer uses a sigmoid activation function to constrain the predictions in the range $[0, 1]$. The score of the parameter ϕ is multiplied by $\frac{\pi}{2}$ to cover the whole valid angular domain, while the scaling parameters α_e, β_e and γ_e are limited by a lower bound of $0.1 \cdot r_i$ to avoid values too close to 0. Note that the parameter values are computed based on the neighboring point distribution. Therefore, this grouping method enables the network to learn a search area shape that allows aggregation of relevant points, filtering out potential outliers. For instance, consider the example in fig. 3.11B, where the representative point (white star) belongs to a wall. The dynamic ellipsoid grouping method enables the network to regress the flat ellipse which contains only wall-points, discarding points from the nearby vehicle (gray box). In this way, the SA layer can learn a descriptor for the wall which is unaffected by vehicle-points. The radius of the preliminary spherical grouping has been set to $r = 1.5 \cdot r_i$ in order to use a broader local area for the parameter estimation task. Finally, it is worth stressing out that each representative point has its own parameters, regressed based on the local point distribution. The static method, instead, uses multiple settings of the ellipsoid parameters. Specifically, it uses five search area configurations. One uses the Euclidean distance in eq. (3.12) – i.e. $\alpha_e = \beta_e = 1$, $\phi = 0$. The remaining four adopt ellipsoids defined by $\alpha_e = 0.5$ and $\beta_e = 2$, rotated by $\frac{\pi}{4}$ each – i.e. $\phi \in \{0, \frac{\pi}{4}, \frac{\pi}{2}, \frac{3\pi}{4}\}$. It has been decided to not apply any scaling along the Z axis – i.e. $\gamma = 1$. Figure 3.11C depicts the 2D projection of the search areas. Note how a rotation of $\frac{\pi}{2}$ is tantamount to switch the value of the parameters α_e and β_e. Therefore, the ellipse described by the configuration $\alpha_e = 0.5$, $\beta_e = 2$, $\phi = \frac{3\pi}{4}$ coincides with one generated by $\alpha_e = 2$, $\beta_e = 0.5$, $\phi = \frac{\pi}{4}$ – where ϕ is in the valid range interval $[0, \frac{\pi}{2}]$. The five local neighborhoods are then fed to the feature extraction module to encode context-related information. Finally, an attention mechanism, similar to the one described in section 3.2.3, is used to combine the features learned in every configuration. The same settings used in RadarPCNN are

adopted for the attention network, though five signatures have to be fused together instead of two. This method, similarly to the dynamic one, has the ability to learn the search area based on the point distribution. Indeed, the attention mechanism enables the network to select the configuration which results in the most informative descriptor. Yet, it is constrained to a fixed set of configurations.

Table 3.4 contains the results collected from this experiment. The standard spherical grouping method proves superior semantic segmentation performance, significantly outperforming the two ellipsoid-based methods in both positive classes. This is attributed to two different reasons: one connected to the network operations and the other related to the back-propagation gradient flow. Note that RadarPCNN uses SA-layers with PointNet as feature extractor. This module performs its task in two steps. Shared FC-layers learn new signatures for every point in the neighborhood. Then, max-pooling filters this information by retaining, for each feature, the maximum value across all points in the neighborhood. In this way, only the points which produce maximum feature-wise values provide a contribution to the representative point signature. Consequently, the network has a way for selecting the most relevant neighboring points. In addition, note that the grouping operation is a mere selection process, which does not involve any transformation of the input. Therefore, there is no direct gradient signal flowing in, thus hindering the optimization process. This latter observation is confirmed by the results of model using the dynamic ellipsoid method. It uses the loss function to estimate the best ellipsoid parameters configuration. However, it achieves very poor performance, thus indicating that it cannot properly solve the parameter estimation task. The results of this experiment suggests that RadarPCNN does not need a perfect grouping process, since it adopts other procedures to filter eventual neighboring outliers. For this reason, it has been decided to adopt the standard spherical-based method to perform the grouping procedure. Finally note how, despite leading to worse semantic segmentation performance, the models using ellipsoid-based grouping show higher complexity than the spherical one.

Doppler experiments. Radar point clouds contain information about object-related properties. Doppler represents the most valuable measurement, providing information about the velocity profile of a target. This section is dedicated to an investigation on the network usage of radar velocity measurements. The ability to measure velocity is a very interesting radar property. Doppler is computed exploiting the frequency shift caused by the difference in velocity between sensor and target. More precisely, it represents a measure of the relative radial object speed. Since every radar reflection possesses a Doppler measurement, the most simple practice consists in using it as additional point feature. However, providing the network with relative velocity information might be misleading. Therefore, a different approach consists in using the absolute radial velocity. This can be computed by compensating the sensor-motion – i.e. subtracting the radial component of the ego-vehicle velocity to the measured Doppler. There are several ways to perform this task. In this work, odometry data are used to estimate the ego-vehicle velocity. As a result, every radar reflection is provided with an ego-compensated Doppler measurement, thus well-suited to be used as point feature. Furthermore, note how an object is strongly characterized by its velocity, differentiating it from other nearby objects. For this reason, another way of using the Doppler information is as z spatial point coordinate

Table 3.4: RadarPCNN performance using different grouping methods (%). Inference time is computed on Nvidia GeForce RTX 2080 (ms).

	Moving Pedestrian			Moving Vehicle			Average				
Method	Prec.	Recall	F_1	Prec.	Recall	F_1	Prec.	Recall	F_1	Param.	Time
Dynamic Ellipsoid	44.70	71.94	55.14	66.97	89.68	76.64	55.83	80.81	65.89	191.6K	28.1
Static Ellipsoid	**48.92**	70.00	57.25	71.81	90.09	79.92	60.36	80.04	68.58	204.7K	27.7
Spherical	48.64	**75.82**	**59.27**	**75.78**	**91.44**	**82.88**	**62.21**	**83.63**	**71.07**	**175.3K**	**26.8**

– bear in mind that the sensors used herein provide poor elevation resolution.

This section proposes an analysis on the performance of RadarPCNN while using the Doppler feature in different configurations. In this way, it is possible to gather insights about the impact of this feature on the network. Four models have been evaluated, using both relative and ego-compensated Doppler either as point-feature or z-coordinate. Table 3.5 contains the result of the experiment. Note how the network performance benefit from absolute velocity measurements. In both its usage configurations, the ego-compensated Doppler introduces significant improvements, increasing of circa 10% the F_1 score produced by the uncompensated Doppler feature. This effect is not surprising. Indeed, consider an object that moves at the same speed of the ego-vehicle. It produces radar reflections with Doppler holding null velocity measurements. As a consequence, the network is not able to detect that the object is moving, as it has no information about the ego-velocity. Interestingly, both classes benefit from usage of the absolute velocity in similar ways. Concerning the usage of the Doppler feature, incorporation in the point-coordinates proves to yield advantages over its adoption as point-feature. This effect is attributed to the additional spatial separation, introduced by the Doppler coordinate, between points of objects with different speed. Indeed, consider, for instance, a vehicle driving-by a parked car. Although these two instances are close to each other in the XY plane, they possess completely different velocity profiles. Therefore, by setting the Doppler as z-coordinate, it is possible to force an additional physical separation between points from the two different objects. Considering the internal operations of RadarPCNN, there are two processes that can take advantage of this setup. Indeed, it enables the sampling process to sample representative points for close-by objects with different speed. At the same time, it helps the grouping process to produce cleaner clusters, reducing the probability of finding points from objects with diverse velocity profiles in the same group. Furthermore, note how pedestrians result more affected by this configuration, experiencing $2\times$ the improvement that the vehicle class does. This is attributed to the nature of the objects of this class. Indeed, despite pedestrians have a weaker velocity signature than vehicles, they are often found near stationary objects, such as building walls. Therefore, there are more cases in which the spatial separation introduced by the Doppler feature plays a fundamental role. Finally, note how the model using the ego-compensated Doppler as z-coordinate is able to achieve superior performance, combining the benefits of both configurations. For this reason, it has been decided to use this setup in RadarPCNN.

3.4.3 Visual Analysis

Visual investigations are very useful for gathering fine details about the network operations. They provide qualitative results, enabling the identification of corner-case situations which affect the algorithm output. This section reports insights collected from a visual examination of RadarPCNN predictions on test-set samples. Figure 3.12 displays an example. BBs indicate the presence of a ground-truth object, while points, colored based on the network prediction, represent the input radar point cloud – see caption for further details about color-coding. The ego-vehicle is moving in an urban scenario, with pedestrians on the curbstones (red BBs on left and right), moving vehicles (yellow BBs in front and rear), standing vehicles at a traffic light (gray BBs in rear) and a slow moving vehicle that is starting its motion

Table 3.5: RadarPCNN performance under different usage of the Doppler feature (%).

Method	Usage	Moving Pedestrian			Moving Vehicle			Average		
		Prec.	Recall	F_1	Prec.	Recall	F_1	Prec.	Recall	F_1
Doppler	Feat	37.15	63.74	46.94	65.81	79.08	71.84	51.84	71.41	59.39
Doppler	Coord	42.97	70.67	53.37	69.65	79.89	74.42	56.31	75.28	63.89
Comp. Doppler	Feat	46.96	71.03	56.54	74.16	91.13	81.62	60.55	81.08	69.08
Comp. Doppler	Coord	**48.64**	**75.82**	**59.27**	**75.78**	**91.44**	**82.88**	**62.21**	**83.63**	**71.07**

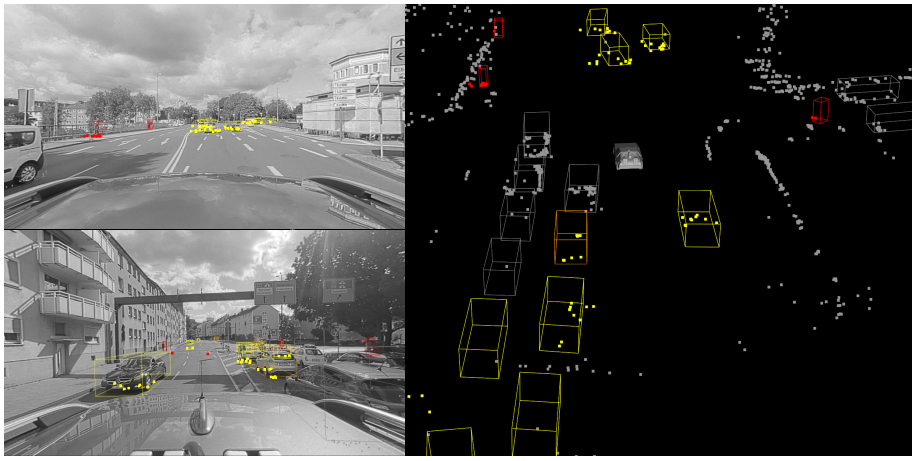


Figure 3.12: RadarPCNN predictions. Yellow/red points are moving vehicle/pedestrian predictions. Ground-truth BBs follow the same color-map. Gray BBs belong to stationary vehicles, while orange BBs mark vehicles with absolute speed between 1 and 2.5 m/s. The z-component is zeroed and aligned with the road for visualization clarity. Left. Front/rear camera view. Right. 3D visualization. Best viewed in color.

(orange BB in rear). Note how the network is able to correctly classify most of the points, detecting moving instances of both pedestrian and vehicle classes. Moreover, note the yellow points just outside the BBs of the three cars in front of the ego-vehicle. They are still classified as moving vehicles. However, these are not wrong predictions. Indeed, from the camera view, it is possible to notice that there is no other object in their neighborhood. Therefore, they are considered radar-related artifacts, possibly the result of multi-path propagation. Finally, it is worth noting the yellow points inside the orange BB. Remarkably, RadarPCNN recognizes that these reflections belong to a moving vehicle, though they are not labeled as moving – because the object's speed is below the designed threshold. This suggests that the network's concept of a moving instance is not strictly correlated to the threshold values set during training. Yet, the gray points inside the gray BBs prove that the network is able to distinguish static from dynamic instances. Further visualization results containing RadarPCNN predictions are provided in appendix A.2.

Case of study: bushes as pedestrians. The visual analysis conducted highlighted the ability of the network to generalize to the proposed semantic segmentation task. In addition, it revealed the source of a systematic error: sometimes the pedestrian class is mistakenly predicted for bushes reflections. Figure 3.13 contains an example from the test-set. The ego-vehicle is standing after a traffic light which has just turned green. Note that RadarPCNN can predict the correct class for most of the points, including the pedestrian on the left and the slow moving vehicles in front. Yet, there is a small cluster of red points on the right of the ego-vehicle, without any nearby BB. From the top-left image (right-facing camera), it is possible to notice the presence of bushes around the red points. To assess whether this is an algorithm-specific issue, the same visual analysis has been repeated for predic-

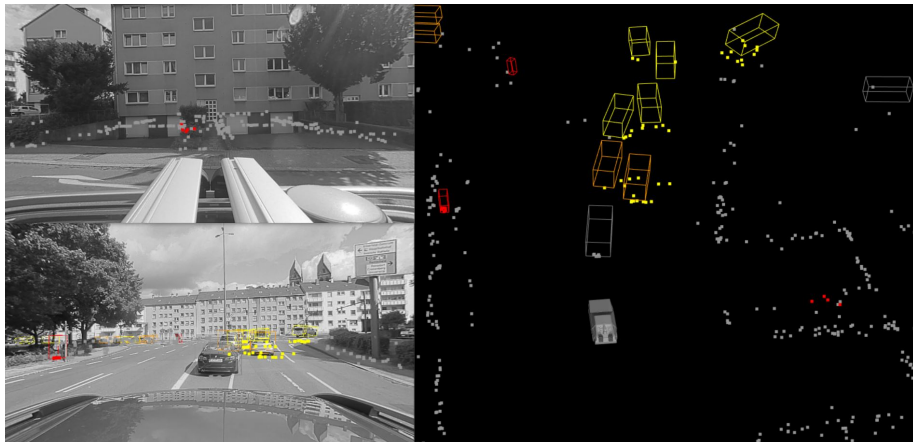


Figure 3.13: RadarPCNN predictions. Bushes points on the right of the ego-vehicle predicted as pedestrian. The same color-coding as in fig. 3.12 is adopted. Left. Right/front camera view. Right. 3D visualization. Best viewed in color.

tions from PointNet++ [81]. However, it has been observed that both networks are affected by similar error sources, thus suggesting that they are not related to the algorithm. Another theory assumes that these errors are sensor-related. Indeed, it has been noted that bushes have different moving parts which produce fluctuations in the Doppler measurements. Other studies have shown that this effect can be found in radar measurements of pedestrians [78]. Therefore, it has been proposed to attribute the prediction errors to a strong radar similarity between instances of these two classes.

In order to prove this theory, an additional experiment has been performed. It has been decided to compare the Doppler distributions generated by different classes of ground-truth objects: i.e. misclassified bushes, correctly classified stationaries and GT pedestrians. In more detail, first, it has been collected a set of 200 bushes points from the test-set mistakenly predicted as pedestrians. Then, the ego-compensated Doppler distribution is computed from the values associated to the points in the set. The same operations are repeated for reflections from stationary object correctly classified as the negative class – buildings and parked vehicles points are used. Figure 3.14 plots the results of the experiment. Note that, due to the low amount of samples, the Doppler distributions are plotted as histograms. The Doppler distribution of ground-truth pedestrian points (black dotted line) is reported for reference – all pedestrian points in the test-set are used. It is worth noting how stationary objects exhibit a very narrow Doppler spectrum: nearly 90% of the collected points assume a Doppler value in the interval $[-0.2, 0.2]$ m/s. Contrarily, despite presenting a major concentration around the stationary Doppler value, bushes show a wider spectrum, with values scattered from -1.6 to $+1$ m/s. Notably, their histogram has a strong resemblance with the pedestrian distribution, thus confirming the Doppler similarity between these two classes. Furthermore, note how bushes and pedestrians share similar spatial properties. Indeed, they can often be found nearby static objects – like the building in fig. 3.13. Consequently, it is possible to attribute the reason of the confusion between bushes and pedestrians to

the radar perspective of the two objects, which sometimes does not provide enough information to differentiate each other. However, this aspect does not produce a major effect, as most of the times RadarPCNN is able to correctly classify bushes as the negative class, as proved by the gray points from the hedge on the front-right of the ego-vehicle in fig. 3.13. Further visual examples about wrong pedestrian predictions of bush points are provided in appendix A.3.

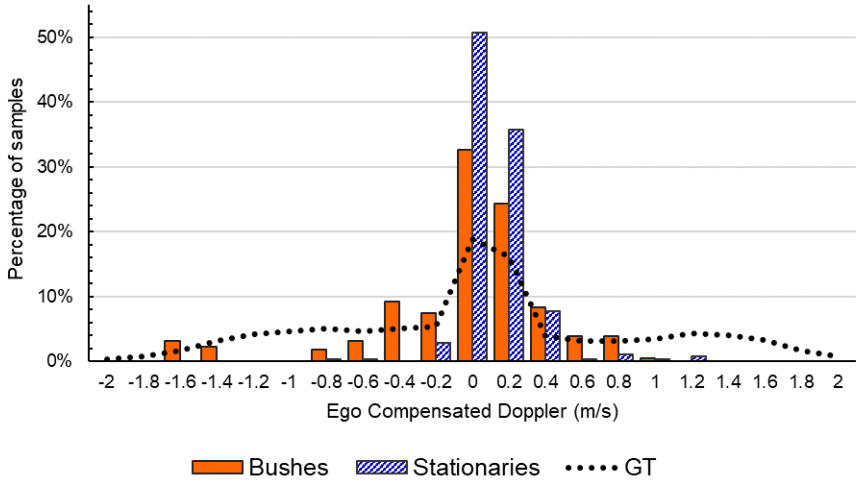


Figure 3.14: Histogram of Doppler values from bushes points falsely predicted as pedestrian (orange bars) and stationary points correctly predicted (blue hatched bars). The black dotted line shows as reference the distribution of Doppler values from GT pedestrian points. Best viewed in color.

Chapter 4

A Survey on the usage of Radar Features in Pointnets for Pedestrian Detection

The previous chapter proves that the performance of point-wise processing techniques on radar data can be improved by leveraging intrinsic properties of the sensor. In this way, it is possible to achieve reliable detection of nearby objects, such as vehicles and pedestrians, thus enabling environment perception solutions based on radar. However, in an autonomous vehicle scenario, there are specific targets whose recognition is of upmost importance. An example is represented by vulnerable road user (VRU) objects, of which pedestrians are the most common instance. Members of this group are particularly exposed to faulty operations of autonomous vehicle systems. In addition, their small sizes limit the ability of the sensor to observe them, thus hardening the recognition task. Consequently, it is critical to study and assess the decision process which leads to the detection of these instances.

In this chapter, it is provided a focus on the ability of point-wise processing methods to detect pedestrians. In particular, it has been targeted the goal of establishing how these methods use the information contained in the radar features. The first section introduces the task of pedestrian recognition with automotive radar sensors. Section 4.2 describes the role played by the algorithm during the investigation and provides an overview of the models used in this work. Section 4.3 reports details about the experimental setup, such as dataset, network configurations and the procedure used to conduct the experiments. Finally, the results of the investigation are provided, complemented with observations and comments.

The following own contributions are provided in this chapter:

- Establishment of Doppler as the most relevant radar feature for the classification of pedestrians.
- Assessment of the usage of the radar features in various point-based methods.
- Assessment of the ability of PointNet++ to approximate the GT pedestrian Doppler distribution.
- Assessment of the role played in PointNet++ by the Doppler feature and the spatial coordinates.

The results discussed in this chapter were previously published in [121].

4.1 Radar-based Pedestrian Detection

A key factor towards the acceptance of autonomous driving vehicles relies in their effectiveness [70]. Among the various tasks that such an advanced system has to execute, a crucial one consists in the collection of knowledge about the surrounding environment. Indeed, it is of paramount importance to have information about the road boundaries as well as the objects present in the scene. A special class of objects in an autonomous driving scenario is VRUs. Members of this class share the characteristic of being particularly endangered by potential malfunctioning of the system. Therefore, the ability to detect and locate the presence of these objects is critical for ensuring safe operation. Examples of VRU instances are pedestrians, cyclists, and motor-cyclists. Note how they also share the characteristic of being small compared to other objects in the scene – e.g. vehicles. Since smaller sizes often result in fewer measurements, regardless of the sensing technology, this aspect further complicates the detection task.

A commonly adopted sensor in the automotive industry is radar. In the last decades, it has been progressively more involved, enabling cutting-edge applications like guard-rail detection [22] and ACC [17]. Yet, the recognition of VRUs with radar remains an open challenge. Among various objects, the task of pedestrian detection has particularly captured the interest of the research community. Several researchers have approached the problem by examining the radar measurements produced by pedestrian instances [24, 31]. Others have used these observations to build a pedestrian recognition system [32, 77].

Ritter and Rohling [24] noted that, compared to other objects, pedestrians have a heterogeneous reflective surface. They proposed to distinguish two parts with different properties. The HAT (head, arm, trunk) area reflects the radar signal with high-amplitude and low variance in Doppler. The lokomotor (legs) section, instead, results in signal with lower energy and wide Doppler spectrum – due to smaller dimensions and a relative velocity component. In addition, the authors compared the radar signals generated by both pedestrian and vehicle objects moving in the radial direction of the sensor. From this experiment, they found out that pedestrians produce a broader Doppler spectrum. Consequently, they proposed to use the radar spectral signature to characterize a pedestrian.

Bartsch et al. addressed the binary classification task between pedestrians and static objects [31]. They developed a very simple recognition algorithm to allow backtracking of the errors from the output to the raw data. The resulting algorithm is based on observations. Beyond the properties found in [24], they noted that pedestrians are characterized by smaller sizes than most static objects – e.g. parked cars. Therefore, they grouped together the sensor data belonging to the same physical object and extracted 5 hand-crafted feature to represent the cluster. Two of them based on the shape/size of the cluster; the other three based on the Doppler spectrum. Since they found out that the radar intensity of signals reflected by pedestrians is highly fluctuative, they decided not to use this feature. The classification algorithm works by setting membership functions. The extracted features are matched with them and the scores are summed up to produce the final classification output. The experiments suggested that the ego-velocity compensation

plays a major role for distinguishing pedestrians from static objects. Furthermore, it turned out that the Doppler spectrum contains the most relevant information for the recognition of pedestrians. Yet, due to its simplicity, the algorithm could not detect pedestrians with low Doppler values.

Heuel and Rohling targeted the task of distinguishing pedestrians from vehicles [32]. Similarly to [31], they noted that it is possible to use size and velocity to characterize objects from the two classes. In particular, they observed that pedestrians have large Doppler profiles – due to the micro Doppler components produced by the various moving parts – combined with narrow range profiles – small sizes. Contrarily, vehicles show small Doppler variance – the body moves as a whole – together with large range profiles. In addition, they observed that vehicles produce reflections with higher RCS values than pedestrians – due to the metallic reflective surface. Therefore, the authors developed a classification system that exploits these class-related properties. Specifically, they first grouped together points belonging to the same physical instance. Then, they extracted eight hand-crafted features to represent the cluster: four Doppler-related, three range-related and the RCS. Finally, they fed the cluster signatures to an SVM model [15] to predict a class for the group of points. The system resulted effective for differentiating pedestrians from longitudinal moving vehicles. However, it often classified laterally moving vehicles as pedestrians. For this reason, the authors leveraged the Doppler measured for the same object to estimate longitudinal (radial) and lateral (tangential) velocity components. They extracted three new features and retrained the SVM model. They proved that the system benefits from the new information introduced, showing improved classification performance and lower confusion rates.

Prophet et al. proposed a system for the detection of pedestrians [77]. The authors used DBSCAN [16] to cluster together points of the same object. Then, they extracted cluster-related signatures, extending the feature-set proposed in [32]. Finally, they used a classification model to estimate the output class. Several different algorithms were evaluated, including decision tree [6], SVM, K-nearest-neighbors (KNN) [4] and NN. All the classifiers proved to be effective in the detection of pedestrian clusters, thus suggesting that the proposed feature-set discriminates pedestrians from other objects. Furthermore, the authors showed that, though pedestrians share similar statistics with other VRU objects, it is possible to differentiate between them. Indeed, cyclists have higher reflected power, due to the metallic structure of the bicycle. Contrarily, dogs exhibit lower reflected power, higher range profiles and different velocity signatures – due to the four moving legs.

The recognition of DL as a tool for solving highly-complicated tasks has produced a progressive adoption of this technology in the automotive industry [117, 95]. It advanced the state-of-the-art of vehicle perception systems, enabling the effective recognition of VRUs [98]. Researchers have proved the ability of DL to process radar data. For instance, Wöhler et al. showed that an LSTM network can distinguish pedestrian points from other object classes like bike and vehicle [67]. However, as discussed in section 2.3.2, the introduction of point-wise methods [65, 64, 74] – a new class of NNs – has enabled efficient processing of point clouds. By generalizing these approaches to radar data, it has been possible to achieve state-of-the-art performance [93, 81]. For instance, the authors in [81] proved that PointNet++ [64] can effectively detect pedestrians from raw radar point clouds. However, though these architectures significantly outperform classical radar processing techniques, they provide poor

interpretability. This property, characteristic of NN algorithms, makes it difficult to determine a causality between input and output.

4.2 Radar Features in Point-wise Techniques

Previous works use simple, human-interpretable solutions to study the ability of detecting objects from radar data. In this way, it is possible to easily trace prediction errors back into the input domain and, consequently, infer the role played by every input feature. However, despite this strategy can be used to understand which feature contains the most critical information, it does not allow one to fully determine the real correlation between input features and target task. Indeed, every algorithm has its own set of rules, defining a unique usage of the input features. For instance, consider a model which uses only the single-point radar features to detect pedestrians in an urban environment. Based on this algorithm, the spatial position of the points is irrelevant for the task at hand – as pedestrians are not confined in a specific spatial location. However, pedestrians are usually located near static objects like parked cars or buildings. Therefore, by using human-interpretable solutions, it is possible to back-trace only errors related to the human-interpretation of the task.

It is believed that, in order to perform a comprehensive analysis, it is paramount to involve the best algorithm for the given task. Herein, it is reported an investigation about the usage of radar features for the task of pedestrian detection with point clouds. Point-wise processing approaches represent the current state-of-the-art in this field. They perform the pedestrian detection task by addressing a semantic segmentation problem. Hence, they localize a pedestrian in the scene by predicting the proper class for its radar reflections. However, pedestrians are severely underrepresented in radar point clouds. Indeed, compared to other instances, such as vehicles, they provide a lower amount of point-measurements and miss a strong radar signature. In this work, instead of focusing on the question "How is it possible to detect pedestrians in radar point clouds?", the following questions are asked:

- Why point-wise techniques are surprisingly good at pedestrian detection?
- How do they use radar features?
- Which are the most relevant features for this algorithm?
- How do other techniques use the radar features?

Since PointNet++ [64] represents the best point-wise processing architecture tested on radar point clouds, it has been selected as benchmark. In order to gather more insights about the usage of radar features in this framework, two supplementary point-wise processing networks have been implemented: PointNet [65] and Shared FC. They perform only part of the processing operations of PointNet++, thus enabling one to attribute a particular usage of the features to a specific module of the framework. Finally, a random forest (RF) model is evaluated, to obtain a comparison with a solution not based on NN technology. In this way, it is possible to assess eventual usages of the radar features specific of NNs. The next sections contain a detailed descriptions of the selected models.

4.2.1 PointNet++

It represents the state-of-the-art architecture for consuming raw point clouds, extending the CNN concept of locally-shared, hierarchical processing to point-wise techniques. The resulting framework adopts an encoder-decoder structure. The encoder section builds a sparser point cloud to hold the information extracted from local patches of the input. It uses set-abstraction (SA) layers as building-blocks to perform three operations: sampling, grouping and feature extraction. Figure 2.10 shows a graphical example of the processes in the SA layer. The sampling process selects a subset of the input points – using the farthest point sampling (FPS) algorithm. The grouping process collects neighborhood patches around every sampled point, clustering together points from the input point cloud. The Euclidean distance is used to define the neighborhood area. Finally, the feature extraction process leverages the information held by points in the patches to compute signatures describing the local regions. PointNet is adopted to perform this operation and the global signature is used as local descriptor. The decoder section of the architecture, instead, propagates the local information held by the sparser point cloud into the original point locations. It adopts feature-propagation (FP) layers to accomplish this operation. They perform an inverse-distance weighted interpolation to spread the information of the sparser point cloud into the point locations of the denser point cloud. Then, they leverage the interpolated features as well as the local signature extracted for the same point in the encoder section to compute new point-descriptors. Skip-connections are used to couple every FP layer with the corresponding SA layer. Finally, the point-signatures produced by the last decoder layer are processed with a shared FC classification network to obtain a class score for every point – as in eq. (3.15). An exhaustive description of the PointNet++ framework, including formal derivation, is provided in section 3.2.2 – eqs. (3.2)–(3.6). Further details about PointNet, instead, are reported in section 4.2.2.

4.2.2 PointNet

It is the predecessor of PointNet++. Among point-wise processing techniques, it represents a milestone, as it is the first network purposefully designed to consume raw point clouds. Figure 2.9 illustrates the processing scheme of PointNet. It relies on two main components: shared FC-layers and max-pooling. The former perform an order invariant processing by using the same kernel of weights to extract features for all the points. Formally, given a point cloud $\boldsymbol{\mathcal{X}}$ with points $\boldsymbol{x^i} \in \mathbb{R}^m$ – where m is the number of features – a single shared FC-layer computes, for every point, a new signature

$$\boldsymbol{x_s^i} = f_s(\boldsymbol{x^i}; \boldsymbol{W_s}, \boldsymbol{b_s}) = \mathrm{ReLU}\big(\boldsymbol{W_s}^T \boldsymbol{x^i} + \boldsymbol{b_s}\big) \ , \tag{4.1}$$

where $\boldsymbol{W_s}$ and $\boldsymbol{b_s}$ are trainable parameters – weight and bias, respectively – and ReLU is used as activation function – i.e. $\mathrm{ReLU}(x) = \max(x, 0)$. Note how this operation shares the same parameter-set across all the points. By stacking several shared FC-layers on top of each other, it is possible to achieve a deep processing and produce a point cloud $\boldsymbol{\mathcal{X}_{FC}}$ with enriched point-related feature-descriptors. The max-pooling operator, instead, extracts a global signature, containing information about the whole point cloud. It is applied on the latent point cloud produced by the shared FC-layers. Formally, given $\boldsymbol{\mathcal{X}_{FC}}$ with n points and m' features and the

associated matrix $\boldsymbol{X_{FC}} \in \mathbb{R}^{m' \times n}$ – whose columns contains the point-features –, it computes the global descriptor $\boldsymbol{x_g} \in \mathbb{R}^{m'}$ as

$$\boldsymbol{x_g} = \text{MaxPool}(\boldsymbol{X_{FC}}) = \begin{bmatrix} \max_i \{x_1^i \quad \forall\, i \in [1..n]\} \\ \vdots \\ \max_i \{x_{m'}^i \quad \forall\, i \in [1..n]\} \end{bmatrix} \tag{4.2}$$

where x_j^i is the j-th feature of the i-th point in $\boldsymbol{\mathcal{X}_{FC}}$ – i.e. the value contained in the i-th column and j-th row of $\boldsymbol{X_{FC}}$. Therefore, for each feature, this operation collects the value from every point and retains only the contribution of the point with the maximum value. The global signature can be used as it is to perform classification tasks, where the target is to predict a single class for the whole point cloud – e.g. scene recognition. When used for semantic segmentation tasks, instead, PointNet broadcasts the global signature to every input point. Consequently, for each point, it provides as output a signature $\boldsymbol{x_{PtN}^i} \in \mathbb{R}^{2m'}$, such that

$$\boldsymbol{x_{PtN}^i} = \left[\boldsymbol{x_{FC}^i}^T, \boldsymbol{x_g}^T\right]^T \quad \forall \quad \boldsymbol{x_{FC}^i} \in \boldsymbol{\mathcal{X}_{FC}} \ , \tag{4.3}$$

containing a set of point-specific features – $\boldsymbol{x_{FC}^i}$ – and a set of features shared across all points – $\boldsymbol{x_g}$ –, holding context information. This signatures can then be processed with a shared FC classification network to compute the output class scores, as shown in eq. (3.15). Furthermore, note that the original implementation of PointNet in [65] includes the usage of a spatial transformer network to achieve invariance w.r.t different perspectives – by rotation and translation of the input. However, in autonomous vehicle applications, the point cloud is always centered around the ego-vehicle. Therefore, this operation is skipped in the network proposed for the experiments. Finally, note how PointNet differs from PointNet++ in that it enriches the point-signatures with global rather than local context information. Consequently, it allows one to collect insights about the ability of extracting global and local information from the radar features.

4.2.3 Shared FC

It consists in the PointNet architecture, without the max-pooling operation. This network processes the input point cloud by using only a collection of shared FC-layers. Formally, for every input point $\boldsymbol{x_i} \in \boldsymbol{\mathcal{X}}$, it produces a point-signature

$$\boldsymbol{x_{sFC}^i} = f_{s_3}\Big(f_{s_2}\Big(f_{s_1}(\boldsymbol{x^i}; \boldsymbol{W_{s_1}}, \boldsymbol{b_{s_1}}); \boldsymbol{W_{s_2}}, \boldsymbol{b_{s_2}}\Big); \boldsymbol{W_{s_3}}, \boldsymbol{b_{s_3}}\Big) \ , \tag{4.4}$$

by iteratively applying eq. (4.1). Although eq. (4.4) uses exactly three FC layers – i.e. f_{s_1}, f_{s_2} and f_{s_3} –, the network is not limited to this configuration. Note how the signature computed for every point is independent from other points present in the cloud. Consequently, the network relies on the radar point-measurements to infer the single-point class, without exploiting any context information. Furthermore, note how the kernel-weights are shared across all points, as they are independent from the point-index i. Finally, in order to perform the semantic segmentation task, the point-signatures $\boldsymbol{x_{sFC}^i}$ are processed with a shared FC classification network, as shown in eq. (3.15). Though Shared FC consists in a very simple point-wise processing technique, it allows one to gather insights about the importance of the global descriptor in PointNet for specific usages of the radar features.

4.2.4 Random Forest

It is another supervised ML method [20]. However, differently than the previous approaches, it does not have any trainable weights, but relies on an advanced thresholding scheme. RFs are a family of ensemble-based learning algorithms that leverage a collections of decision tree models [6, 8] to perform the task. By averaging the predictions of multiple deep decision trees, indeed, RFs make the method more robust to noise in the train-set.

Decision trees use a labeled dataset to learn how to separate samples from different classes based on the input features. They owe their name to the processing diagram used to formulate their predictions, resembling a family tree. Herein, only non-categorical classification trees[1] are considered, as categorical and regression trees are out-of-the-scope of this work. The building blocks of decision trees are nodes, branches and leaves. Nodes select a feature from the input and compare its value with a learned threshold. They perform a binary classification task, steering the input into one of their output branches. Branches, in turn, have the task of connecting two nodes at different layers, providing the lower-layer node output as input to the node in the next layer. Finally, leaves are special nodes with no output branches. They are associated with a given outcome and determine the model output prediction. In a classification tree, every leaf is associated with a class prediction.

Decision trees work in a top-down fashion. At inference time, the input is fed to the first layer, containing a single node. It is then passed to a node in the next layer until it reaches a leaf node. Figure 4.1 contains as example of decision tree diagram with 4 layers and 8 leaves. During the training procedure, the model sets its parameters, such as tree-structure and nodes configuration. It starts with the first node. Given as input the train-set $\boldsymbol{\mathcal{X}}$ with samples $\boldsymbol{x^i} \in \mathbb{R}^m$ and associated labels $y^i \in [1..d]$, it produces two output sub-sets $\boldsymbol{\mathcal{X}_j^0} = \{(\boldsymbol{x^i}, y^i) \in \boldsymbol{\mathcal{X}} \mid x_j^i < \gamma_j\}$ and $\boldsymbol{\mathcal{X}_j^1} = \boldsymbol{\mathcal{X}} \setminus \boldsymbol{\mathcal{X}_0}$, where x_j^i is the j-th feature of the i-th sample $\boldsymbol{x}^i$, γ_j the node

[1]Where the input features have numerical values instead of categories and the output is a classification task.

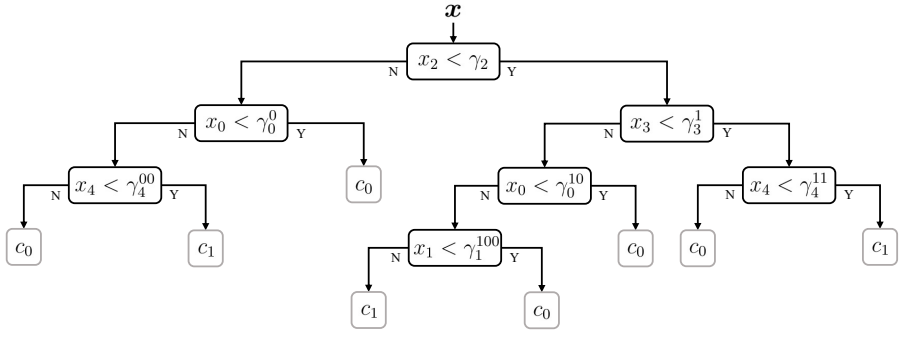


Figure 4.1: Example of decision tree diagram for a binary classification task. Black boxes denote computational nodes, while gray boxes show leave nodes. x_i refers to the i-th feature of $\boldsymbol{x}$, while γ_i represents the corresponding threshold. Since the same input feature can be used by different nodes, a super-script index is used to identify the threshold for a specific node.

threshold and $\cdot \setminus \cdot$ the set-difference operator. The learning task consists in finding the input feature which produces the the best split. A metric function quantifies the goodness of the split, computed as

$$I_j(\boldsymbol{\mathcal{X}}) = \frac{|\boldsymbol{\mathcal{X}}_j^{\mathbf{0}}|}{|\boldsymbol{\mathcal{X}}|} \cdot I(\boldsymbol{\mathcal{X}}_j^{\mathbf{0}}) + \frac{|\boldsymbol{\mathcal{X}}_j^{\mathbf{1}}|}{|\boldsymbol{\mathcal{X}}|} \cdot I(\boldsymbol{\mathcal{X}}_j^{\mathbf{1}}) \ , \tag{4.5}$$

where $I(\cdot)$ is the set-based metric and $|\cdot|$ the set-size operator – returning the amount of samples in the set. A common selection for $I(\cdot)$ is the Gini index, defined as

$$I(\boldsymbol{\mathcal{A}}) = 1 - \sum_{k=1}^{d} p_k^2 \ , \tag{4.6}$$

where $\boldsymbol{\mathcal{A}}$ is a generic set, d the total number of output classes and $p_k \in [0, 1]$ the fraction of element in $\boldsymbol{\mathcal{A}}$ labeled as class k, i.e. $p_k = \frac{|\boldsymbol{\mathcal{A}}^{\boldsymbol{k}}|}{|\boldsymbol{\mathcal{A}}|}$, where $\boldsymbol{\mathcal{A}}^{\boldsymbol{k}} = \{(\boldsymbol{x}^{\boldsymbol{i}}, y^i) \in \boldsymbol{\mathcal{A}} \mid y^i = k\}$ is the set of samples with label k. Note that the Gini index achieves its minimum (at 0) when the set has only elements from a single class – i.e. a pure set. Therefore, the first node is set to use the j^*-th feature of the input such that

$$j^* = \arg\min_{j \in [1..m]} I_j(\boldsymbol{\mathcal{X}}) \ , \tag{4.7}$$

with $I_j(\boldsymbol{\mathcal{X}})$ defined as in eq. (4.5). Similarly, it is possible to define the node threshold, γ_j, as the value which produces the best score for the j-th feature. However, technically, the algorithm first learns the threshold that produces the best score for every feature as

$$\gamma_j^* = \arg\min_{\gamma_j \in \mathbb{R}} I_j(\boldsymbol{\mathcal{X}}) \quad \forall \ j \in [1\,..\,m] \ , \tag{4.8}$$

and then the most discriminative feature is identified by means of eq. (4.7). At this point, the learning process moves to the second layer. It has two nodes which repeat the procedure described for the first node, but using as input $\boldsymbol{\mathcal{X}}_{\boldsymbol{j^*}}^{\mathbf{0}}$ and $\boldsymbol{\mathcal{X}}_{\boldsymbol{j^*}}^{\mathbf{1}}$, respectively. For instance, $\boldsymbol{\mathcal{X}}_{\boldsymbol{j^*}}^{\mathbf{0}}$ is split into $\boldsymbol{\mathcal{X}}_{\boldsymbol{j}}^{\mathbf{00}}$ and $\boldsymbol{\mathcal{X}}_{\boldsymbol{j}}^{\mathbf{01}}$ and the best node configuration is set. The learning process continues to build new layers until certain convergence-conditions are matched. Common conditions are maximum number of layers (a.k.a. model depth), minimum amount of leaf-samples or minimum Gini index in the leaf nodes[2]. In this way, it is possible to limit the overfitting problem, as otherwise the algorithm would keep splitting the data until every leaf has a pure set.

Random forest (RF) uses an ensemble of decision trees to perform its predictions. In particular, it stores several trained models and query all of them with the input to infer the class predictions. The final output will be selected as the class with the majority of votes – i.e. the one which is predicted most times by single decision trees. During the training stage, RF uses a family of techniques called *bootstrapping aggregation*. Specifically, given the train-set $\boldsymbol{\mathcal{X}}$, composed of inputs and labels, it samples B random sets $\boldsymbol{\mathcal{X}}_{\boldsymbol{b}} \subseteq \boldsymbol{\mathcal{X}}$, with $b = [1..B]$. The sets $\boldsymbol{\mathcal{X}}_{\boldsymbol{b}}$ are sampled with replacement[3] and generally have the same size of the original train-set – i.e. $|\boldsymbol{\mathcal{X}}_{\boldsymbol{b}}| = |\boldsymbol{\mathcal{X}}|$. Then, every set is used to train a different decision tree model. Note that this operation cannot be avoided, as tree-models trained on the same dataset exhibit very similar (or identical) configurations, thus producing highly-correlated

[2]Note that a different set-based metric function might require to set the maximum value per leaf node as condition, instead of the minimum.

[3]Every sample can be present multiple times in the same set.

outputs. By using different train-sets, bootstrap aggregation techniques force each decision tree to learn a different node-configuration, hence reducing the inter-tree correlation. Another popular technique used to encourage diversity in the forest is *feature bootstrapping*. It is applied during the training process and consists in selecting, for each node of each decision tree, a random subset of the input features for performing the data-split. In this way, it is possible to limit the impact of the strongest feature. Indeed, every tree will always use the most dominant feature to perform the data-split at the first node. By randomly concealing part of the input features, trees are forced to find out new strategies for classifying the input, thus strengthening the model's generalization ability. As a consequence, RF greatly improves the performance of a single decision tree model, reducing the variance in exchange for slightly higher bias and lower model interpretability.

4.3 Experimental Setup

This section describes the setup used during the investigation. Radar point clouds are used as input for all the tested models. In order to evaluate how the different techniques use the radar features for detecting pedestrians, it is addressed a semantic segmentation task similar to the one in section 3.3. Therefore, the models are trained to classify each point according to one of three classes: moving vehicle, moving pedestrian or background. However, besides the overall comparison proposed in table 4.1, only results on the pedestrian class are analyzed. Performance is measured in terms of F_1 score, computed as described in eq. (3.16).

4.3.1 Dataset

The dataset described in section 3.3.1 is used during the proposed investigation. However, it has been extended with new data, recorded in highway scenes. The sensor-configuration adopted by the vehicles that collected the new data is unchanged and illustrated in fig. 3.6. Although pedestrians rarely occur in highway scenes, it is important that the algorithms can detect their absence, as it will impact the precision rate and, consequently, the F_1 score. In total, the new dataset counts more than 44 million points, labeled according to the classes of the task. Every point is determined by its XY Cartesian coordinates, plus measured Doppler velocity and RCS value. Therefore, the input point cloud is provided with 4 radar features. The Doppler value associated to the points is already compensated by the ego-velocity to obtain absolute radial velocity measurements – following the insights collected from table 3.5. Consequently, throughout this chapter, the terms Doppler and ego-compensated Doppler are used indistinctly to refer to the latter quantity – unless otherwise specified. The dataset is split into train and test set, similarly to how is reported in table 3.1.

4.3.2 Experiment Procedure

The ability of NNs to use the input information in a nested, non-linear fashion, enables them to solve a various range of very complicated tasks. At the same time, however, this processing operations hinder the interpretation and explanation of the system. Indeed, since it does not exist a straightforward relation between input

and output, it is not possible to determine a priori how the input features would affect the algorithm. Therefore, to collect insights about the usage of the input features, it has been decided to design a simple experiment, exploiting the cause-effect paradigm. In particular, it is proposed to study the behavior of the model (effect) when queried with an input whose features have been altered (cause).

The experiment devised operates exclusively at the evaluation stage and requires already trained models. Unless otherwise specified, the original, unperturbed train-set is used to train all models. During the evaluation process, the original test-set is tampered and the models performance on this new set are measured. By comparing the results of the same model on the original and perturbed set, it is possible to gather insights about the effects produced by the specific modification. Since the goal of the experiment is to examine the usage of the radar features, it has been decided to alter them one-by-one, applying an additive white Gaussian noise. In this way, indeed, it is possible to corrupt the information provided by a specific feature, observe how the model performance is affected and, consequently, infer the role it plays in each model. Formally, given the original test-set $\mathbb{X}^{test}$, it has been computed the altered set $\tilde{\mathbb{X}}_j^{test}$ by perturbing the j-th feature of the points. Therefore, for a generic point cloud $\boldsymbol{\mathcal{X}}$ from the test-set with points $\boldsymbol{x^i} \in \mathbb{R}^m$, the corresponding altered point cloud $\tilde{\boldsymbol{\mathcal{X}}}_{\boldsymbol{j}}$ is obtained as

$$\tilde{\boldsymbol{\mathcal{X}}}_{\boldsymbol{j}} = \left\{ \tilde{\boldsymbol{x}}^{\boldsymbol{i}} \mid \tilde{x}_k^i = x_k^i + \nu^i \;\text{ if }\; k = j \;\text{ else }\; \tilde{x}_k^i = x_k^i \;\; \forall\, \boldsymbol{x^i} \in \boldsymbol{\mathcal{X}} \right\}, \tag{4.9}$$

where $\tilde{\boldsymbol{x}}^{\boldsymbol{i}}$ is the i-th point of the altered point cloud $\tilde{\boldsymbol{\mathcal{X}}}_{\boldsymbol{j}}$, $\tilde{x}_k^i$ and x_k^i the k-th feature of $\tilde{\boldsymbol{x}}^{\boldsymbol{i}}$ and $\boldsymbol{x^i}$, respectively – $k \in [1..m]$ –, and ν^i the additive noise, sampled from a normal distribution, i.e. $\nu^i \sim \mathcal{N}(0, \sigma)$. Note that the noise factor is sampled for every point in the point cloud – as indicated by the superscript i – and, therefore, it is not shared among them. Furthermore, note how only the selected feature j is perturbed, while the other remains unchanged. In this way, it is possible to generate m different altered sets, selecting the input features one-by-one. The dataset used for the experiments has $m = 4$. However, since there are no reason for the models to be affected differently when only the x or y spatial location is perturbed[4], it has been decided to treat the XY Cartesian coordinates as a single feature and alter them together. Consequently, when the XY features are perturbed, the noise term is sampled from a bi-variate white Gaussian distribution, i.e.

$$\boldsymbol{\nu^i} \sim \mathcal{N}\Big(\mathbf{0_2}, \frac{\sigma}{\sqrt{2}}\mathbf{I_2}\Big), \tag{4.10}$$

where $\mathbf{0_2} \in \mathbb{R}^2$ is the vector whose elements are all zeros and $\mathbf{I_2} \in \mathbb{R}^{2\times 2}$ the identity matrix with ones on the diagonal and zeros elsewhere. Note that the noise factor for each of the XY feature is sampled independently from the other – null correlation coefficient. Moreover, note how the standard deviation in each direction is scaled down by $\sqrt{2}$, to maintain the noise magnitude consistent with the sets where the other features are altered.

During experimentation, eq. (4.9) has been used to compute the point clouds for three different altered sets: $\tilde{\mathbb{X}}_{RCS}^{test}$, $\tilde{\mathbb{X}}_{D}^{test}$ and $\tilde{\mathbb{X}}_{XY}^{test}$, perturbing RCS, Doppler and XY,

[4]Note that they are de-coupled from range and azimuth measurements of the radar sensors, as they are referred to the ego-vehicle coordinate system.

respectively[5]. In order to collect more insights about the usage of each feature, the same experiment is repeated several times, extracting the three altered test-sets with different magnitudes of the white Gaussian noise – i.e. changing the value of $\sigma > 0$. Furthermore, to limit the randomness introduced by the artificial perturbation, it has been decided to compute the altered-test sets with the same configurations 10 times. The models are evaluated with every set and their average results used to generate plots. Note that the altered-sets contain radar reflections that no longer adhere to the intrinsic properties of the sensor. Therefore, they cannot be used to train the models but only to study their reaction to a disruption of the input information. Another compelling argument consists in the fact that the models can be confused by test-data that do not respect the nature of those in the train-set. However, the numerical results produced on the altered-sets are not used to infer the models performance on specific sensor-settings, but rather to gather relevant patterns that allow the study of the models behavior.

4.3.3 Network Configurations

This section reports the configurations of the tested models. The settings described in section 3.3.2, designed to enrich the input point cloud and limit the class imbalance of the problem, are adopted in this work. PointNet++ receives as input a point cloud whose locations are defined by the XY Cartesian coordinates and the ego-compensated Doppler – used as z-coordinate. The measured RCS value is used as point feature. The other methods do not require a distinction between point coordinates and features. Concerning the NN-based models, they all share the same FC classification network, composed of two layers of 64 and 32 channels, interleaved by dropout with rate 0.5. The output layer adopts the configuration described in section 3.3.2. Since RF produces class predictions as output, it does not require any classification network nor output layer.

PointNet++ It has been decided to use a PointNet++ architecture with two encoder-decoder layers. The first SA layer takes a point cloud of 1200 points and embeds local context information into 500 point-locations. FPS is used at the sampling stage, while the grouping process creates three clusters, collecting points 1, 1.5 and 2 meters around the representative one. PointNet is used to extract descriptors of 32, 32 and 64 channels, respectively, out of each local neighborhood, thus producing a 128-channels signature for every representative point. The second SA layer further encodes the features extracted by the first layer, resulting in a point cloud of 150 points. Neighborhood areas of 4, 6 and 8 meters are used, producing descriptors of 64, 64 and 128 each. Therefore, the output of the encoder section is a point cloud with 256-channels signatures. The two FP layers, propagate the extracted information into the corresponding input point locations, using shared-FC networks of 128 and 64 channels, respectively. The FC classification network computes the final semantic segmentation scores.

PointNet PointNet is implemented using 5 shared FC layers. Each of them takes as input the output of the previous layer. They are designed to extract 32, 64, 128,

[5]For completeness, note that the notation in eq. (4.9) has to be adapted to the multivariate noise factor when used to describe $\tilde{\boldsymbol{\mathcal{X}}}_{\boldsymbol{XY}}$.

256 and 256 features, respectively. The point-signatures produced by the last layer are then processed with max-pooling to produce a 256-channels global descriptor. Therefore, to each point is assigned a signature of 512 features which is fed to the classification network that computes the output class scores.

Shared FC This network has been implemented to gather insights about the impact of the global signature of PointNet on the pedestrian detection task. Therefore, it reuses the same configuration of PointNet. However, the 256-channels signatures produced by the shared-FC network is processed with an additional layer instead of max-pooling. Shared FC produces as output point clouds with 128 features. Finally, the FC classification network computes the segmentation scores for each point.

Random Forest The RF model used in this work uses 100 decision trees. Due to the dataset size, it is not possible to load all the samples in memory simultaneously. Therefore, both train-set and trees are split in 4 partitions. Bootstrapping aggregation with $B = 25$ is then used to train the decision tree models in each partition. Feature bootstrapping is also used, with sub-sampling factor $\sqrt{m}$. The Gini index in eq. (4.6) is used as set-based metric function. Two convergence criteria are set. The model depth is constrained to a maximum amount of 25 layers. At the same time, the leaf nodes are forced to have a minimum amount of 10000 samples to avoid overfitting to peculiar samples. Finally, class weights are used to deal with the dataset imbalance problem. They are employed to provide more emphasis to a given class during the set-based metric computation. It has been decided to set the weights to 1.4, 0.95 and 0.2 for the pedestrian, vehicle and other class, respectively. In practice, they are normalized and used to multiply the term p_k^2 of the corresponding class in eq. (4.6).

4.4 Results

Before investigating how the models use the radar features, it is worth analyzing their performance on the proposed semantic segmentation task. Table 4.1 summarizes the results of the tested models collected on the original test-set $\mathbb{X}^{test}$. Notably, PointNet++ achieves superior performance, considerably outperforming the other tested models on both classes. However, this does not come as a surprise, since this architecture represents the current state-of-the-art for semantic segmentation of raw radar point clouds. Focusing on the other models, it is interesting to note how RF is able to perform better than Shared FC. Bear in mind that, though these two models perform completely different processing operations, both process each input point independently from each other – i.e. they do not use any context-related information source. Therefore, the results in table 4.1 suggest that, under this configuration, an advanced thresholding of the radar features is more effective than learning to abstract them into new point-signatures. By comparing the performance of Shared FC and PointNet, it is possible to collect important insights about the role played by the context information in the detection task. Note how the introduction of the max-pooling operation produces a drastic boost in performance, improving the results on both classes by circa 9%. This proves that the global descriptor contains extremely relevant information, crucial for the correct estimation of the

Table 4.1: Summary of the models evaluated on the original test-set. F_1 scores on the positive classes are reported in percentages.

Method	Vehicle	Pedestrian	FLOPS
PointNet++	**76.82%**	**59.44%**	953.3 M
PointNet	70.02%	40.61%	366.9 M
Shared FC	61.00%	31.62%	**347.1 M**
Random Forest	65.04%	32.45%	—

semantic segmentation class of the points. The last column reports details about the computational complexity of the models. Note how PointNet++ represents the most computational-expensive solution, using nearly 2.7× more FLOPS than PointNet. Shared FC, instead, is the cheapest solution, though it shows an amount of FLOPS fairly comparable with PointNet. Finally, note that the different capacity of the models does not represent an issue, as this chapter is focused on the ability of the models to use the features, rather than their detection capabilities. The remainder of this section studies only the performance of the models on the pedestrian class, as it is in the scope of this survey.

4.4.1 Feature Perturbation

This section contains the results of the experiment proposed in section 4.3.2. Therefore, the models are evaluated on the altered sets $\tilde{\mathbb{X}}^{test}_{RCS}$, $\tilde{\mathbb{X}}^{test}_{D}$ and $\tilde{\mathbb{X}}^{test}_{XY}$. The results achieved are used to create the plots in figs. 4.2 and 4.3, reporting the F_1 scores on the pedestrian class versus various configurations of the noise magnitude σ^2. The altered sets that sample from a noise distribution with $\sigma = 0$ coincide exactly with the original, unperturbed test-set $\mathbb{X}^{test}$. Note how none of the models exhibits a performance improvement when evaluated on the altered-sets, regardless of the perturbed feature. This is an expected behavior, since the noise destroys information potentially relevant for the execution of the task. Otherwise, it would have suggested that the feature is confusing the algorithm. In this case, however, the models would have learned not to use the specific feature, as they all learn their task from data. Finally, it has been decided to separate the results in two plots, because the RCS feature required further experimentation to fully assess its usage by the tested models.

Figure 4.2 provides interesting insights about the usage of both XY and Doppler feature in the tested models. The most evident result concerns the usage of the Doppler feature. Note, indeed, how all the models are significantly affected by a perturbation of this feature, resulting in F_1 scores below 15% on the noisiest test-set configuration – i.e. $\sigma = 1$. This proves the enormous importance of the Doppler feature for the task. Yet, this is not surprising. Indeed, Doppler contains information about the dynamic properties of an object, therefore, the models strongly rely on this feature to distinguish moving pedestrians from static and/or fast-moving objects. Focusing on the single model results, it is possible to notice how, as the noise magnitude increases, PointNet tends to achieve similar performance to Shared FC: the blue and gray solid curves get really close when the noise magnitude $\sigma^2 = 0.3$. Since the only difference between the two networks consists in the max-pooling operation,

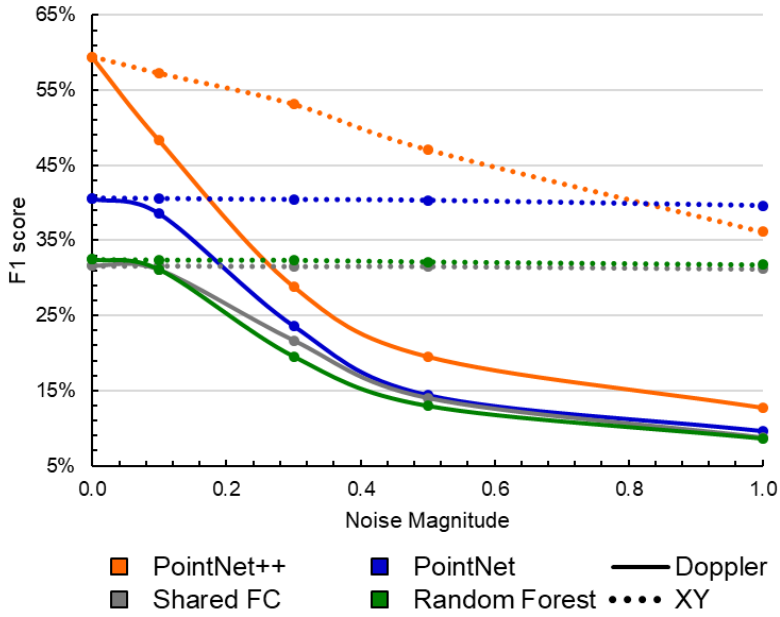


Figure 4.2: Models performance on Doppler (solid) and XY (dotted) altered test-set with different noise magnitudes. Every color represents a different model. Doppler results in the most important feature while PointNet++ is the only model using XY information. Best viewed in colors.

this behavior suggests that a perturbation of the Doppler feature limits drastically the effects of the global descriptor. Consequently, it is possible to deduce that PointNet extracts in the global signatures dynamic information about the current scene – such as the average velocity of the object – that enables the network to identify the conditions when pedestrians are more likely to be encountered. For instance, an highway scene, containing several fast moving vehicles, reduces the pedestrian occurrence probability; contrarily, an urban domain will result in numerous stationary reflections, thus defining an environment more prone to contain pedestrians. Another interesting observation can be obtained by comparing the behavior of Shared FC and RF. Note, indeed, how the former results more robust to a Doppler perturbation: the gray solid line starts below the green one ($\sigma = 0$), but they intercept around $\sigma^2 = 0.1$ and achieve the highest difference at $\sigma^2 = 0.3$. This hints that Shared FC relies less on Doppler than RF and exploits more the other radar features to detect pedestrians. Yet, both of them achieve similar performance – circa 9% F_1 score – when the noise magnitude keeps increasing and further corrupts the velocity information.

Completely different insights can be collected by analyzing the results produced on the test-sets with perturbed XY coordinates. Indeed, the models manifest profoundly diverse usages of this feature. As shown in fig. 4.2, only the dotted curve from PointNet++ results in a performance drop – up to 23% in the noisiest configuration –, while all the other models exhibit flat lines, with performance variations within 1%. This is a very interesting finding, since it suggests that PointNet++ is

the only model, among the tested ones, that makes significant usage of the point coordinates to predict the pedestrian class. This effect is attributed to the extremely different processing scheme adopted by the models. Note, indeed, how PointNet++ is the only architecture which extracts local context information, using the relative distance between points to collect shape-related attributes and/or capture the interaction with nearby objects. All the other models process the points as a list, using the coordinates as auxiliary features, as if the points in the input were independent from each other. Nevertheless, note how the performance drop produced in PointNet++ by a perturbation of the XY coordinates is not comparable with the one produced by the Doppler feature: the network can still achieve an F_1 score higher than 35% on the noisiest configuration, against circa 13% when Doppler is altered.

Figure 4.3 reports the performance achieved by the models when tested with the $\tilde{\mathbb{X}}^{test}_{RCS}$ altered set. When the RCS feature is perturbed with the same noise configuration used in fig. 4.2, the models result unaffected. Indeed, they produce flat lines, similarly to the ones produced on the XY altered-sets. In this case, however, even the performance of PointNet++ results uninfluenced. For this reason, it has been decided to conduct a further experiment to assess the usage of the RCS radar feature by the models. In particular, the models are evaluated on a new altered test-set, where the RCS information has been completely destroyed. Formally, this is tantamount to sample the noise factor from a normal distribution with infinite magnitude – i.e. $\sigma \to \infty$. Therefore, the models scores are annotated to the extreme right of

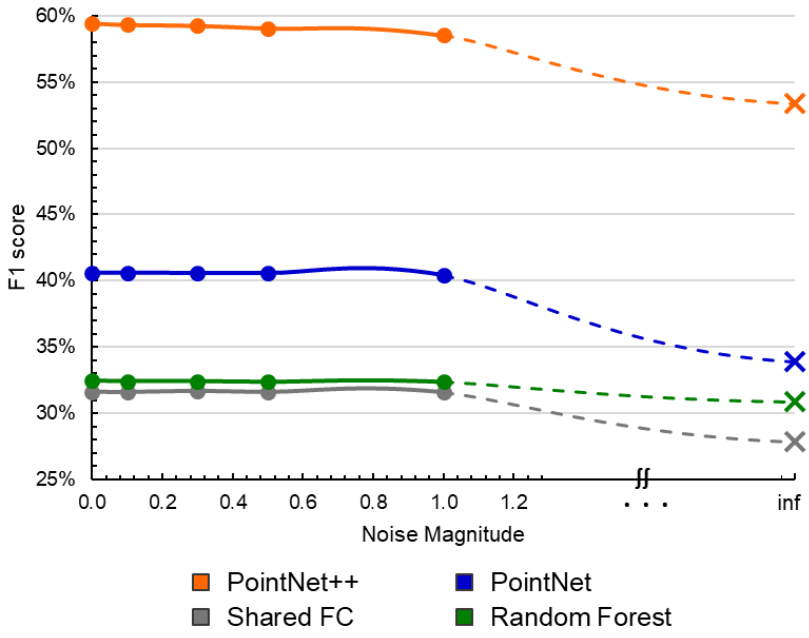


Figure 4.3: Models performance on RCS altered test-set with different noise magnitudes. Cross markers report the models performance while completely destroying the information. Best viewed in colors.

Table 4.2: F_1 score performance of the models trained with and without the RCS feature. RCS is useful for the semantic segmentation task.

Configuration	PointNet++	PointNet	Shared FC	Rand. Forest
with RCS	**59.44%**	**40.61%**	**31.62%**	**32.45%**
without RCS	57.99%	40.45%	30.04%	31.52%

the plot (cross markers), connecting with dashed lines the results obtained with the last finite noise magnitude configuration. From an implementation point-of-view, it has been decided to force the RCS value of all the points to a constant value of 0 – to ensure numerical stability. Note how, when this noise configuration is adopted, all models exhibit a noticeable drop in performance. This suggests that, though RCS is not the most important feature, it contains relevant information for the detection of pedestrians. Remarkably, PointNet shows the highest degradation, decreasing its F_1 score by circa 7%, even more than PointNet++. This hints that this architecture relies on RCS more than the other models to detect pedestrians. Furthermore, note how RF results in the lowest performance drop, showing a variation of less than 2% – i.e. almost half as much as the degradation produced in Shared FC. Therefore, it is possible to conclude that NN-based solutions can leverage all the input information, learning also from a less informative feature like RCS. Finally, it has been decided to perform a last experiment, in order to find out whether the models might benefit from not using the RCS feature at all. Indeed, since fig. 4.3 proved that it does not contain prominent information for the detection task, its inclusion in the input point-signature might force the models to use this information, thus generating confusion in the decision-process. During this experiment, the RCS feature has been removed from the whole dataset – both train and test sets. Then, all four models have been trained from scratch and evaluated, using the new dataset. Table 4.2 reports the performance of the models when trained and tested on the dataset with and without the RCS feature. Note how all models achieve better results on the dataset that includes the RCS feature, thus proving that it does not generate confusion but rather support the solution of the semantic segmentation task. Indeed, this feature is proportional to the visibility of the object from a radar perspective – combining its distance to the sensor with the active reflective surface. Therefore, the models can leverage this information to improve the effectiveness of the detection process. In particular, NN-based models can use this feature to adjust the filtering operations based on the object visibility.

4.4.2 Doppler Analysis

The results in section 4.4.1 prove that Doppler contains the most prominent information for the detection of pedestrian reflections. The very poor performance produced when perturbing this feature, indeed, suggests that the models use it as a prediction-enabler: i.e. if a point shows a particular Doppler configuration, then the other features are used to determine whether it is a pedestrian point; otherwise the point obtains a different class prediction. In this section, it is provided a deeper investigation on the Doppler feature, reporting the results of additional experiments that unveil further details about its usage.

A particularly interesting experiment consists in the analysis of the Doppler values that enable the models to predict the pedestrian class. This can be achieved by extracting statistics from specific sets of points. Herein, it has been decided to compare the Doppler distributions of points predicted as pedestrian by the different tested models. To compute this statistic, first, it is extracted, for each model, the set containing only points of the test-set which result in pedestrian predictions. Then, the Doppler values of the points are used to create a probability distribution. In particular, the Doppler support has been divided into non-overlapping intervals and the amount of point-occurrences in each interval is annotated. Finally, the interval-counts are normalized by the total amount of points in the set to obtain probability scores. The models used in this experiment are the same shown in table 4.1, hence trained on the original, unaltered train-set. Note, moreover, that the points are collected from the original test-set, without any perturbation of the radar features.

Figure 4.4 contains the results of the experiment. It plots the Doppler distribution of different models, obtained by interpolating the probability scores at adjacent intervals. In addition, the Doppler distribution of GT pedestrian points is reported to provide a reference (dotted line). The same procedure is used to generate this curve, but the statistics are computed on a set containing all the points which are labeled as pedestrian. Note that, since the Doppler support is divided in intervals of 0.2 m/s, the score at a generic point x refers to the percentage of pedestrian predictions with Doppler values in the interval $[x-0.1, x+0.1)$ m/s. For instance, the value of the blue curve at 0.2 m/s indicates that nearly 5% of the points predicted as pedestrian by PointNet have Doppler values in the interval $[0.1, 0.3)$ m/s.

By analyzing the plot, it is possible to gather many interesting insights. Bear in mind that the pedestrian class refers to moving instances which have an absolute velocity higher than 0.5 m/s. Since it is not physically possible for pedestrians to

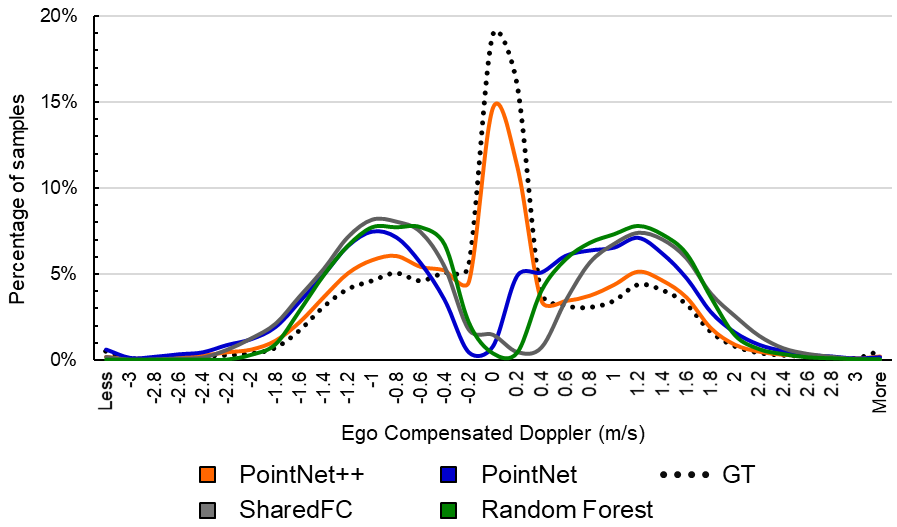


Figure 4.4: Doppler distributions of pedestrian predictions. Different colors represent different models. The dotted black curve refers to the Doppler distribution of GT pedestrian points. Best viewed in color.

achieve high speed values – compared to other objects such as vehicles –, they often assume low velocity values (hardly ever beyond 3 m/s). Therefore, it is natural to expect the GT pedestrian Doppler distribution to be the sum of two Gaussian distributions, centered around $\pm d_{avg}$, i.e.

$$\text{pdf}(D) = \mathcal{N}(-d_{avg}, \sigma_d) + \mathcal{N}(d_{avg}, \sigma_d) \ , \tag{4.11}$$

where $\text{pdf}(\cdot)$ stands for probability density function, D is the Doppler random variable. $d_{avg} > 0$ and $\sigma_d > 0$ are, respectively, the average absolute velocity of pedestrians and the corresponding standard deviation, computed as in eq. (4.12), where N is the total number of points in the set and D^i the Doppler value of the i-th point,

$$d_{avg} = \frac{1}{N} \sum_{i=0}^{N} |D^i| \ , \tag{4.12a}$$

$$\sigma_d = \sqrt{\frac{1}{N} \sum_{i=0}^{N} \left(|D^i| - d_{avg}\right)^2} \ . \tag{4.12b}$$

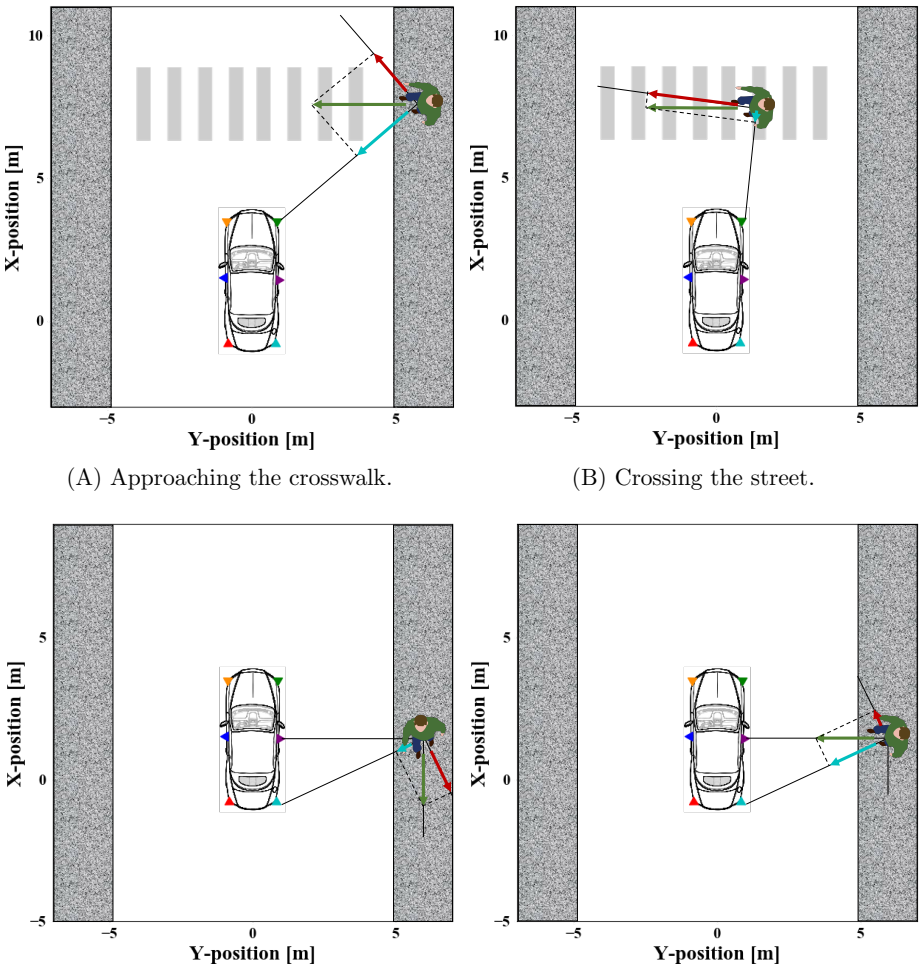
This assumption is partially confirmed by the dotted line in fig. 4.4. Note, indeed, the presence of two side-lobes, resembling two Gaussian distributions centered around circa -0.8 and 1.2 m/s, respectively. However, only a limited amount of pedestrian points have a Doppler configuration sampled from these two distributions. Indeed, it is possible to notice the presence of an unexpected third Gaussian component, centered around 0.1 m/s, with a smaller standard deviation than the other two distributions. Remarkably, this component accounts for the majority of the points, as $P(D \in [-0.2, 0.4])$ – the probability of a pedestrian point to have a Doppler value in the interval $[-0.2, 0.4]$ m/s –, computed as in eq. (4.13), is nearly 45%,

$$P(D \in [a, b]) = \int_a^b \text{pdf}(D) \ dD \ . \tag{4.13}$$

This effect is attributed to two aspects: the nature of the radar velocity measurement and the behavior of pedestrian instances in the scene. Bear in mind, indeed, that Doppler provides a measure of the relative radial velocity of the object. Although it is possible to compensate the sensor-velocity to achieve an absolute measurement, the tangential velocity component of every point remains unknown[6]. Consequently, pedestrians that move tangentially w.r.t. the sensor produce reflections with null Doppler. Yet, since they have a non-null absolute velocity, they will be labeled as moving, thus explaining the presence of samples around the stationary Doppler. Furthermore, note that pedestrians are specific objects that can be found performing a finite set of actions: they are often either sitting/standing-still, crossing the street or walking along the curbstone. Instances which perform the first action are labeled as stationary objects. The other two actions generate positive point-labels. However, they rarely result in pedestrians moving away/toward the sensor. Contrarily, they often move in directions with a strong component in the tangential plane, thus producing a significant divergence between the Doppler measured velocity and the real absolute velocity of the object. Figure 4.5 contains examples of common

[6]It is possible to recover the tangential velocity of an object by leveraging the radial measurements of multiple points. However, this approach requires the point-to-object association to be known a priori.

actions taken by pedestrians, highlighting the radial and tangential velocity components. Consider, without loss of generality, that the ego-vehicle is standing in the scene, so that the measured Doppler coincides with its ego-compensated version. Figures 4.5A and 4.5B show a pedestrian crossing the street. Note how, when the pedestrian is leaving the curbstone to approach the crosswalk (fig. 4.5A), the absolute velocity (green arrow) is split in almost equal radial (cyan arrow) and tangential (red arrow) components w.r.t. the front-right sensor (green triangle). Consequently, a pedestrian moving at a velocity of 0.5 m/s would produce a Doppler measurement of circa 0.25 m/s. However, as it moves towards the middle of the street, its radial



(A) Approaching the crosswalk.

(B) Crossing the street.

(C) Walking along the curbstone.

(D) Walking towards the ego-vehicle.

Figure 4.5: Possible actions of moving pedestrians. The sensors are identified with triangle markers, oriented according to the mounting setting. The green arrows show the absolute velocity of the pedestrian. Red and cyan arrows represent, respectively, the tangential and radial components of the absolute velocity. Radar Doppler measures the length of the cyan arrow. Best viewed in colors.

component will decrease[7]. Notice, indeed, how much the size of the cyan arrow has shrunk in fig. 4.5B, while the tangential component has grown to become almost as big as the green arrow. In this configuration, the Doppler value is critically different from the real absolute velocity. Similar consideration can be repeated for pedestrians walking along the curbstone. Note, indeed, that curbstones often run either in the lateral direction (y-axis) in the front or rear of the vehicle or in the longitudinal direction (x-axis) on the sides of the vehicle. In the former case, the velocity vectors assume configurations similar to the ones in figs. 4.5A and 4.5B. The latter scenario, instead, is depicted in fig. 4.5C. Note that, when the pedestrian is on the side of the ego-vehicle, both right-facing (purple triangle) and rear-right (cyan triangle) sensors measure very low Doppler values. In particular, the right-facing sensor has the radial axis exactly perpendicular to the moving direction. Therefore, the absolute velocity lies entirely in the tangential plane and the sensor will read a null Doppler value. Similar observations can be repeated for the rear-right sensor. Indeed, though it is not completely orthogonal to the moving direction, it will still measure a very small radial velocity, as proved by the small size of the cyan arrow. Note that, when the pedestrian is not at the side of the vehicle, it assumes similar configurations to fig. 4.5A. Finally, as a comparison, fig. 4.5D shows the configuration produced by a pedestrian walking towards the ego-vehicle. Notice how the Doppler value of the right-facing sensor measures exactly the absolute object velocity, while the rear-right sensor provides an accurate estimate. However, this scenario rarely happens and, frequently, the measured Doppler value is consistently different from the absolute object velocity, as proved by the high-amount of pedestrian samples with values close to the stationary Doppler.

The fact that the GT pedestrian Doppler distribution diverges from eq. (4.11) provides significant information about the perception of pedestrians in the Doppler domain. By analyzing the curves generated from the model prediction-sets in fig. 4.4, it is possible to gather further insights about the usage of this feature. Note how the Doppler distributions produced by all the models present the two Gaussian components derived in eq. (4.11). This suggests that the models are prone to predict the pedestrian class when the reflections exhibit a low Doppler value – around $\pm 1\ m/s$. However, it is surprising to note how PointNet++ is the only technique which produces a distribution similar to the GT one. Indeed, it presents a main Gaussian component centered around $0.1\ m/s$, describing a big portion of its predictions – $P(D \in [-0.2, 0.4]) = 34\%$ circa. This proves that PointNet++ performs a more complex usage of the radar features, extracting the properties that describe the pedestrian class. Among the other models, PointNet attempts to perform pedestrian predictions for points with Doppler values close to $0\ m/s$. However, only less than 12% of predictions have Doppler values in the interval $[-0.2, 0.4]$, while $P(D \in [-1.8, -0.4])$ and $P(D \in [0.6, 2.0])$ are both above 40%. Finally, the remaining models just limit themselves to predict the pedestrian class if the points have a small, non-null Doppler value, showing $P(D \in [-1.8, -0.4] \cup [0.6, 2.0]) > 90\%$.

Figure 4.6 reports the results of the same experiment, repeated on the false-negative sets. Therefore, the colored lines show the Doppler distribution of pedestrian points which are missed by the models. The GT Doppler distribution curve is computed exactly as in fig. 4.4. It has been decided to normalize the interval-counts by the total amount of GT pedestrian points rather than the size of the correspond-

[7]Assuming that it maintains a constant velocity.

ing set. Consequently, the curves are not real probability distributions, as the area under them is not 100% – i.e. $P(D \in [-\infty, +\infty]) \neq 1$. Note, however, how this does not affect the shape of the curves: they are just scaled down to fit under the dotted curve. The proposed plot enables one to collect interesting insights. Note, indeed, how the area under a colored curve accounts for the percentage of missed pedestrian points. Instead, the area between the colored and dotted curve consists in the percentage of detected samples – i.e. the recall score. Figure 4.6 confirms that PointNet++ performs a different processing compared with the other models. Notice, indeed, how it is possible to distinguish only the orange curve in the interval $[-0.2, 0.4]$. The other curves are superimposed with each other and the dotted line. This suggests that these models miss almost all the pedestrian points with Doppler values in this interval. Therefore, the points which are predicted as pedestrian by PointNet in this interval – 12% of the whole set – are nearly all false positives. Regarding PointNet++, it is possible to note how, despite it predicts the pedestrian class for points with Doppler in $[-0.2, 0.4]$, it misses several of them. Numerically, they account for nearly 25% of the total GT pedestrian points and circa 58% of points in the specific interval. Yet, PointNet++ is able to detect the remaining 42% circa of points, compared to less than 5% in the other models. Furthermore, note how PointNet++ shows superior performance also in the interval $[-1.8, -0.4] \cup [0.6, 2.0]$. Indeed, here, the orange curve is always found below the other curves, thus producing a higher recall score. Numerically, it misses only 8% circa of total pedestrian points against more than 12% in the other models, which account, respectively, for nearly 16% and 25% of the GT pedestrian points in this interval.

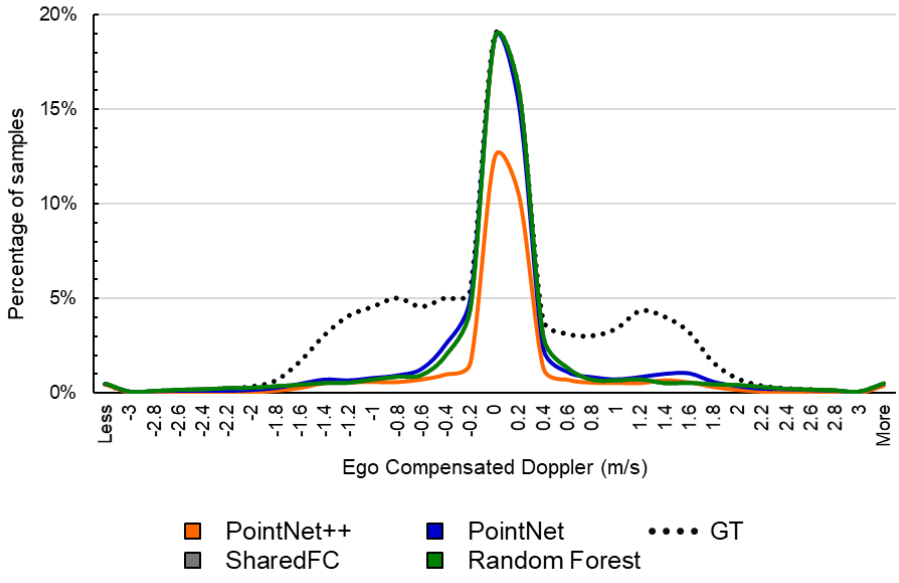


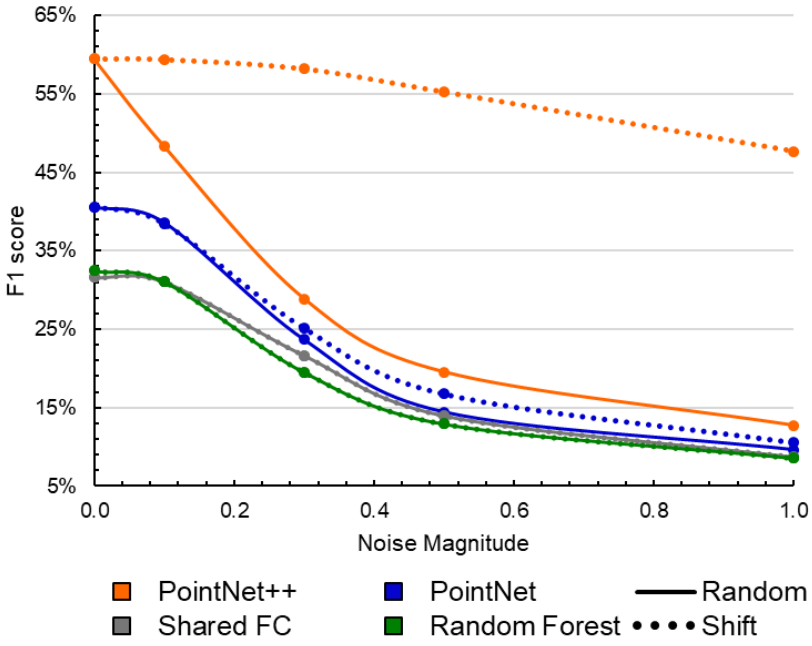
Figure 4.6: Doppler distributions of missed pedestrian points. Different colors represent different models. The dotted black curve refers to the Doppler distribution of GT pedestrian points. Best viewed in color.

Random vs Shift Noise In order to gather further insights about the usage of Doppler, it has been decided to conduct an additional experiment. Specifically, it is studied how the models are affected when this feature is perturbed with a different noise factor. The experimental procedure described in section 4.3.2 is adopted. Therefore, the models are evaluated on altered versions of the test-set, where a single feature is perturbed with additive white Gaussian noise. Equation (4.9) describes the perturbation process: given a point cloud $\boldsymbol{\mathcal{X}}$ from the original test-set, the corresponding perturbed point cloud $\widetilde{\boldsymbol{\mathcal{X}}}_{\boldsymbol{j}}$ is obtained by adding to the j-th feature a noise term sampled from a normal distribution. Note that every point experiences a (potentially) different noise term. For this reason, this noise configuration is referred to as random noise. In this section, it has been decided to modify the perturbation process: a single noise term is sampled and used to alter the j-th feature of all the points in the same point cloud, i.e.

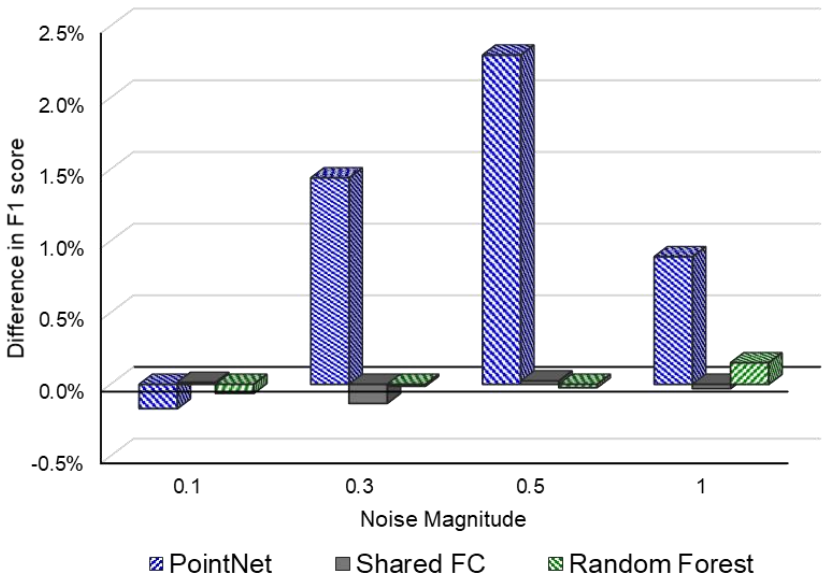
$$\widetilde{\boldsymbol{\mathcal{X}}}_{\boldsymbol{j}} = \left\{ \widetilde{\boldsymbol{x}}^{\boldsymbol{i}} \mid \widetilde{x}^i_k = x^i_k + \nu \;\text{ if }\; k = j \;\text{ else }\; \widetilde{x}^i_k = x^i_k \;\; \forall\, \boldsymbol{x}^{\boldsymbol{i}} \in \boldsymbol{\mathcal{X}} \right\}. \tag{4.14}$$

Note that, compared to eq. (4.9), ν has no longer the superscript i, thus remaining constant across points of the same point cloud. This results in a shift of the Doppler values which is equal for all the points. Therefore, this noise configuration is referred to as shift noise. Note, however, how the noise term is not shared across different point clouds, which are shifted by (potentially) different factors. The noise distribution remains unchanged, i.e. $\nu \sim \mathcal{N}(0, \sigma)$. Finally, note how the shift noise is less powerful than the random noise – given equal magnitude – as it does not corrupt the relative local velocity information.

Figure 4.7A contains a comparison on the models performance under both random and shift noise perturbation of the Doppler feature. Note how the solid lines are the very same curves plotted in fig. 4.2. It is worth noting how PointNet++ exhibits once more a different behavior than the other models. Indeed, it results less affected by the shift noise than the random one: when its magnitude is below 0.3, it experience almost no performance drop ($< 2\%$). This is attributed to the different processing performed by this architecture, which uses relative point coordinates to extract local information. Since the shift noise does not affect the difference in Doppler between points of the same point cloud, it does not corrupt the information leveraged by the network, which is able to maintain its performance – bear in mind that Doppler has been used as z-spatial coordinate. Interestingly, however, when the noise magnitude increases, the performance of the network experience a noticeable degradation. This effect is attributed to the huge importance of the Doppler feature for the detection task. Indeed, as the noise magnitude increases, so does the shift experienced by the point-feature values. This, in turn, increases the probability of producing Doppler configuration which are unnatural for pedestrians, thus preventing the network to predict the correct class – fig. 4.4 shows that PointNet++ is less prone to classify as pedestrian points with Doppler values above 3 m/s. Yet, notice how the performance drop produced by the shift noise is significantly lower than its random version: the two configurations result in a difference of nearly 35% in terms of F_1 score. Concerning the other models, it is possible to note how the shift noise produces results that are very similar to its random version. This behavior, however, cannot be fully appreciated from the plot in fig. 4.7A, therefore, it has been decided to produce a bar-chart, reporting the difference between the performance produced by shift and random noise. In this way, it is possible to analyze even



(A) Absolute performance. PointNet++ shows reduced sensitivity to Doppler shifts.



(B) Difference in performance between random and shift noise.

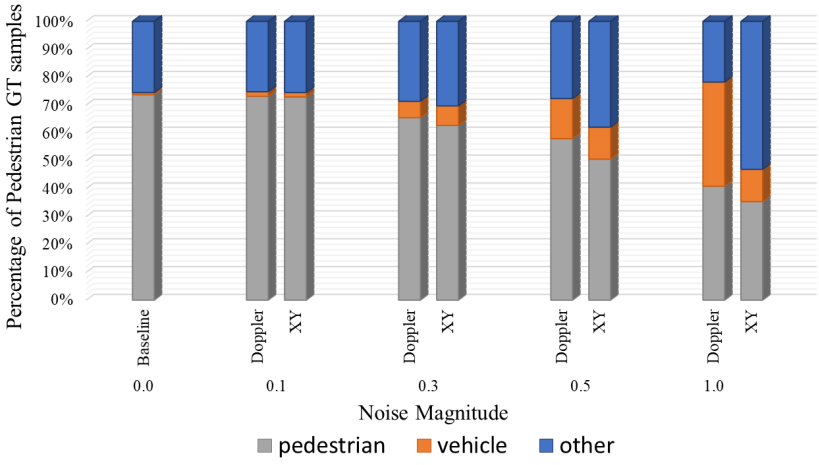
Figure 4.7: Models performance vs noise magnitude under random (solid) and shift (dotted) noise applied to the Doppler feature. Best viewed in colors.

very small variations in performance. Figure 4.7B contains the bar-chart. Note how both Shared FC and RF exhibit similar negligible differences: considering all the noise configurations, the performance achieved on the two noise factors never differ more than 0.2%. PointNet results slightly more robust to the shift noise than the random one, showing differences up to 2%. This effect is attributed to the usage of the global signature. Note, indeed, how the shift noise has the same effect of a poor estimation of the ego-velocity – a systematic error would result in a Doppler shift. As the global signature captures information of the whole scene, it can be used by the network to resolve the erroneous Doppler measurement and still predict the correct class.

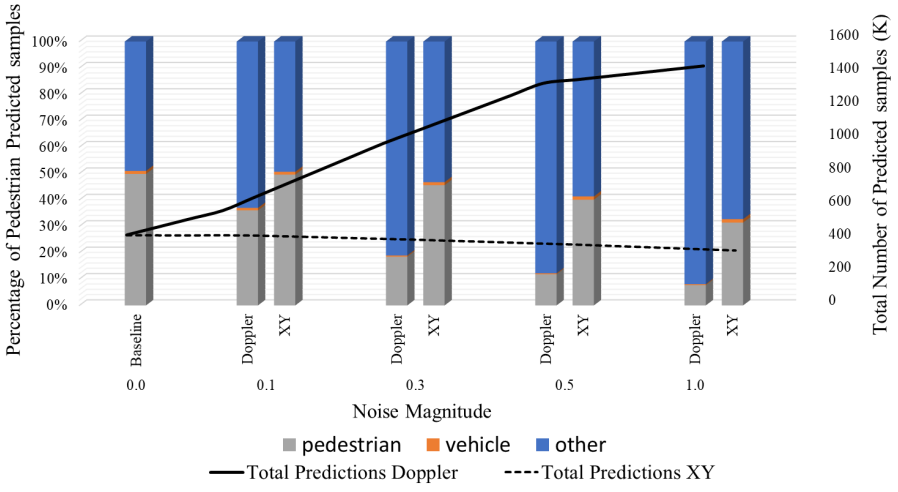
4.4.3 Focus on PointNet++

The results of the experiments in sections 4.4.1 and 4.4.2 prove that PointNet++ uses the radar features differently than the other tested models. For this reason, it has been decided to perform a deeper analysis, focused on this architecture. In more details, it has been proposed an investigation on the class-confusion produced by perturbations of the radar features. The random noise factor, as described in section 4.3.2, is used. Moreover, since perturbations of the RCS feature have minor effects on the performance, only Doppler and XY are considered. Therefore, the same experiment which led to the results of fig. 4.2 is repeated. However, instead of collecting the F_1 scores, for each point in the test-sets, both GT class and network prediction are annotated and used to build the column-charts in fig. 4.8.

Figure 4.8A shows how the network classifies all GT pedestrian points in the test-set, thus providing a measure of the recall score. As the noise magnitude increases, it is possible to recognize two effects: when the Doppler is perturbed, more pedestrians are misclassified as moving vehicles (orange segment) than background/other (blue segment); contrarily, when the spatial XY coordinates are altered, it is the background/other class which dominates the erroneous prediction-set. The former effect is attributed to the fact that, when perturbing the Doppler feature, it is more likely to produce a high value (proper of moving vehicles) than a value close to 0 (characteristic of stationary objects, accounted for in the background/other class). Consider, for instance, a pedestrian point with measured Doppler value of 1.5 m/s. By sampling the noise term from a normal distribution with $\sigma = 1$, there is a 30% probability that the Doppler value will be increased by at least 0.5 m/s. Contrarily, there is only 8% probability that the noise would not alter the Doppler value – considering the interval $[-0.1, 0.1]$ m/s – and circa 5% probability to produce a value close to 0 – i.e. noise term in $[-1.7, 1.3]$ m/s. The effect produced by a perturbation of the XY coordinates, instead, can be attributed to missing context information. Indeed, although the Doppler feature has the right configuration, the surrounding point-geometry is altered, thus preventing the network from predicting the pedestrian class. Yet, since the characteristic Doppler of pedestrians is often low – more than 95% of points are in the interval $[-2, 2.2]$ m/s –, the network does not find the conditions for predicting the moving vehicle class and, therefore, decides for the background/other class. Finally, an interesting result can be observed by comparing the heights of the gray segments produced by the altered test-sets. Note, indeed, how perturbations of the XY coordinates always generate lower recall scores than Doppler – with noise magnitude $\sigma = 1$, it is circa 43% against nearly 49% from



(A) 100% accounts for all GT pedestrian points. The height of each bar segment reports the percentage of GT pedestrian points classified as the proper class. The height of the gray segment represents the network **recall** score.



(B) 100% accounts for all pedestrian point-predictions. The height of each bar segment shows the percentage of pedestrian predicted points from the proper GT class. The height of the gray segment reports the network **precision** score. The secondary axis (right-hand side) plots the total number of points predicted as pedestrian on the Doppler (solid) and XY (dashed) altered test-sets.

Figure 4.8: Column-chart showing the distribution of pedestrian predictions or GT points across different classes vs different noise magnitudes. Column-segments of different colors represent different classes. Best viewed in color.

Doppler. This suggests that the network relies more on XY than Doppler for the detection of pedestrians, thus contradicting the observations made so far. However, it must be noted that it is possible to achieve 100% recall by simply predicting always the same class, regardless of the input features. Therefore, in order to produce correct observations, it is paramount to analyze the corresponding precision scores – which, for instance, will be 0% for the previous model example with 100% recall.

Figure 4.8B provides a measure of the network precision, as it reports the GT class distribution of pedestrian predicted points. Note how a perturbation of the Doppler feature significantly affects the amount of correctly predicted pedestrian points: when the noise magnitude σ is 1, the network achieves a precision of nearly 6%, against almost 30% on the XY altered-set. Therefore, it is possible to conclude that the XY coordinates are important to achieve high recall values – i.e. to correctly classify pedestrian points –, while Doppler is fundamental to limit the amount of pedestrian predictions from other classes. This is additionally supported by the results plotted on the secondary axis of fig. 4.8B, showing the total amount of points predicted as pedestrian. Indeed, it is possible to note how a perturbation of the Doppler feature (solid line) critically increases this number – up to $3\times$ the amount of predictions on the original test-set when $\sigma = 1$ –, thus strongly affecting the precision score. Conversely, the XY altered sets (dashed line) produce a slight reduction in the number of points predicted as pedestrian. Finally, it is worth noting how, in both XY and Doppler altered sets, it is the background/other class that represents the main source of error. This can be attributed to different reasons. In the Doppler altered sets, the main cause is the augmentation of the velocity feature of points which have similar spatial context configuration of pedestrians. For instance, consider objects like poles, bushes or trees, which are associated with the background/other class. They produce ego-compensated Doppler values close to 0 m/s, but share the common spatial properties of pedestrians – i.e. they are found near other static objects like parked cars or buildings. Consequently, by tampering the Doppler value of their radar reflections, it is possible to generate the conditions for predicting the pedestrian class. This does not hold true for moving vehicles. Indeed, since they have well-defined Doppler signatures, the noise perturbation has little effect – its magnitude is small compared to the feature value. In addition, they rarely share the spatial configuration of pedestrians. Similar considerations can be repeated on the XY altered sets. However, the effects produced are limited, because it is much more rare to find points that have the proper Doppler signature but miss the spatial context configuration. Moreover, compared with Doppler, it is less likely that the XY noise would create the proper context configuration, as more points must assume specific spatial positions. Therefore, the drop in precision achieved by an XY perturbation is mostly attributed to correct pedestrian predictions no longer produced. Indeed, as shown in the secondary axis, when $\sigma = 1$, the network total predictions number goes from circa $394K$ (on the original test-set) to $300K$. This accounts for a loss of nearly 24% of pedestrian predicted points, which are almost all from the GT pedestrian class ($102K$). Consequently, it is possible to conclude that PointNet++ reacts to a Doppler perturbation by predicting points from other classes as pedestrians. On the other hand, a perturbation of the XY coordinates forces the network to misclassify GT pedestrian points. Yet, the overall performance drop produced by a perturbation of the Doppler feature results more severe, thus establishing its role as most informative feature.

Chapter 5

An Advanced Point-based Architecture for Detecting Objects in Radar Point Clouds

The previous chapters of this document prove the ability of point-wise processing techniques to extract practical information from radar point clouds through the semantic segmentation task. However, in order to enable safety functions and driver assistance systems, it is important to obtain a reliable and more accurate perception of the surrounding environment. Despite semantic segmentation outputs provide useful insights about the distribution of the objects, they do not fully characterize the scene. Indeed, it is crucial to gather details about the object boundaries and determine their appearance and space occupation, as this allows the prediction of future behaviors, thus triggering appropriate actions.

This chapter is dedicated to the ultimate perception task – i.e. object detection. The first section briefly introduces the various environment perception tasks enabled by radar point clouds. The second section presents the proposed multi-class object detection system, providing details about the solutions designed to optimize the prior art. Section 5.3 describes the setup used to perform the experiments. Finally, section 5.4 reports the results collected, proving the effectiveness of the proposed solutions.

The following own contributions are made:

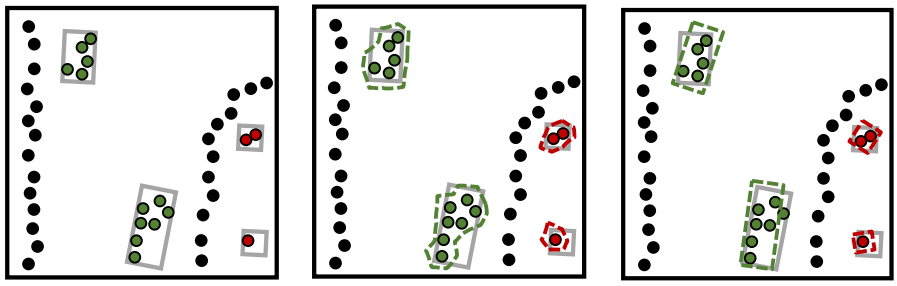
- Development of a novel point-wise processing architecture for the object detection task.
- Design of a center-proposal task, generalizing to radar point clouds the image-based anchor-localization task in [101].
- Design of a regression-aided framework for point-wise processing techniques.
- Development of a method for the reduction of box-predictions per object.
- Adaptation of the spherical regression task in [96] for box-orientation estimation.
- Design of a dynamic weighting factor to optimize learning of the box-orientations.

- Development of a low-cost Intersection-over-Union (IoU) score approximation, useful to generate labels for box-confidence estimation.
- Evaluation of the proposed system in real-world scene with multiple targets of various classes.
- Evaluation of the usage of multiple radar frames to improve the performance in the object detection task.

The method described in this chapter obtained a patent in [107].

5.1 Environment Perception through Radar Point Clouds

Radar point clouds provide useful information about the environment around the sensor. Every point represents an object observation, capturing relevant properties of the target. Therefore, by leveraging this data, it is to possible achieve a sense of awareness about the surrounding space. As shown in fig. 5.1, there are various ways of representing this knowledge. One of them uses the semantic segmentation task, providing details about the objects distribution in the scene. It consists in the classification of all the input points according to a given set of object classes. Figure 5.1A displays an example of environment perception through semantic segmentation. Notice how the green labeled points indicate the presence of an instance of the vehicle class in a specific region of the space. It is very straightforward to address the semantic segmentation task on point clouds. For this reason, it is the most common and widespread perception representation on such data-format. Section 2.3.2 contains an overview of the most successful DL approaches for semantic segmentation of radar point clouds. Although this task allows the localization of objects, however, it fails to provide exhaustive information about them. For instance, it does not enable distinction between different physical instances. This setback can be circumvented



(A) Semantic Segmentation. (B) Instance Segmentation. (C) Object Detection.

Figure 5.1: Examples of various type of environment perception tasks. Points represent radar observations while gray boxes identify GT objects in the scene. The point color shows the semantic segmentation output class: background (black), vehicle (green) and pedestrian (red). Colored dashed contours identify regions which contain points from the same physical instance. Colored dashed boxes refer to predicted box-object proposals. Best viewed in color.

by assuming that every point represents a different object. However, despite this assumption might hold true for objects which are represented by few points, it dramatically fails on objects which are associated with multiple-points. Consider, for instance, the pedestrian at the bottom-right corner in fig. 5.1A. Since it generates a single radar-reflection, it is possible to detect the presence of a pedestrian instance by correctly classifying the single-point. However, the same cannot be repeated for the vehicle at the bottom of the scene. Indeed, it generates several point-detections, therefore, the number of physical instances it is not straightforwardly determined.

Nevertheless, the perception information provided by the semantic segmentation output can be leveraged to execute more advanced processes. One example is represented by the instance segmentation task. It extends the semantic segmentation output by associating every point with a single physical object, as shown in fig. 5.1B. The simplest procedure for performing this task uses two-steps: first every point is associated with a class, then, a clustering algorithm groups together data-points which belong to the same instance. Note how, through the semantic segmentation output, it is possible to filter the data in the point clouds, extracting only those points which are related to objects of a given class, thus facilitating the grouping process. Concerning radar point clouds, Schumann et al. [114] recently proved the ability of point-wise processing approaches to perform the instance segmentation task. The authors developed a two-stage model that consumes raw radar point clouds and produces instance proposals. The first stage uses a modified version of PointNet++ [64] to generate the semantic segmentation scores. Furthermore, an additional output head leverages the classification features to produce direction predictions: for every point, it estimates the vector which would move the point from its spatial location to the center of the object it belongs to. The outputs of the first stage are then propagated to the instance segmentation module. This latter shifts the input points by the predicted direction vectors so that points which belong to the same instance lie closer together. Then, it uses the semantic segmentation scores to distinguish points from different classes and produce instance proposals. The neighboring points are collected and two proposals are merged if they have a non-empty intersection. Finally, the instance proposals are processed with a PointNet [65] network that provides a single class score for all points belonging to the same instance.

Despite the instance segmentation task allows recognition of different physical objects, it does not fully characterize them. For instance, it does not provide shape information, thus limiting the object perception to its point-observations. A more advanced task is object detection/recognition. It consist in the joint estimation of location and appearance of the physical instance in the scene through box-predictions. An example is shown in fig. 5.1C, where the colored boxes display the object detection output. Note how this perception task enables full representation of the physical objects in the environment, providing details about its location, shape and orientation. In the DL community, object detection is a very widespread task, especially on very dense data provided by camera or lidar sensors [80, 79, 101, 123, 104]. However, it is not easily addressed on radar point clouds, due to the low density of the data. Recently, Danzer et al. [92] proposed a single-class solution for amodal BB estimation on radar data. Inspired by the work of Qi et al. [79], they developed a three-stage network that consumes raw point clouds to predict oriented-boxes for cars. The first stage is a patch proposal module. It extracts a subset of the points

in the input and deliver them to the next stage. The authors used every point in the input as patch-center and defined the patch as the local point cloud in a window of given length and width. The second stage has the task of detecting whether the patches contain a vehicle. Therefore, it receives as input all the points in a single patch and produce a classification score, by means of a PointNet [65] architecture. Furthermore, the patches which are classified as vehicle are semantically segmented, leveraging the point-related signatures as well as the global descriptor computed in the classification stage. The points which are segmented as vehicle are extracted from the patch and delivered to the last stage, where the amodal BB estimation takes place. Here, the first operation consists in a resampling of the patch center, which is set as the centroid defined by the vehicle points. Then, a transformer PointNet is used to perform a rough estimate of the object center, shifting once more the patch center. Finally, a box regression PoinNet is used to predict the box parameters – i.e. length, width, orientation and center. It adopts a classification-regression framework: several pre-defined templates – with different size/orientation configurations – are classified; then, residuals are regressed to adjust the final prediction. This work proved the ability of point-wise processing methods to estimate object-boxes for vehicle instances using radar point clouds.

5.2 Multi-Class Object Detection Architecture

Point-wise approaches represent an effective tool for processing radar point clouds. Several researchers have adopted this technology to perform semantic segmentation tasks [81, 93, 118]. However, in order to assess the full potential of point-wise methods, it is paramount to compare them with grid-based techniques. Since the semantic segmentation task produces an output which is strictly bound to the input representation, it does not enable such analysis. Indeed, it is difficult to perform a fair comparison between a segmented grid and a segmented point cloud, as the evaluation procedure will always result biased. Other researchers have proved the ability of point-wise processing methods to execute more advanced tasks like instance segmentation [114] and object detection [92]. This latter, in particular, provides an output which is dissociated from the input representation, thus enabling the comparison with other families of techniques. Recently, Dreher et al. [109] used the object detection task to analyze the performance produced by a point-wise [92] and a grid-based [80] method on radar point clouds. The authors showed that the former solution processes this data-format more efficiently than the latter and attributed this effect to the point-to-grid transformation – resulting in many empty cells which amplify the overfitting effect.

Despite the model developed in [92] achieved better performance than grid-based solutions, it still has few shortcomings. The main drawback can be found in the first two stages, which perform an unnecessarily redundant processing. Indeed, the patch proposal module uses all the points in the input as patch centers. Therefore, given a point cloud $\boldsymbol{\mathcal{X}}$ with N points, it produces N patches. Then, every patch is fed to the next stage, where it is computed a patch-classification score – based on the points in the patch – as well as semantic segmentation scores – i.e. a score for every point. However, some patches contain very similar neighborhood configurations, thus forcing the network to repeat the same processing. Consider, for instance, two points that lie very close to each other. These points will generate two different

patches, sharing most of the neighboring points. Therefore, it is very likely that both patches will result in the same classification output. Yet, they will be processed separately by the network. Now, assume that both patches are classified as containing an object. In this case, they will be propagated to the next module which computes the semantic segmentation scores. Consequently, the points shared by both patches will be classified twice, most likely with the same label – since they have very similar neighborhood configurations. Note how this effect becomes more severe when multiple points share similar spatial locations – as it happens, e.g. for moving vehicles, which generate a dozen of point-reflections. Furthermore, note that every patch which is positively classified leads to an object-box proposal. In this way, assuming that the network predicts the correct class for each patch, every point reflected by an object results in a BB estimation. Though prediction of multiple boxes for the same instance is not a critical problem – they can be filtered by a non-maxima suppression (NMS) algorithm –, this implementation results in a very large number of boxes per object. In addition, the estimated boxes are not provided with a confidence value, thus hardening the filtering process. This also affects the precision of the model, as patches mistakenly classified as vehicle cannot be filtered out by subsequent stages. Finally, though the authors of [92] provided a mechanism for the recognition of objects from different classes, it has not been evaluated, thus leaving a question mark on the ability of the model to detect multiple object classes. Similar considerations can be repeated for the size estimation output, as the dataset used to evaluate the method contained only objects of similar sizes – and only a single object per patch.

It is believed that the shortcomings of the model proposed in [92] can be solved by designing a more efficient network architecture. For instance, it is possible to reduce the amount of redundant processing by semantically segment the whole input point cloud. In this way, it is possible to use the center-point score to classify the patch and the neighbor-points score as semantic segmentation output. For this reason, it has been decided to develop an advanced point-wise processing architecture for multi-class object detection in radar point clouds. The proposed model adopts a two-stage scheme. The first stage uses the pre-processing module developed in [108] and a PointNet++ [64] architecture to semantically segment the input point cloud. In addition, similarly to [114], it regresses direction (or offset) vectors to move the points closer to the object center. The second stage leverages the output of the first stage to define center proposals. It uses a PointNet++ SA layer, tampering the sampling process with the semantic segmentation signal. Finally, the box parameters are estimated by instantiating an anchor-free regression-aided task. Figure 5.2 contains the overview of the model, while the next sections provide a detailed description of its parts.

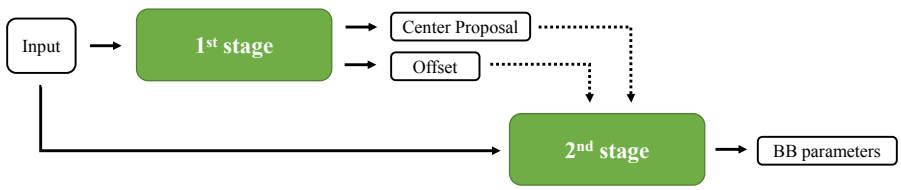


Figure 5.2: Overview of the proposed architecture.

5.2.1 First Stage

Figure 5.3 contains the processing scheme adopted in the first stage of the proposed architecture. It receives as input a radar point cloud in its raw form – i.e. $\boldsymbol{\mathcal{X}} = \{\boldsymbol{x^i} \in \mathbb{R}^d\}$, where d is the number of radar features and $\boldsymbol{x^i}$ the i-th point, $i \in [1..N]$ and $N = |\boldsymbol{\mathcal{X}}|$ the total number of points. This is fed to the pre-processing module developed in [108] which enriches the point-signatures by encoding the information contained in the radar features. It consists in a small shared FC-network and returns as output all the input points, with a deeper feature-set – i.e. $\boldsymbol{\mathcal{X}_p} = \{\boldsymbol{x_p^i} \in \mathbb{R}^{f_p}\}$, where $f_p > d$ is an arbitrary number of features. At this point, the latent point cloud is processed with PointNet++ [64] to capture local structural information. The output of this module consists in a point cloud $\boldsymbol{\mathcal{X}_{Ptn}}$, whose points $\boldsymbol{x_{Ptn}^i} \in \mathbb{R}^f$ maintain the same spatial location of the input ones. However, the feature-descriptor associated with every point encodes the radar properties of the target occupying a specific region as well as relative local context information.

This knowledge is leveraged in the output layer to determine the distribution of the object-of-interest in the scene. In particular, it is addressed a semantic segmentation task, where every point is associated with a class-label. However, the classic conception of the task is altered so that the score associated to every point consists in the probability of the point to be the center of a physical object. In this way, it is possible to exploit the semantic segmentation formulation to produce center proposals. Indeed, points which are classified as background can be discarded, while those associated with a specific class can be used to estimate a BB proposal. In order to teach to the network the center proposal task, every point is labeled as the corresponding object class if it lies in the vicinity of the object center. Therefore, similarly to [101], the area inside the GT boxes is divided into center-region and transition-region, as shown in fig. 5.4A. Points which lie in the center-region are associated with a positive GT class, while those which are located in the transition-region are labeled as background. However, to avoid confusion during training, points in the transition-region are masked out, so that if one of them is predicted as center-proposal the network in not punished. Points outside the transition-region are marked as background. The parameters α_l and α_w control

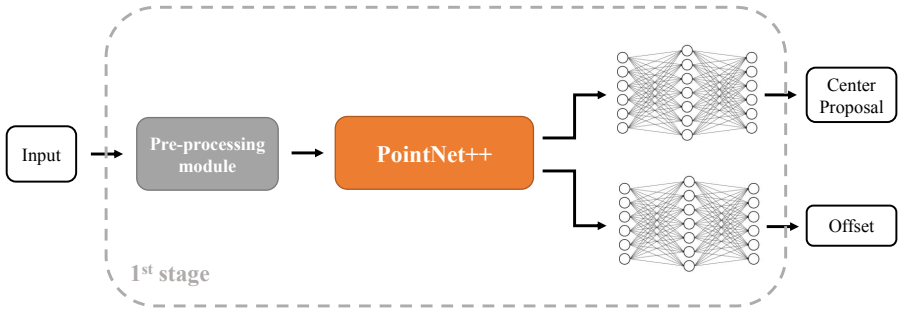
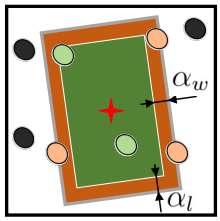
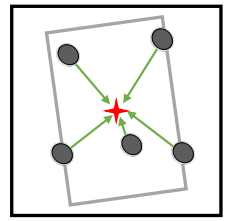


Figure 5.3: First stage of the proposed architecture. The pre-processing module computes deep point-descriptors abstracting the radar features. PointNet++ is used to encode information of local spatial proximity between points. For every point, it is computed a center-proposal score and an offset vector.



(A) Center proposal GT.



(B) Offset GT.

Figure 5.4: GT for learning the outputs of the first stage. The gray rectangles represent GT boxes and the red stars show the object centers. The green and orange colored areas define, respectively, center- and transition-region. Green points are labeled as object centers, while orange points are masked out. Black points are assigned to the background class. The gray color marks generic points. Green arrows show the GT offset associated with each point. Best viewed in color.

the size of the two regions. Note how, when $\alpha_l = \alpha_w = 0$, the center proposal task coincides exactly with the original semantic segmentation task. Furthermore, note how the center proposal task does not require perfect detection scores: it is more important to achieve a high recall than precision. Indeed, since only points labeled as positive would result in BB predictions, if an object has no center proposal, it will not be detected. Contrarily, if a background point is mistakenly proposed as object center, the next stage of the network has the ability to filter it out. Nevertheless, it is fundamental that the network achieves at least a reasonable precision score. Focal loss [110], defined in eq. (3.17), is used to learn this task, in order to deal with the large class imbalance between candidate center proposals and background points.

Besides detecting potential center-proposals, the first stage of the proposed architecture has the ability to refine the object-center estimates. Indeed, it computes an additional output – offset – to predict, for every point, the vector which would shift it from the location where it has been measured to the center of the corresponding object, similarly to [114]. Figure 5.4B contains an example, illustrating the GT used to train this output head. Notice how every arrow originates in a point location and ends at the object center. Background points, which are not associated with any object, receive a null GT vector but are masked out during training. In practice, the network regresses the spatial components of the vector – e.g. in the 2D space, it would estimate the magnitude of the shift along x and y axes. To reduce the complexity of the regression task, it has been decided to design a regression-aided scheme: for the x-component of the vector, the network is forced to output a scalar value, $\tilde{x}_{off}$, in the range $[-1, 1]$; then, the final estimate, $\hat{off}_x$, is computed as

$$\hat{off}_x = s \cdot \tilde{x}_{off} \ , \tag{5.1}$$

where $s > 0$ is a scalar parameter. The same mechanism is repeated on the other vector components, using different, independent network prediction values. Note how, given a value of s large enough, eq. (5.1) does not limit the output representation. Yet, it reduces the search-space of the solution in the range $[-s, s]$, thus facilitating the estimation task. Furthermore, it enables control over the magnitude of the network output. Note, indeed, that the predicted offset vectors are used to

move the points towards the object center. However, the points generally reside within the object boxes, therefore, it is not necessary to have a broad range of possible solutions. Finally, as a side effect, by using the regression-aided formulation, it is also possible to avoid center-proposals to be moved towards different physical instances. The smooth L_1 (or Huber) regression loss [40, 3] is used as objective function to learn the task. It is defined as

$$L_{\delta_H}(e_{rr}) = \begin{cases} \frac{1}{2}e_{rr}^2, & \text{if } |e_{rr}| \leq \delta_H, \\ \delta_H(|e_{rr}| - \frac{1}{2}\delta_H), & \text{otherwise .} \end{cases} , \tag{5.2}$$

where δ_H is a custom parameter – a.k.a. Huber delta – and e_{rr} the error signal. For the offset regression task, the error signal is set to the difference between prediction and GT of every vector component, separately – i.e. $e_{rr}^x = \hat{off}_x - off_x^{gt}$, where $\hat{off}_x$ is the x-component of the offset vector prediction and off_x^{gt} the corresponding GT.

5.2.2 Second Stage

The second stage of the proposed architecture receives as input the very same input point cloud. However, it leverages the output of the first stage to filter out background points and estimate the final object BBs. Figure 5.5 contains the processing scheme adopted at this stage. The first operation consists in shifting the input point locations by the estimated offset vectors. The resulting shifted input, $\boldsymbol{\mathcal{X}_{sh}}$, maintains all the input points, but moves their locations such that points belonging to objects of interest lie closer to the center of the associated box. This point cloud is fed to a SA layer to filter out the center proposals from the first stage and produce a list of object proposals. Three main operations are performed here: sampling, grouping and feature extraction (with PointNet). In order to generate object proposals, however, the sampling process has been tampered. In particular, the center proposal output from the first stage is used as hard-attention signal to select regions of the scene where it is likely to find objects. This is accomplished by means of a masking operation: for each point $\boldsymbol{x_{sh}^i} \in \boldsymbol{\mathcal{X}_{sh}}$, the associated center-proposal score x_{cp}^i is compared with a threshold γ_{cp} to decide whether to maintain or discard it. Formally, this process produces a point cloud $\boldsymbol{\mathcal{S}_{cp}} \subset \boldsymbol{\mathcal{X}_{sh}}$ such that

$$\boldsymbol{\mathcal{S}_{cp}} = \{\boldsymbol{x_{sh}^i} \in \boldsymbol{\mathcal{X}_{sh}} \mid x_{cp}^i > \gamma_{cp}\} . \tag{5.3}$$

Note how, under the assumption that the first stage of the network learns to perform the center proposal task perfectly, $\boldsymbol{\mathcal{S}_{cp}}$ does not contain any background point. However, it still provides a large number of center proposals per object. Therefore, it has been decided to perform a downsampling operation. Since the center point-proposals lie close to the box-centers – they are sampled from the shifted input –, it is possible to use the FPS algorithm. Indeed, in this configuration, the distance between points from different physical instances will be higher than the distance between points of the same instance. Consequently, the FPS algorithm will select at least one point per instance – provided that the required number of points to sample is high enough –, producing an object proposal list $\boldsymbol{\mathcal{X}_{obj}} \subset \boldsymbol{\mathcal{S}_{cp}}$. This can be straightforwardly observed under the assumption of ideal offset predictions – where all center proposal points exhibit a shift that moves them exactly to the respective box-centers. In this case, indeed, FPS will sample all box-centers and, then, duplicate some of them until the required number of samples is achieved.

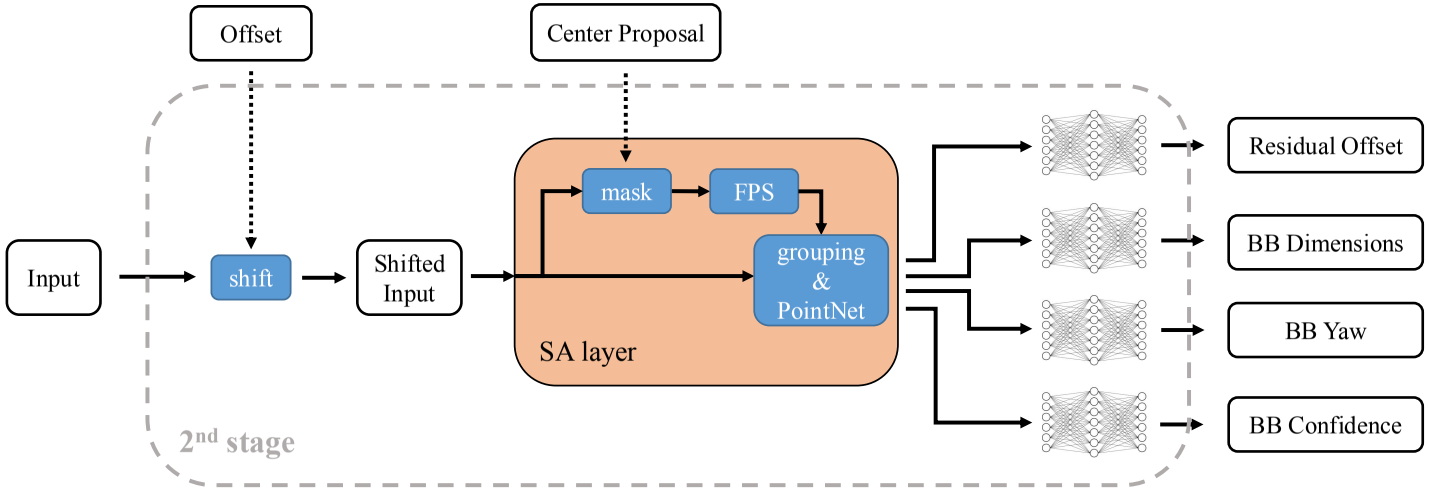
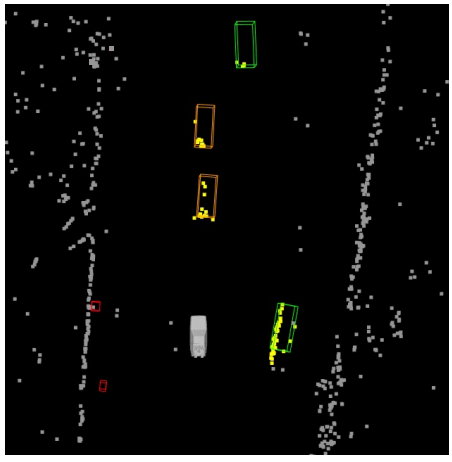
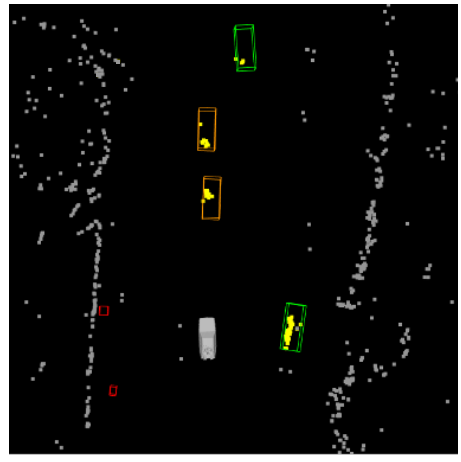


Figure 5.5: Second stage of the proposed architecture. The input is shifted using the offset predictions from the first stage. The center proposal output is used to tamper the sampling process of an SA layer. FPS is used to down-select the center proposals and generate object proposals. The grouping and feature extraction (PointNet) operations compute a feature-descriptor for each object proposal. Four output layers estimate the BB parameters.

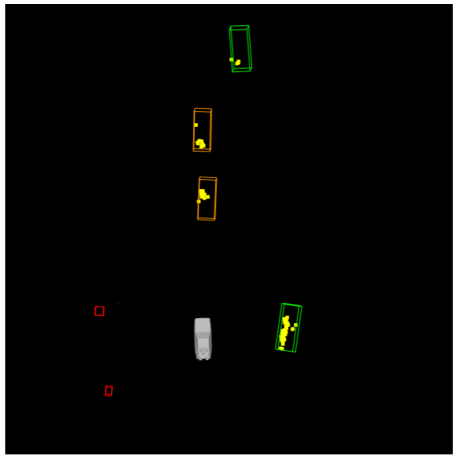
Figure 5.6 shows an example of the effect of the network operations described so far. Figure 5.6A contains a generic input point cloud $\boldsymbol{\mathcal{X}}$ with the associated GT object boxes – see caption for color coding. Figure 5.6B displays the same point cloud, where the points are shifted by the offset predictions – i.e. the shifted input $\boldsymbol{\mathcal{X}_{sh}}$. Note how, compared to fig. 5.6A, the yellow points lie closer to the center of the respective object box, while the locations of the gray points are nearly unchanged. Figure 5.6C shows the effect of the masking operation, producing the point cloud



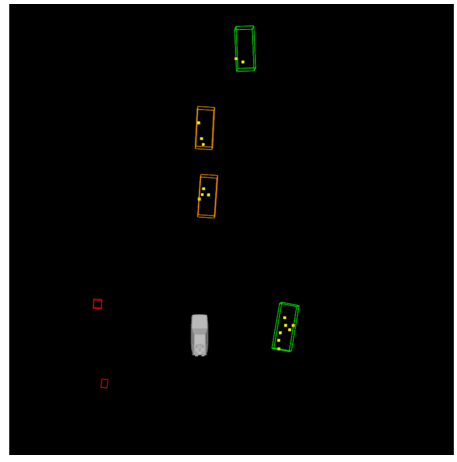
(A) Input – $\boldsymbol{\mathcal{X}}$.



(B) Shifted input – $\boldsymbol{\mathcal{X}_{sh}}$.



(C) Masked shifted input – $\boldsymbol{\mathcal{S}_{cp}}$.



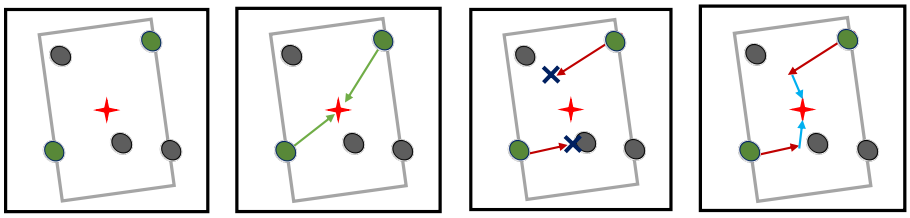
(D) Object proposals – $\boldsymbol{\mathcal{X}_{obj}}$.

Figure 5.6: Visual example of the network operations of the second stage. Points are colored according to the center proposal output of the first stage. Yellow represents vehicle-center, while gray points are classified as background. Boxes correspond to GT objects in the scene. The green color marks moving vehicles, orange for slowly moving vehicles and pedestrian boxes are denoted in red. Best viewed in color.

$\boldsymbol{\mathcal{S}_{cp}}$. Note how only the yellow points survive this process. Finally, fig. 5.6D displays the object proposal points in $\boldsymbol{\mathcal{X}_{obj}}$ which are generated using the FPS algorithm. Note how the point cloud results significantly downsampled, reducing the number of box-predictions per instance from few dozens to a handful. Furthermore, note how the processes described remain effective even under not perfect or ideal predictions of the first stage. Indeed, in fig. 5.6, some points outside the boxes are predicted as centers and they are not shifted exactly to the center of the object. Yet, the second stage proposed herein can discard the background points and down-select the center proposals to generate few object proposals per instance.

The sampling process in the SA layer selects the points which will result in box predictions. Therefore, it is necessary to compute feature-descriptors which contain relevant information for the BB estimation task. This is accomplished by means of the (unaltered) grouping and feature extraction processes of the standard SA layer implementation. In particular, the grouping process collects points in the vicinity of the object proposal ones, while PointNet is used as feature extractor to compute high-dimensional signatures. Note that the shifted input $\boldsymbol{\mathcal{X}_{sh}}$ is used at the grouping stage, therefore, background points are included in the local neighborhoods. In this way, it is possible to encode context information in the feature descriptors. For instance, if a vehicle is passing by a parked car, it is possible to exploit the radar properties of local neighboring points to extract information about the space occupied by the moving object. Moreover, note that the signatures computed at this level are associated with object proposals. Therefore, instead of providing information about an abstract location of the space, they can be considered as containing a description of the object properties. Consequently, it is possible to leverage them to address the box estimation task. The proposed architecture is designed to regress the parameters of oriented BBs, by means of an anchor-free regression-aided formulation. The configuration of the four output heads – for location, size, orientation and confidence – are described in details in the next sections. Finally, in order to detect objects from different classes, the second stage of the architecture is repeated for every output class. In this way, it is possible to fine-tune the network parameters to best accommodate the characteristics of objects from a specific class.

Residual offset The location of the object proposal points are defined by both center proposal output and offset regression. However, the first stage is not perfect and can produce erroneous predictions. For this reason, it has been decided to equip the second stage with a residual offset regression head. Similarly to the offset head in the first stage, it has the task of predicting the vectors which shift the points towards the center of the associated object box. In this way, the network have the ability to fine tune the BB final location estimation, correcting eventual errors from the first stage. Figure 5.7 contains an example that clarifies the definition of residual offset. Figure 5.7A shows a GT object box and the associated radar reflections. Consider, now, that all points are classified as center proposals and that only the green points are selected by the FPS algorithm. In order to produce an accurate box-prediction, they must be moved to the center of the box (red star), as shown by the green arrow in fig. 5.7B. However, offset predictions from the first stage might not be perfectly accurate and contain residual errors. Figure 5.7C illustrates an example, where the red arrows account for the offset predictions and the blue crosses determine the point-locations after the shift has taken place. By means of the residual offset



(A) Center-proposal. (B) GT offset. (C) Predicted. (D) Residual.

Figure 5.7: Example of residual offset. The gray rectangle represents a GT BB and the red star its center. The points are center proposals. The green points are the ones which survive the FPS selection. Green arrows show the GT offset vectors while red arrows display an example of not accurate offset predictions. Blue crosses determine the location of the green points after being shifted. Cyan arrows show residual offset vectors. Best viewed in color.

output, it is possible for the network to adjust the box-location estimate, as shown by the cyan arrows in fig. 5.7D. This output head is configured similarly to the offset prediction head in the first stage. Therefore, the network is forced to output scores in the range $[-1, 1]$, and, then, eq. (5.1) is used to constrain the final predictions in the range $[-s, s]$. The scalar parameter s used for the residual offset predictions is set to half the value used in the first offset regression head. In this way, the network is forced to learn smaller variations. To train the network, the very same loss configuration of the offset regression task is used, using the smooth L_1 objective function [40, 3] in eq. (5.2) and the estimate-GT difference as error signal. The final estimate of the BB locations are defined by the input point locations, the offset predictions of the first stage and the residual offset estimates, i.e.

$$\hat{\boldsymbol{loc}}^{\boldsymbol{i}}_{\boldsymbol{obj}} = \boldsymbol{loc}^{\boldsymbol{j}} + \hat{\boldsymbol{off}}^{\boldsymbol{j}} + \hat{\boldsymbol{res}}^{\boldsymbol{i}} \ , \tag{5.4}$$

where $\hat{\boldsymbol{loc}}^{\boldsymbol{i}}_{\boldsymbol{obj}}$ is the final location estimate for the i-th object in $\boldsymbol{\mathcal{X}_{obj}}$, $\hat{\boldsymbol{res}}^{\boldsymbol{i}}$ the associated residual offset prediction, $\boldsymbol{loc}^{\boldsymbol{j}}$ the location of the j-th input points $\boldsymbol{x}^{\boldsymbol{j}} \in \boldsymbol{\mathcal{X}}$ which contributes to the i object prediction[1] and $\hat{\boldsymbol{off}}^{\boldsymbol{j}}$ the associated offset vector estimate from the first stage.

BB Dimensions The object detection task differs from other formulations as it provides an estimate of the objects' appearance. This is achieved by predicting the shape of the box enclosing a physical instance, i.e. width, length and height. A common way of approaching this task consists in the usage of anchor/template shapes, so that the network can address a classification task to select the configuration which best fits the GT box-shape. However, this formulation needs a large number of templates to predict boxes of various shapes and sizes. Moreover, it requires an additional regression task to refine the final estimate and produce more accurate predictions. In order to avoid limitations of the output representation, it has been decided to address an anchor-free regression-aided problem, similarly to

[1]Bear in mind that $\boldsymbol{\mathcal{X}_{obj}} \subset \boldsymbol{\mathcal{X}_{sh}}$, as both masking and FPS operations reduce the number of points. Therefore the index of the point contributing to the i-th object prediction is not necessarily the same.

the offset regression task. Therefore, for instance, the BB width dimension estimate is computed as

$$\hat{w} = s \cdot e^{\tilde{w}} \ , \tag{5.5}$$

where $s > 0$ is a scalar parameter and $\tilde{w}$ the network output, constrained in the interval $[-1, 1]$. In a similar way, it is possible to obtain the predictions for the other shape parameters. Note that eq. (5.5) differs from eq. (5.1) as the scalar does not directly multiply the network output, but rather its exponential value. That is because negative solutions are not of interest. Furthermore, note how the same output interval can be achieved with eq. (5.1), by limiting the network to generate output values in the interval $[0, 1]$. However, it has been preferred to use eq. (5.5), as it enables a larger range of outcomes than eq. (5.1), given the same scalar value s. For instance, by setting $s = 6$, it is possible to produce estimates in the interval $[2.2, 16.3]$ m circa, which would cover most solutions for, e.g. regressing the length of vehicle objects – considering also long vehicles like buses. In addition, the usage of the exponential function constrains the back-prorogation gradient, thus leading to stabler training and convergence, as shown in [96]. During training, this task is learned by means of the smooth L_1 objective function [40, 3] in eq. (5.2). However, the error signal is calculated using a variant of the bounded IoU loss in [85], i.e.

$$e^{w}_{rr} = 1 - \min\left(\frac{\hat{w}}{w^{gt}}, \frac{w^{gt}}{\hat{w}}\right) \ , \tag{5.6}$$

where $\hat{w}$ and w^{gt} are, respectively, the predicted and GT width values.

BB Yaw In birds-eye view representations, the estimation of box-dimensions does not suffice to fully characterize the appearance of objects in the scene. For this purpose, it is of paramount importance to obtain information about the orientation of the objects. However, the regression of angular quantities is a task which is very difficult to model, due to the periodicity of the target around $\pi/2\pi$. To circumvent this obstacle, the proposed architecture adopts the spherical regression implementation provided in [96]. Liao et al. found out that a generic angular regression task consists in the estimation of a unit vector in the n-dimensional sphere. Therefore, it can be split in the regression of the associated n components. However, since the resulting vector lives on the surface of the n-sphere, the components are not mutually independent, but are bound by an L_2-norm constraint, i.e.

$$\sum_{k}^{n} x_k^2 = 1 \ , \tag{5.7}$$

where x_k is the k-th component of the vector. As an example, consider the task of regressing an angle ϕ in the 2D-sphere – i.e. a circle. It can be performed by estimating two components, $x_1 = \cos(\phi)$ and $x_2 = \sin(\phi)$ such that

$$x_1^2 + x_2^2 = \cos^2(\phi) + \sin^2(\phi) = 1 \ . \tag{5.8}$$

Consequently, inspired by the softmax activation function – commonly adopted in multi-class classification tasks –, they devised the spherical exponential function: an activation function which produces L_2-normalized output scores. Formally, given

the unbounded multi-variate network output vector $\tilde{\boldsymbol{x}}$, with scalar elements $\tilde{x}_j \in [-\infty, \infty]$, it is defined as

$$x_k = S_{exp}(\tilde{x}_k, \tilde{\boldsymbol{x}}) = \frac{e^{\tilde{x}_k}}{\sqrt{\sum_j (e^{\tilde{x}_j})^2}} \ , \tag{5.9}$$

where x_k represents the L_2-normalized estimate of the k-th vector component. Note that eq. (5.9) cannot produce estimates with negative sign, thus enabling only predictions in the positive section of the sphere. In the 2D-circle, it can generate angle estimates, $\hat{\phi}$, which results in both $\cos(\hat{\phi}) \geq 0$ and $\sin(\hat{\phi}) \geq 0$, i.e. $\hat{\phi} \in [0, \frac{\pi}{2}]$. Therefore, the authors proposed to finalize the angle regression task by adding n binary classification problems. In this way, it is possible to assign a sign – plus or minus – to each of the n-components and enable predictions of angles in the whole sphere.

The task solved by the proposed architecture does not require estimation of pitch and roll – i.e. rotation in the XZ or YZ plane – as, in automotive applications, it is very rare to encounter objects with significantly diverse orientations along these planes. However, it is crucial to regress the box-orientation in the XY plane – i.e. yaw –, as it describes how the objects occupy the space. Therefore, it has been decided to limit the BB orientation estimation to solutions in the 2D-circle. Furthermore, since the estimated yaw is used only to rotate the box predictions, it is possible to reduce the search space to the semi-circle, specifically in the interval $[-\frac{\pi}{2}, \frac{\pi}{2}]$, as the same rotation can be obtained by adding/subtracting multiples of π. Consequently, the yaw regression head is configured to produce three outputs: the two components of the angle (i.e. cosine and sine) in absolute value and a single binary classification problem. Note, indeed, that it is sufficient to predict the sign of $\sin(\phi)$, as $\cos(\phi) \geq 0 \ \forall \ \phi \in [-\frac{\pi}{2}, \frac{\pi}{2}]$. Formally, the network uses the spherical exponential function in eq. (5.9) to convert unbounded predictions $\tilde{x}_\theta$ and $\tilde{y}_\theta$ into L_2-normalized component estimates, i.e.

$$\hat{cos}_\theta = S_{exp}\big(\tilde{x}_\theta, [\tilde{x}_\theta, \tilde{y}_\theta]^T\big) = \frac{e^{\tilde{x}_\theta}}{A} \quad \text{and} \quad \hat{sin}_\theta = S_{exp}\big(\tilde{y}_\theta, [\tilde{x}_\theta, \tilde{y}_\theta]^T\big) = \frac{e^{\tilde{y}_\theta}}{A} \ , \tag{5.10}$$

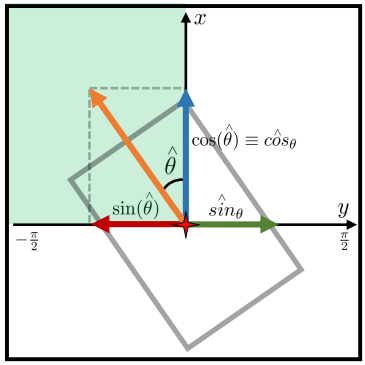
where $A = \sqrt{e^{2\tilde{x}_\theta} + e^{2\tilde{y}_\theta}}$ is the L_2-normalization factor. For the classification output, instead, the network produces a score $\hat{\theta}_{cls} \in [0, 1]$ by means of the sigmoid activation function. Then, the final BB yaw estimate is computed as

$$\hat{\theta} = \text{sign}\big(2 \cdot \hat{\theta}_{cls} - 1\big) \cdot \arctan\left(\frac{\hat{sin}_\theta}{\hat{cos}_\theta}\right) . \tag{5.11}$$

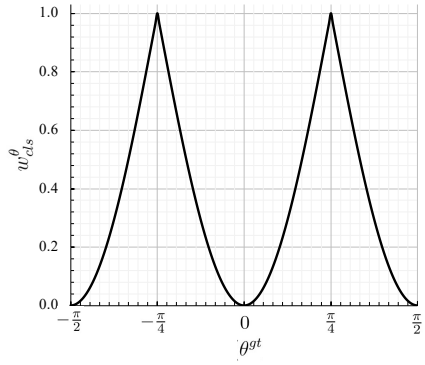
Figure 5.8A contains a visual example showing the operations that lead to yaw regression. Note how the cosine estimate $\hat{cos}_\theta$ always coincides with the cosine of the estimated angle $\cos(\hat{\theta})$ (blue vector). Instead, the sine prediction $\hat{sin}_\theta$ (green vector) requires the classification output (green area) to fully determine the sine of the estimated angle $\sin(\hat{\theta})$ (red arrow).

During training, the regression task is learned by means of the smooth L_1 objective function [40, 3] in eq. (5.2). The mean squared error (MSE) between prediction and absolute GT value is used as error signal, i.e.

$$e^{\theta}_{rr} = \frac{1}{2}\big(\hat{cos}_\theta - |\cos(\theta^{gt})|\big)^2 + \big(\hat{sin}_\theta - |\sin(\theta^{gt})|\big)^2 \ , \tag{5.12}$$



(A) Visual example of the yaw-regression network operations.



(B) Values produced by the dynamic weight function in eq. (5.13).

Figure 5.8: Yaw-regression head. The gray box represents an object and the red star its center. The orange vector shows the yaw-orientation of the box. Blue and green arrows display the network predictions. The green area represents the classification output which permits the estimation of the red arrow. Best viewed in color.

where $\theta^{gt} \in (-\frac{\pi}{2}, \frac{\pi}{2}]$ is the GT yaw angular value. Note how, eq. (5.12) forces the regressed values to approximate the absolute value of cosine and sine of the angle. The classification task, instead, adopts the cross-entropy objective function [2], which is defined in eq. (3.17), by setting $\alpha_t = 1$ and $\gamma = 0$. Additionally, it has been observed that the effect of an erroneous classification output on the final estimate varies based on the angle value. Indeed, for angles close to the classification boundaries – i.e. $-\frac{\pi}{2}$, 0 and $\frac{\pi}{2}$ – it has a lower impact than for those in the core section of the classification region – i.e close to $-\frac{\pi}{4}$ or $\frac{\pi}{4}$. To visualize this, assume, for instance, a correct regression output: for an angle $\theta^{gt} = 5°$, a wrong classification output would produce a prediction $\hat{\theta} = -5°$ which results in an error of 10°; contrarily, for an angle $\theta^{gt} = \frac{\pi}{4} = 45°$, an erroneous classification output would generate an estimate $\hat{\theta} = -45°$ which is 90° away from the correct value. For this reason, it has been designed a dynamic weighting factor

$$w^{\theta}_{cls} = 2 \cdot \left[1 - \max\left(\sin^2(\theta^{gt}), \cos^2(\theta^{gt})\right)\right], \tag{5.13}$$

that scales down the classification output when close to the decision boundaries. Note how, the proposed weight finds its minima of 0 in $\theta^{gt} = m \cdot \frac{\pi}{2}$ and its maxima of 1 when $\theta^{gt} = (2m+1) \cdot \frac{\pi}{4}$ – where $m \in \mathbb{Z}$, the integer set. Figure 5.8B contains a plot showing the value of the weight from eq. (5.13) in the interval $[-\frac{\pi}{2}, \frac{\pi}{2}]$. Consequently, the BB yaw regression head is trained with a cumulative loss calculated as

$$L_\theta = L_{\delta_H}(e^{\theta}_{rr}) + w^{\theta}_{cls} \cdot \mathrm{CE}(\hat{\theta}_{cls}, \theta^{gt}_{cls}) \tag{5.14}$$

where $\mathrm{CE}(\cdot,\cdot)$ represents the cross-entropy loss function [2] and θ^{gt}_{cls} the GT class associated with the angle θ^{gt} – i.e.

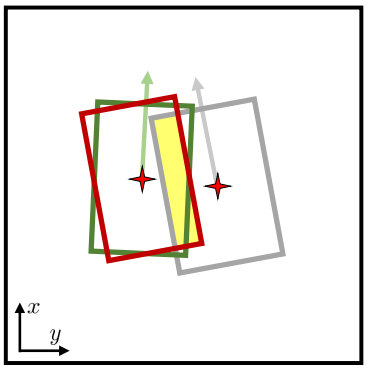
$$\theta^{gt}_{cls} = \begin{cases} 1, & \text{if } \theta^{gt} \in (0, \frac{\pi}{2}], \\ 0, & \text{otherwise .} \end{cases} \tag{5.15}$$

In this way, the network will not be penalized for wrong prediction in proximity of the decision boundaries and can focus on regressing the correct sine/cosine values.

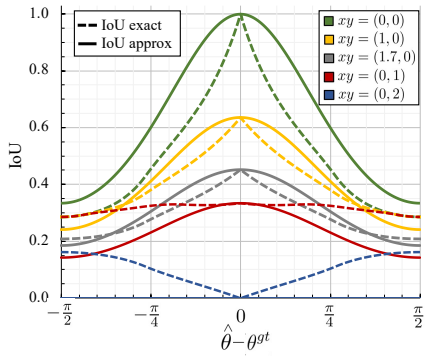
BB Confidence The object proposal list is severely dependent on the outputs of the first stage. In particular, it is the center proposal signal that defines the points which are candidates to become objects. However, since the center proposal output is not perfect, it might mistakenly propose, as center candidates, points assigned to the background class – i.e. not belonging to any object class. In addition, though the second stage has the ability to sub-select the center proposals, it still results in multiple box-predictions per object, as shown in fig. 5.6D. These aspects have led to the decision to equip the network with a BB confidence output, providing a score which describes how confident the network is about the correctness of each box-prediction. In this way, it is possible to divide the burden of detecting the object-locations between first and second stage. Indeed, the confidence output enables the network to discard wrong object proposals, thus filtering eventual errors from the center proposals output. Moreover, it provides critical information for sorting out multiple box-predictions for the same object, particularly useful in post-processing algorithms such as non-maxima suppression (NMS).

From an implementation perspective, the BB confidence head requires, for each box-proposal, the prediction of a scalar value in the interval $[0, 1]$. In order to teach the network to generate box-confidence outputs, it is necessary to define both the problem and the associated GT labels. In this work, it has been decided to address a binary classification task and use the Intersection-over-Union (IoU) score between GT and estimated BBs to determine the class-label. In particular, for each box-prediction, it is computed the IoU score with every object present in the scene. Then, if the maximum score results higher than a given threshold $\gamma^{+}_{conf} \in [0, 1]$, the prediction is associated with a positive label; otherwise, it is assigned to the negative class. As a result, the network is trained to predict high confidence-scores for boxes with large IoU. Additionally, to avoid confusing the network, it has been decided to mask out boxes whose IoU score lies in the interval $[\gamma^{-}_{conf}, \gamma^{+}_{conf}]$, where γ^{-}_{conf} is a suitable threshold such that $0 \leq \gamma^{-}_{conf} \leq \gamma^{+}_{conf}$. In this way, the network is not penalized for erroneously predicting an high-confidence score for such box-predictions. Finally, note how, despite the BB confidence head is trained using the classification formulation, its output is not thresholded to produce a class estimation. Instead, the score is adopted as confidence value. The focal loss function [110] defined in eq. (3.17) is used to learn this task.

The last remark concerns the computation of the IoU score. Indeed, though this task is easily executed when the boxes are aligned along the same axis, it is not trivial in the case of rotated boxes. Additionally, note that, since the parameters of the predicted boxes are not known a priori, the IoU has to be computed live during training, thus preventing the usage of advanced time-consuming methods. For this reason, it has been developed a low-cost empirical solution that approximates the IoU of unaligned BBs. It consists in a three-steps process. First, the intersection area of aligned BBs, INT_{align}, is computed. The boxes are built using GT and estimated location and size parameters. However, both boxes are aligned along the angular direction specified by the GT yaw parameter. Figure 5.9A shows a visual example for the computation of the aligned intersection area. Note how the red box (aligned predicted-box) maintains both position and size of the green box (estimated) but is



(A) Illustration about the computation of the aligned intersection area.



(B) Comparison between exact IoU and approximation proposed in eq. (5.17).

Figure 5.9: Computation of the approximated IoU. The gray and green boxes represent, respectively, the GT and estimated BBs. Arrows display the orientation of the boxes. The red box shows the estimated BB after alignment with the GT yaw. The yellow area represent INT_{align}. The solid and dashed lines in the plot show, respectively, the approximated and exact IoU score. Every line corresponds to a different configuration. Size estimation is considered exact – i.e. both GT and estimated box are $4.5m$ long and $2m$ wide. The position of the predicted box is changed by the values reported in legend – e.g. $xy = (1, 0)$ indicates that the predicted box has a $1m$ estimation error along the x-axis. Best viewed in color.

aligned with the gray box (GT). The second step consists in the computation of a scaling factor w^{IoU} that down-weights INT_{align} based on the correctness of the yaw prediction. Formally, it is calculated as the squared cosine value of the error angle, scaled to fit in the range $[0.5, 1]$, i.e.

$$w^{IoU} = 0.5 + 0.5 \cdot \cos^2(\hat{\theta} - \theta^{gt}) \ . \tag{5.16}$$

Finally, the approximated IoU value is obtained, dividing the approximated intersection area by the union area – computed as the sum of the two box-areas minus the approximated intersection – i.e.

$$IoU_{approx}(\hat{\boldsymbol{box}}, \boldsymbol{box_{gt}}) = \frac{INT_{align} \cdot w^{IoU}}{\text{area}(\hat{\boldsymbol{box}}) + \text{area}(\boldsymbol{box_{gt}}) - INT_{align} \cdot w^{IoU}} \ , \tag{5.17}$$

where $\hat{\boldsymbol{box}}$ and $\boldsymbol{box_{gt}}$ contains, respectively, the estimated and GT box parameters – i.e. location, size and yaw – and area$(\cdot)$ computes the area of a box by multiplying its dimensions. Figure 5.9B provides a comparison between the proposed IoU score approximation and an exact version – from the package SHAPELY [23]. The IoU score is plotted against different values of the yaw error $\hat{\theta} - \theta^{gt}$. Different colors show various configurations of the position of the predicted box w.r.t the GT one. Note how, when the estimated BB location is correct or has an offset along the x-axis (long dimension), the proposed approximation resembles the exact IoU score (green, yellow and gray lines). The red curves show the scores produced by a prediction with center located at the boundary of the GT BB, along the y-axis – i.e. the small

dimension. Note how, in this configuration, the approximated and exact IoU diverge: the solid line decays while the dashed line remains almost constant. However, this effect is not critical. Indeed, the exact IoU maintains its score value because an increase in the yaw-error pushes the front (or back) part of the predicted box to get inside the GT one. This observation is supported by the blue dashed line. Here, when the two boxes are aligned – i.e. $\hat{\theta} - \theta^{gt} = 0$ –, they do not overlap. Yet, as the estimated yaw becomes less correct, the exact IoU increases, achieving a maximum when the two boxes are perpendicular – i.e. $\hat{\theta} - \theta^{gt} = \frac{\pi}{2}$. As a result, it is possible to conclude that, compared to the exact IoU, the proposed approximation gives more importance to the correctness of the estimated parameters. Since eq. (5.17) is used to generate GT labels for the confidence output head, this is an appreciated effect. Indeed, it will tend to label with the positive class, predictions which provide accurate estimates of the box-parameters. Finally, note that the proposed IoU approximation is better-suited for live computation, as it runs in less than $6.5\mu s$, while the exact IoU computation requires circa $200\mu s$.

5.3 Experimental Setup

This section provides details about the settings adopted during the experimental procedure. The proposed network architecture is challenged with the recognition of objects belonging to two classes: moving vehicle and moving pedestrian. In particular, the 2D object detection task is addressed. Therefore, the network estimates the XY box-location, length, width and yaw. It has been decided to use precision, recall and F_1 score, defined in eq. (3.16), to evaluate the object detection performance. When possible, precision-recall (PR) curves are reported, in order to obtain results which are invariant from confidence threshold values. The ability of the network to estimate the BB parameters – i.e. location, size and orientation – is analyzed using IoU scores. The exact computation provided by the SHAPELY package [23] is used. In addition, the accuracy of the network to estimate the orientation of the objects is evaluated, using the root mean squared error (RMSE) between predictions $\hat{\theta}$ and the corresponding GT values θ^{gt} – i.e.

$$RMSE(\hat{\theta}) = \sqrt{\mathbb{E}\left[\left(\hat{\theta} - \theta^{gt}\right)^2\right]}\ , \tag{5.18}$$

where $\mathbb{E}[\cdot]$ represents the expectation operator. Since RMSE gives more emphasis to large errors, it is preferred to the mean average error (MAE), computed as

$$MAE(\hat{\theta}) = \mathbb{E}\left[|\hat{\theta} - \theta^{gt}|\right]\ . \tag{5.19}$$

Note, indeed, how a yaw estimation off by few degrees still provides a box-orientation which is similar to the GT one. On the other hand, a large yaw error significantly affects the perception of the object, especially when the length-width ratio is large. Consequently, small yaw errors have a minor impact and, therefore, can be more tolerated than large ones, which are particularly undesired.

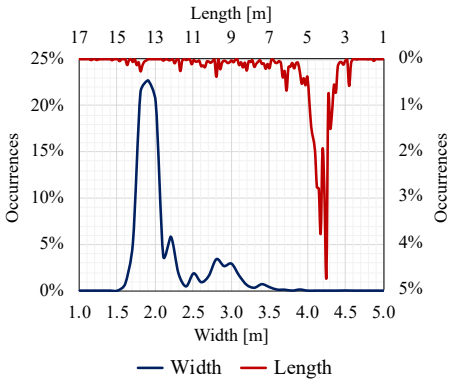
5.3.1 Dataset

The dataset used to evaluate the proposed architecture is the same used in chapter 4. A full description is provided in sections 3.3.1 and 4.3.1. The input data

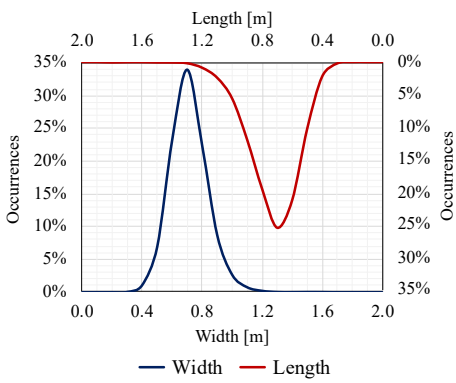
are radar point clouds, where each point is identified by 4 features: i.e. the XY spatial coordinates, ego-compensated Doppler and RCS. However, differently than section 4.3.1, every point is not associated with an object class. Indeed, the GT labels consist in a collection of boxes with related parameters. In particular, every frame is associated with a list of box-parameters, representing the objects present in the scene at the time of recording. Table 5.1 reports statistics about the GT objects in the dataset – both train and test-set. Note that it contains nearly $700K$ instances from the vehicle class (for an average of 10 per frame) and more than $200K$ from the pedestrian class (an average of 3 per frame). However, only less than $150K$ instances per class are moving. Heavy urban environments are covered by the dataset: there are scenes with almost 80 pedestrians in a single frame or nearly 25 moving vehicles. Other statistics reported in table 5.1 concern the size of the GT objects present in the dataset – i.e. length and width. Note how the vehicle class is not limited to conventional vehicles, but extends to large vehicles like buses or trucks –

Table 5.1: Statistics of box-labels from vehicle and pedestrian objects in the dataset (train and test set). The count columns report the total amount of instances in the dataset as well as the average and maximum per-frame count. The length and width columns, instead, contain information about the object sizes (in meters). Statistics of both moving and stationary objects (All) as well as only moving objects (Mov.) are reported.

		Count [/]			Length [m]			Width [m]		
Class		Total	Avg	Max	Avg	Max	Min	Avg	Max	Min
Vehicle	All	699.4K	10	52	4.55	16.42	1.78	1.93	3.28	1.21
	Mov.	147.6K	2	23	5.13	16.42	1.78	2.06	3.21	1.21
Pedestr.	All	215.8K	3	78	0.65	1.84	0.37	0.65	1.62	0.33
	Mov.	147.7K	2	45	0.68	1.84	0.37	0.66	1.62	0.33



(A) Vehicle objects.



(B) Pedestrian objects.

Figure 5.10: Distribution of the size of BB from different classes. The blue lines refer to the width parameter, while the red lines report statistics about the length. Best viewed in color.

as shown by the large maximum length value. More details about the dimensions of boxes from different classes are provided in fig. 5.10. It reports the distributions of length and width for objects of different classes. Note that vehicle instances show diverse distributions than pedestrians, due to the variety of objects encompassed by the vehicle class. Indeed, as shown in fig. 5.10A, the former result in Poisson-like distributions, with most of the values concentrated around common vehicles configurations – i.e. length in $[3.5, 5.3]$ m and width in $[1.6, 2.3]$ m. The latter, instead, present Gaussian distributions, with mean located in $0.7m$ and standard deviation of $0.17m$ and $0.13m$, respectively, for length and width, as reported in fig. 5.10B.

5.3.2 Data-Association and Post-processing

The output of the proposed architecture consists of a list of objects with associated box-parameters. However, as it usually happens in object detection tasks, the output is not directly matched with GT labels – contrarily to, e.g. classification and semantic segmentation tasks. For this reason, in order to evaluate the performance of the model, it is necessary to pair predicted boxes with GT objects. This process is generally called data-association and represents the solution of an assignment problem. Although there exist various advanced algorithms for this task – e.g. Hungarian algorithm [1] –, in this work, it has been decided to implement a simpler method[2]. In particular, it has been adopted an overlap-based method, where each GT object is associated with the box-prediction which shows the highest overlap. Consider, for a given frame, a set of M GT boxes and K predicted boxes. The data-association process creates a matrix $\boldsymbol{C} \in \mathbb{R}^{M \times K}$ whose rows contain the overlap scores between a single GT box and all K predicted boxes. Therefore, the element of the i-th row and j-th column of $\boldsymbol{C}$ – i.e. c_{ij} – holds the overlap between the i-th GT object box, $\boldsymbol{box_{gt}^{i}}$, and the j-th estimated box, $\hat{\boldsymbol{box}}^{\boldsymbol{j}}$, computed as

$$c_{ij} = \max\Big(overlap\big(\boldsymbol{box_{gt}^{i}}, \hat{\boldsymbol{box}}^{\boldsymbol{j}}\big), overlap\big(\hat{\boldsymbol{box}}^{\boldsymbol{j}}, \boldsymbol{box_{gt}^{i}}\big)\Big) \ , \tag{5.20}$$

where

$$overlap\big(\boldsymbol{box_1}, \boldsymbol{box_2}\big) = \frac{\widetilde{area}\big(\boldsymbol{box_1} \cap \boldsymbol{box_2}\big)}{\widetilde{area}\big(\boldsymbol{box_1}\big)} \ , \tag{5.21}$$

with $\widetilde{area}(\cdot)$ the function which returns the area inside the input polygon. Note how, since the function in eq. (5.21) is sensitive to the position of the arguments, the maximum operator in eq. (5.20) ensures that, in case multiple box-predictions for the same instance are present, the smallest box with the highest overlap will be selected. At this point, the data-association process analyzes, sequentially, the rows of $\boldsymbol{C}$ to match GT and prediction BBs. In particular, starting from the first row, it selects the estimated box which results in the highest overlap with the row-related GT box. Formally, given the i-th row, it performs the matching $\Big(\boldsymbol{box_{gt}^{i}}, \hat{\boldsymbol{box}}^{\boldsymbol{j^*}}\Big)$, with

$$j^* = \arg\max_j \boldsymbol{C_i} = \arg\max_j \{c_{ij} \ \forall \ j \in [1..K]\} \tag{5.22}$$

[2]Bear in mind that, compared with a standard assignment problem – e.g. job-worker association, where any job can be assigned to any worker –, here, each box-prediction either represents a GT object or none, i.e. it cannot be used to represent multiple real instances. The critical case – occurring when a box-prediction overlaps with two GT objects which are very close to each other – is too rare to justify the adoption of more advanced techniques.

where $\boldsymbol{C_i}$ is the vector containing the K elements in the i-th row of $\boldsymbol{C}$. However, before completing the matching, the score c_{ij^*} is compared with a threshold γ_{da}. Then, if $c_{ij^*} \geq \gamma_{da}$, the matching is executed and both the i-th row and the j^*-th column of $\boldsymbol{C}$ are zeroed – to avoid associating predicted boxes with multiple labels and vice versa. Otherwise, the matching is discarded and the process is repeated for the next row. Once all rows are analyzed, the data-association process checks if there are boxes not matched and associates them with the background class. Therefore, free GT boxes are considered missed detections (or false negatives), thus affecting the recall score. Unmatched predicted boxes, instead, are accounted for as wrong detections (false positives) and harm the precision of the network. In the proposed evaluation, for the vehicle class it has been set $\gamma_{da} = 0.1$. However, for pedestrian instances, it has been decided to set $\gamma_{da} = 10^{-5}$, to allow matching of boxes that barely touch each other. Note how, since objects of this class have relatively small dimensions, this operation does not result in large position errors.

The data-association procedure described so far is not well-suited for methods that produce multiple box-proposals per object-instance. Indeed, among the several box-predictions, only one will be matched with the GT object. All the others would be considered false positives, thus significantly harming the precision score. For this reason, it has been decided to adopt an NMS algorithm to filter out multiple predictions for the same object. It operates by selecting the boxes with the highest confidence values which have a small overlap with each other. In particular, it first sorts the object-predictions list based on the confidence value predicted by the network. Boxes with confidence values below a given threshold, γ^{conf}_{NMS}, are discarded. Then, the first box in the input predictions-list – i.e. the most confident prediction – is selected and added to the output filtered-list. Subsequently, eq. (5.20) is used to compute the overlap between the next element in the input list and all the elements in the output list. If the maximum overlap does not exceed the value of a given threshold, γ^{ovl}_{NMS}, the element is added to the output list; otherwise it is discarded, since the algorithm considers it as describing an instance already present in the output list. This process is repeated for all the elements of the sorted input list, to decide whether to maintain or reject the object-proposal. Consequently, the output list will contain boxes which overlap with each other by less than γ^{ovl}_{NMS}. In this work, it has been decided to set $\gamma^{conf}_{NMS} = 0.3$ and $\gamma^{ovl}_{NMS} = 0.1$ for object-predictions of both classes. Finally, note that the NMS algorithm described so far is used as a post processing step and it is out of the scope of this report.

5.3.3 Settings

This section provides details about the network configurations. The data-settings described in section 3.3.2 are adopted. Therefore, the network receives as input a raw point cloud containing 1200 radar reflections collected across a window of 200 ms – i.e. 4 frames. The point locations are defined by the XY spatial coordinates. Furthermore, the ego-compensated Doppler value is used as z-coordinate. The RCS measurement is used as point-feature.

The network architecture is split into two stages. The first stage uses a pre-processing module with three shared FC layers of size 8, 16 and 32 channels, respectively. PointNet++ is designed with two encoder-decoder layers. The first SA layer samples 500 representative points using the FPS method. Then, it collects neighbor

points in three areas, i.e. 1, 1.5 and 2 meters. Finally, PointNet is used as feature extractor to produce signatures of 32, 32 and 64 channels, respectively. The second SA layer receives as input the latent point cloud of 500 points with $3+128$ features. It samples 150 representative points, groups neighborhoods areas of 4, 6 and 8 meters and produces point-descriptors of 64, 64 and 128 channels, respectively. The decoder section uses FP layers to obtain signatures for the original point locations. The first one uses shared FC layers of size 128 channels, the second produces signatures of 64 features. Since this section of the architecture generates two outputs – i.e. center proposal and offset – it is equipped with two FC classification networks. Both of them use two layers of size 64 and 32 channels, respectively, interleaved by dropout with rate 0.5. A single shared FC unit is used to produce a score out of each point-descriptor. The sigmoid activation function is then used to compute the probability of each point being a center of an object from a specific class. A similar processing takes place at the offset regression head. However, the hyperbolic tangent activation function is adopted to constrain the network output in the range $[-1, 1]$. Moreover, it has been decided to produce an offset regression output per class. Therefore, different configurations of the scalar parameter s in eq. (5.1) are used. Specifically, for vehicles it is set $s = 10m$, while for pedestrians $s = 2m$. Note how this setting helps the network to focus on the predictions of small offsets for center proposals of pedestrian instances. The center proposal head is trained using focal loss with $\alpha = 0.85$ for vehicles and $\alpha = 0.9$ for pedestrians, while $\gamma = 2$ for both. Due to the sparsity of the radar data, it has been decided to consider as center-region the whole GT BB area and move the transition-region, depicted in fig. 5.4A, outside the box – i.e. $\alpha_l < 0$, $\alpha_w < 0$. In this way, the network will not be penalized for selecting as center-proposal points nearby the object boundaries. This is particularly effective for fast moving objects which generate tails of reflections, when measurements from multiple frames are fused together. Consequently, for vehicles, it has been set $\alpha_l = \alpha_w = -1m$. Instead, for pedestrians, it has been decided to extend the center-region to cover the area of $0.3m$ around the objects. Then, the transition-region with parameters $\alpha_l = \alpha_w = -1m$ is considered. This operation is crucial, as it allows the detection of instances which have no reflection in the BB – bear in mind that pedestrians have limited box-sizes, therefore, a measurement error can easily move the reflections outside the box. In order to learn the offset regression task, it is used the smooth L_1 loss with $\delta_H = 0.2$ for vehicles and $\delta_H = 0.1$ for pedestrians. The common parts of the first stage – i.e. pre-processing module and PointNet++ – are trained using both objective functions, while the classification networks and output layers are trained using the respective losses.

The second stage leverages the outputs of the first stage to produce BB predictions. Every class has its own second stage network and uses the center proposal output and the corresponding offset. The vehicle network uses eq. (5.3) with threshold $\gamma_{cp} = 0.5$ to select the center proposals. Half the number of input points is allowed to survive the masking operation. If more are provided, the ones with highest score are selected; otherwise the first point is repeated to fill the list. The FPS algorithm is set to select 40 points from the masked center proposal list. Therefore, a maximum of 40 moving vehicle objects can be predicted by the network. Two neighborhood areas – 1 and 2 meters – are used to group nearby points and compute descriptors of 128 channels each. Therefore, the tampered SA layer produces an object list of 40 points with 256-features signatures. This signal is used to estimate the BB parameters. In

particular, as specified in section 5.2.2, the network is equipped with four output heads. Every head presents its own FC classification network, using two layers of 128 and 32 channels and dropout with rate 0.5. The residual offset head produces two scalar per point, using the hyperbolic tangent activation function and the parameter $s = 5m$. The smooth L_1 loss is used to train this task, with $\delta_H = 0.1$. The dimensions head estimates length and width of the objects. Therefore, for each point, it produces two scalar values in $[-1, 1]$ using the hyperbolic tangent activation function. Then, it sets the scalar parameter s in eq. (5.5) to $2.5m$ for width estimation and $6m$ for length estimation. This represents the best compromise to cover most of the cases provided in the dataset. Note, indeed, how the predicted width value $\hat{w} \in [0.92, 6.8]$ m and the length prediction $\hat{l} \in [2.2, 16.3]$ m. This task is trained with the smooth L_1 objective function with $\delta_H = 0.5$ for the length parameter and $\delta_H = 0.3$ for width. The BB orientation head produces three scalar outputs per point. The regression outputs use the spherical exponential function in eq. (5.10), while the binary classification score is processed with the sigmoid activation. They are trained using the smooth L_1 loss, with $\delta_H = 0.3$, and the cross-entropy loss with dynamic weight defined in eq. (5.13). Finally, the confidence head uses the sigmoid activation function and it is trained with focal loss. The parameter α is set to 0.65, following an empirical investigation, while $\gamma = 2$. The label generation process uses eq. (5.17) to compute the IoU score and threshold values $\gamma^{-}_{conf} = 0.2$ and $\gamma^{+}_{conf} = 0.35$.

The second stage used to estimate pedestrian BBs adopts a similar configuration to the vehicle network. However, due to the different nature of the objects, it is possible to simplify the task. Unless otherwise specified, the same settings used for the vehicle network are adopted. The number of points allowed to survive the masking operation has been limited to a quarter of the total amount of input points. This is because, though there might be a large number of pedestrian instances in a single frame, each object produces only few radar reflections. The number of object proposal points to sample with FPS has been increased to 50 to cover very dense scenes present in the dataset. Finally, the small sizes of such instances have led to a reduction of the neighborhood search areas to 0.5 and 1 meter. The BB parameter regression task is facilitated by reducing the number of output heads. Indeed, since pedestrian objects often result in squared boxes of small sizes, the estimation of the box-orientation is not critical. For this reason, it has been decided to utilize cylinders to represent pedestrians, instead of boxes. Consequently, it is possible to discard the yaw regression head – as cylinders do not have orientation in the circular plane. Additionally, it has been observed that pedestrians share common size configurations. Indeed, as shown in fig. 5.10B, both width and length of the instances present in the dataset are Gaussian-distributed with small standard deviations. Therefore, it has been decided to use fixed size values and discard the dimension regression head. Specifically, the diameter of the cylinder has been set to the mean value of the Gaussian distributions in fig. 5.10B – i.e. $0.7m$. A crucial task for the detection of pedestrian objects is the position estimation. Therefore, the network maintains the residual offset regression, setting the scalar $s = 1m$. The smooth L_1 loss is used during training, with $\delta_H = 0.1$. Finally, a confidence prediction is generated. However, since both size and orientation of the box are not regressed, it has been decided to use a simpler distance-based metric to generate the GT labels. In particular, the euclidean distance between the center of the

estimated object and all the GT objects is computed. Then, if the smallest distance is lower than a threshold (γ^{+}_{conf}) the positive class is assigned to the box-prediction; otherwise, it is labeled as negative sample. A transition region is also designed, masking out objects whose distance is in the interval $[\gamma^{+}_{conf}, \gamma^{-}_{conf}]$. It has been set $\gamma^{+}_{conf} = 2m$ and $\gamma^{-}_{conf} = 3m$. Focal loss is used to train this task, with $\alpha = 0.8$. Each second stage network is trained using the loss from the corresponding output head. In particular, the common section – i.e. the tampered SA – is trained from all the output heads, while the classification networks are trained with the corresponding objective function. Finally, note that, since the two second stage networks are lightweight and independent from each other, their impact on the inference time is limited, thus enabling the addition of supplementary classes.

The training procedure is split in two steps. Unless otherwise specified the learning rate is set to 0.001 and the batch size to 8. In the first step, the first stage is trained. In particular, first the center proposal head is trained for 8 epochs. Then, the offset regression heads are trained for 6 epochs. Finally, the network is fine tuned using both objectives for 15 epochs and a lower learning rate – i.e. 0.0002. The second training step freezes the first stage of the architecture – i.e. does not allow its parameters to be changed. Then, it trains the network using the objective of each output head, sequentially, for 5 epochs. When the BB dimensions and yaw heads are trained for the vehicle network, the pedestrian network does not train. Finally, the second stages are fine tuned for 15 epochs, with learning rate 0.0002, using all the objective functions.

The architecture configuration described so far is referred to as **4frames**. In addition, the proposed architecture is evaluated by using the point clouds generated in a single radar frame – i.e. 50 ms window. In this way, it is possible to investigate how the input density affects the ability of the proposed solution to perform the object detection task. This configuration is referred to as **1frame**. The architecture uses the same settings of the 4frame counterpart. However, few modifications are adopted to address the lower input density. In particular, in the first stage, the number of representative points in the PointNet++ SA layers is reduced to 200 and 100. Moreover, the focal loss parameter α for the vehicle center proposal task is set to 0.8. Finally, the focal loss parameter α for the confidence estimation of pedestrian boxes is set to 0.9.

5.3.4 Benchmark

In order to fully assess the effectiveness of novel solutions, it is best practice to compare them with a benchmark. In the radar literature, there is only one method that produces box-proposals by consuming raw point clouds [92]. The model has only been tested on a single class and the effect of the addition of multiple object-classes have not been investigated. Therefore, it has been decided to use the implementation proposed in the original paper, addressing only the moving vehicle detection task. The width and depth of the network has been reduced to enable fair comparison with the proposed architecture. The model proposed in [92] consists of three stages. The patch proposal stage operates by cropping out a local point cloud around each input point. A square area of $10m \times 10m$ is used. The second stage receives local patches sequentially and exploits PointNet to produce classification scores. It uses two shared FC layers of size 64 and 64 channels to compute point-related signatures.

Then, three additional shared FC layers of size 64, 128 and 256 channels compute the point-descriptors which are max-pooled to obtain the global signature. This latter is processed with a FC classification network to generate, for each patch, a class score. It is implemented as two FC layers of size 128 and 64 channels, followed by the output layer - a single FC unit. The sigmoid activation function is used and the task trained with focal loss – $\alpha = 0.6$. The patches classified as vehicle are semantically segmented. This is accomplished by concatenating the patch global signature to each 64-features point-signatures. The $256+64$ point descriptors are processed with an FC classification network to produce the semantic segmentation scores. It uses three shared FC layers of size 128, 128 and 64 channels and an output single shared FC unit. The sigmoid activation function limits the output scores in $[0, 1]$. Focal loss with $\alpha = 0.85$ is used to learn this task. The third stage of the model receives only the points that are segmented as vehicle and set the patch center as the centroid defined by these points. Then, the points are queried to a transformer PointNet that predicts the new center of the patch by means of an offset estimation. It uses three shared FC layers of size 64, 64 and 128 channels to generate point-descriptors which undergo a max-pooling operation to obtain a global patch signature. A subsequent FC classification network, with two FC layers of size 128 and 64 channels and two single FC unit, generates the offset predictions. The network predictions are used as final estimate without use of any activation function. Finally, a PointNet architecture uses shared FC layers of size 64, 64, 128 and 256 channels to compute point-signatures. Then, they are max-pooled to obtain a global patch signature which is processed with an FC classification network. In particular, it uses three FC layers of size 256, 128 and 64 channels to produce patch descriptors. Note that each patch results in an object prediction. Therefore, the features are used to produce box-parameters. A template-based formulation is adopted, using 16 templates, combining four length values – 2, 4, 6 and 8 meters – one width value – 2.5 meters – and four yaw values – $-\frac{3\pi}{8}$, $-\frac{\pi}{8}$, $\frac{\pi}{8}$ and $\frac{3\pi}{8}$. Consequently the network outputs 16 classification scores and the associated residual estimates. All the regression tasks are learned by means of the smooth L_1 loss, while the classification task adopts the cross-entropy objective function. The corner loss proposed in [79] is used to jointly optimize all the box-parameters, as suggested in the original paper. The first stage of the model does not require training. The second stage is trained in three steps: first the classification network for 8 epochs, then the semantic segmentation network for 6 epochs and, finally, both are fine tuned for 15 epochs. The same schedule used for the second stage of the proposed architecture is adopted to train the third stage: i.e. 5 epochs for each head and 15 epochs of fine tuning. The model proposed in [92] is referred to as **prior art**. In order to provide a comparison with the proposed architecture, it is evaluated in both 4frames and 1frame configurations. Finally, it has been decided not to use any grid-based approaches as benchmark, as Dreher et al. [109] proved that point-wise techniques are more effective on raw point clouds.

5.4 Results

This section contains the results produced by the tested methods on the object detection task. Unless otherwise specified, all the results reported are collected using the object-box proposals after NMS.

Table 5.2 contains a comparison between the performance of the proposed archi-

Table 5.2: Comparison between proposed approach and prior art on the moving vehicle detection task. In brackets are reported the statistics of the first stage network (first and second stage for the prior art).

Method	Config.	Moving Vehicle [%]			Params	FLOPS
		Prec.	Recall	F_1		
Prior Art	1frame	42.57	52.75	47.12	**354.2K** (159.8K)	2.21G (1.62G)
	4frames	49.02	**74.25**	59.05	**354.2K** (159.8K)	8.85G (6.46G)
Proposed	1frame	**89.06**	58.06	70.29	466.9K (205.3K)	**0.65G** (0.56G)
	4frames	86.85	70.26	**77.68**	466.9K (205.3K)	1.20G (1.03G)

tecture and the prior art [92]. Due to missing confidence information in the prior art model, it has not been possible to use PR curves in the analysis. Therefore, a single operating-point is selected on the PR curves of the proposed architecture and numerically compared with the prior art. In particular, the scores produced by box-predictions with confidence values above 0.5 are used. Moreover, only the results on the moving vehicle class are reported, as it represents the only class of objects detected by the prior art. An investigation about the performance on the pedestrian class is provided in section 5.4.2. Note how, the proposed architecture results in superior performance than the prior art. Under the same configuration, the proposed solution shows an improvement of at least 18% in F_1 score. In particular, in the 1frame setting, the proposed architecture proves to be significantly more effective than the prior art. Indeed, the F_1 score achieves a boost of 23% and both precision and recall scores result improved. The 4frames configuration also shows a neat performance improvement in the proposed architecture. However, here, the prior art achieves a competitive recall score, outperforming the proposed architecture. This effect is attributed to the redundant processing performed by the prior art, described in section 5.2. Indeed, it processes local patches centered around every input point, thus considering them as both center and object proposals. Although this mechanism enables the detection of more objects in the scene, however, it presents some drawbacks. One can be straightforwardly observed in the last two columns of table 5.2, reporting details about the computational complexity of the tested models. Note how, despite the prior art implementation has nearly 25% less parameters than the proposed architecture, it performs considerably more operations. Indeed, the 1frame configuration requires 3.4× the number of FLOPS of the proposed solution, while the 4frames configuration nearly 7.4×. Another severe shortcoming can be observed in the poor precision scores produced by the prior art models. Note, indeed, how the proposed architecture is at least 38% more precise. This is due to the large number of box-predictions generated, which, while favoring the object-recall, harms the model precision. This effect, however, can be attributed to two other reasons. Indeed, since the prior art does not output confidence scores, it does not have the ability to filter the box-proposals. Furthermore, the model has been designed to detect objects in specific conditions: it assumes a single instance to be present in each patch. Indeed, the semantic segmentation output is used to move the patch center closer to the true object center – by setting it as the centroid defined by the points classified as vehicle. However, this operation fails when two or more instances are present in the same patch, producing a patch-center which

lies in-between the objects. Finally, note how, in both methods, the 4frames input configuration leads to better performance than the 1frame counterpart. It produces improvements of at least 7% in terms of F_1 score, with a significant impact on the recall score. This is attributed to the richer information contained in the 4frames input, providing better description of the objects and the relative local context.

5.4.1 Box-Regression

The results in table 5.2 provides information about the ability of the tested models to detect objects. However, they do not include details about the fidelity of the box-predictions. In order to obtain an accurate perception of the scene, it is paramount that the box-proposals closely approximate the shape and appearance of the GT objects. By means of the IoU score, it is possible to analyze the accuracy of the box-predictions. This metric takes into account all the box-parameters – i.e. location, size and orientation –, as large scores can be obtained only when the two boxes under comparison share similar parameter-configurations. Figure 5.11 plots the average IoU score collected on various configurations of the confidence threshold. The scores are obtained by averaging the IoU generated by correct box-predictions across the whole test-set. The exact implementation is used to compute the IoU. In addition, by reporting the score produced on different confidence thresholds, it is possible to observe how various precision/recall operating-points affect the box-accuracy. As expected, the prior art shows flat lines, due to missing confidence information. The left-most section of the plot shows the configurations with high

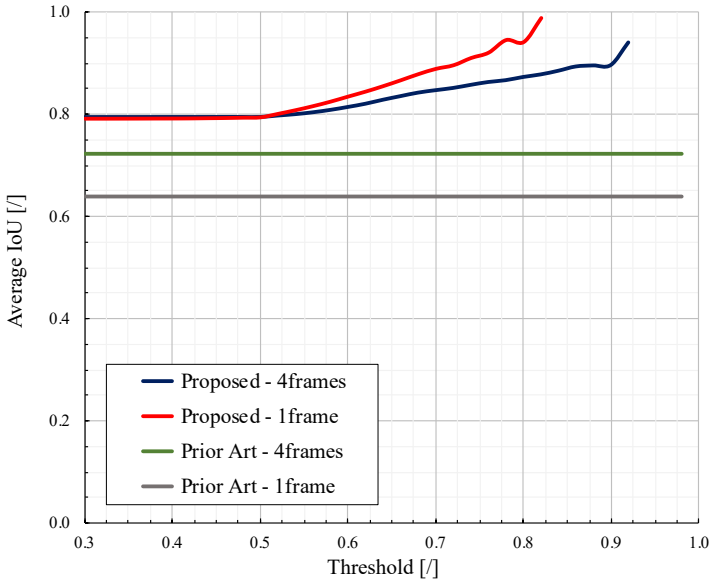


Figure 5.11: Average IoU score of proposed solution and prior art on different values of confidence threshold. Different network configurations are shown with different colors. Best viewed in color.

recall – the lower the threshold, the lesser boxes rejected, the higher the probability to detect all objects. Since the object proposals after NMS have been used to collect the results, the lowest threshold value is γ_{NMS}^{conf} – i.e. 0.3. Note how, here, the proposed architecture shows IoU scores of nearly 0.8 in both 1frame and 4frames settings. This means that the box-predictions have, in average, an intersection area with the corresponding GT which amounts for 80% circa of the total area. The prior art, instead, shows poorer scores, producing an average IoU of almost 0.73 in the 4frames setting and only 0.64 when using a single radar frame. This suggests that the proposed regression-aided formulation generates more accurate box-parameters than the template-based one. Note that, when the threshold assumes a value of 0.5 – i.e. the one used in table 5.2 –, similar considerations can be repeated, as the lines of the proposed architecture are almost flat until this value. Other interesting insights can be collected by analyzing the behavior of the curves produced by the proposed architecture. Indeed, it is possible to note that, as the threshold increases, the IoU score grows. In particular, the right-most configurations, which result in the highest precision values, show scores very close to 1 – 0.94 in 4frames setting and 0.98 in 1frame. Therefore, in this configuration, the network predicts boxes which are very accurate, almost exactly overlapping with the real object-boxes. This proves that the confidence values produced by the network are meaningful, discriminating accurate boxes from poor estimates. Consequently, it validates the proposed solution for the definition of confidence GT labels. Finally, it is worth noting how, in the 1frame setting, the proposed architecture achieves lower confidence values than in the 4frames setting. Indeed, the red line ends at the threshold value of 0.82 while the blue one at 0.92 – because the respective configuration does not predict boxes with higher confidence values. This can be attributed to the poorer information contained in a single-frame radar point cloud, preventing the network from producing high-confident box-predictions.

Another aspect of the networks worth analyzing is the yaw regression head. Despite it is taken into consideration in the IoU computation, it is interesting to investigate how accurately the different methods perform this task. Figure 5.12 plots the yaw RMSE collected on various values of the confidence threshold. Similarly to fig. 5.11, all the correct box-predictions in the test-set are used. However, eq. (5.18) is used to compute the scores. As explained in section 5.3, it has been preferred to use the RMSE measure over MSE because large orientation errors have a significantly higher impact on the final object appearance than small ones. Note how the proposed architecture results in lower error scores than the prior art. At high recall values – i.e. lower threshold –, the proposed architecture produces RMSE scores close to 15° in both 1frame and 4frames settings, while the prior art shows RMSE scores around 20°. This proves that the proposed spherical-based implementation – with dynamic weighting – generates more accurate yaw-estimates than the template-based regression. Furthermore, it is interesting to note how the error score produced by the proposed architecture decreases when the threshold increases – i.e. higher precision. This effect confirms the observations drawn from fig. 5.11 about the network confidence output. In addition, it suggests that, in order to generate high confidence predictions, it is not enough to have accurate location and size predictions, but the network also requires a precise yaw estimate. Finally, note how the most confident boxes estimated by the proposed architecture present very accurate yaw predictions, achieving an RMSE score lower than 5°.

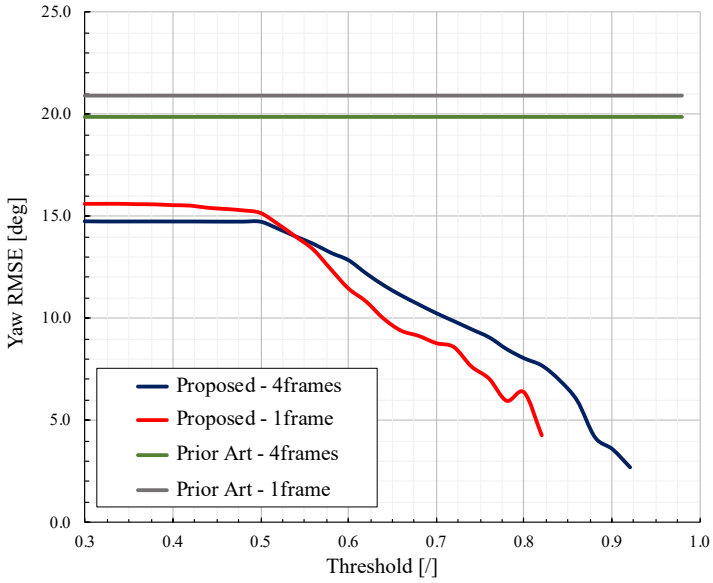


Figure 5.12: Yaw RMSE of proposed solution and prior art on different values of confidence threshold. Different network configurations are shown with different colors. Best viewed in color.

5.4.2 Multi-Class Task

The results reported so far have proved that the proposed architecture optimizes the prior art. Indeed, it shows superior detection performance while using less computational complexity. In addition, it outperforms the method in [92] in the box-regression task, showing higher IoU scores and lower yaw RMSE. In this section, the results of the proposed architecture on the multi-class problem are reported. Figure 5.13 contains the PR curves produced on both moving vehicle and pedestrian classes. As reference, the performance of the prior art models on moving vehicles are reported – cross and diamond markers. It is possible to note how the results produced by the network on the vehicle class are significantly better than on the pedestrian class. This is attributed to the smaller sizes of pedestrian objects, forcing them to result underrepresented in the input. Bear in mind, indeed, that the network uses radar point-reflections as input. Therefore, if an object is not represented by any point, there is no data to enable its predictions. Similar considerations can be repeated for instances described by a single point. Though, technically, it would be possible to detect these objects, the task of inferring the box-parameters is complicated by the limited information present in the input. These observations are supported by the different performance produced in the 4frames and 1frame settings. As concluded from table 5.2, the network achieves superior performance in the 4frames setting. However, it is possible to note how the pedestrian class exhibits a huge performance improvement, resulting in almost 3× the recall score of the 1frame setting – while it increases by only 12% circa for vehicles. This suggests that it is possible to detect more pedestrians by providing stability to the object-

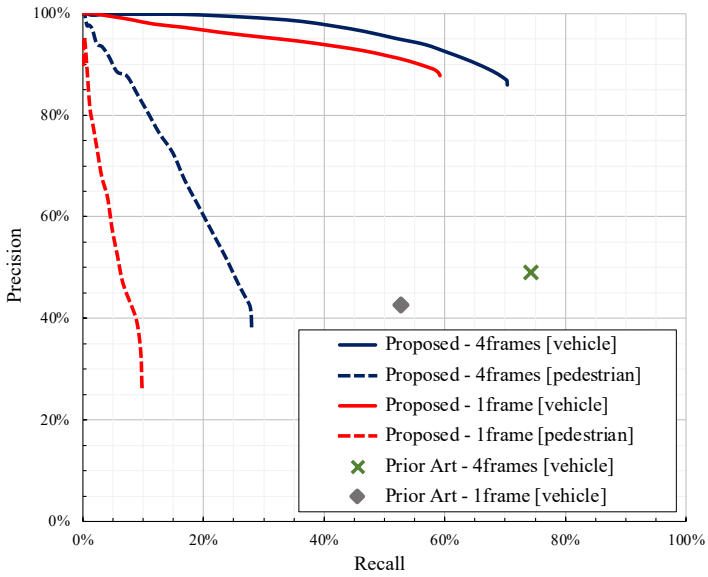


Figure 5.13: PR curves produced by the proposed archicture. Different configurations are reported in different colors. The solid lines show the performance on moving vehicles, while the dashed lines refer to the pedestrian class. The performance of the prior art models on moving vehicles are reported for reference – cross and diamond markers. Best viewed in color.

representation in the input. Indeed, the 4frames inputs contain point-reflections measured in the current frame plus the last 3 frames. Consequently, a pedestrian instance which does not provide any measurement in the current frame, is still represented by data from previous frames. Therefore, this setting ensures more input-points per object, thus providing the network with the information necessary for their detection.

Another interesting study consists in measuring, similarly to section 5.4.1, how accurately the network can estimate the box-parameters in a multi-class problem. Since, for pedestrian objects, the network does not regress the orientation, it has been decided to skip the yaw-error analysis – results on the vehicle class are reported in fig. 5.12. Figure 5.14 plots the average IoU scores collected on different configurations of the confidence threshold. Bear in mind that pedestrian-boxes are predicted with fixed size, therefore, the object location represents the only parameter evaluated. Note how, despite the network achieves worse detection scores on pedestrian objects, it shows competitive performance in terms of box-accuracy. Indeed, at high recall configurations, the box-predictions result in circa 0.7 average IoU in both 4frames and 1frame settings. This proves the ability of the network to correctly estimate the location of pedestrians in the scene. Note how, once more, the small dimensions of pedestrians strongly affect this metric, as even small location errors produce large drops in IoU. By analyzing the behavior of the curves, it is possible to obtain similar observations than in fig. 5.11. Indeed, for both classes, the

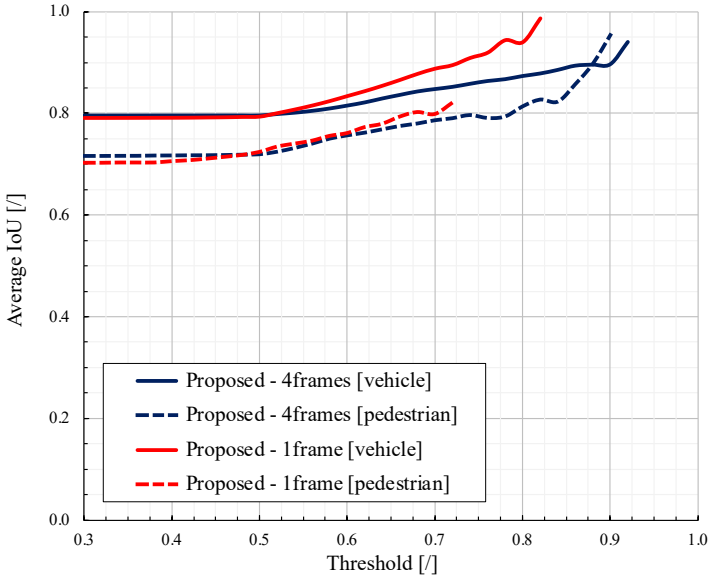


Figure 5.14: Average IoU score of the proposed architecture on different values of confidence threshold. Different network configurations are shown with different colors. The solid lines show the score on moving vehicles, while the dashed lines refer to the pedestrian class. Best viewed in color.

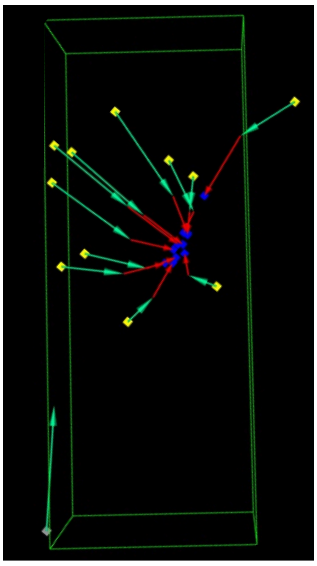
IoU score produced by the network predictions grows together with the confidence threshold. Note, in more details, how pedestrian box-predictions achieve scores as high as 0.95, in the 4frames setting, and 0.82, in the 1frame setting. This suggests that the distance-based process, designed for definition of pedestrian GT confidence labels, is effective and provides the network with meaningful data for learning this task. Furthermore, it is worth noting the different range of confidence values generated by the network. Similarly to vehicle objects, pedestrians result in higher confidence values on the 4frames setting. However, though the highest score in this setting is comparable with the one for the vehicle class – i.e. 0.9 versus 0.92 –, it is decisively low in 1frame setting, as the network does not produce any box with confidence higher than 0.72. This is attributed to the poorer information-content provided by the input, thus supporting the considerations obtained from fig. 5.13. Indeed, the network cannot produce high-confident predictions because it does not have enough data to accurately infer the presence of pedestrian instances. Finally, fig. 5.14 confirms the obstacles faced by the network when detecting pedestrians in radar point clouds from a single frame. Yet, it proves that the network operates well in 4frames setting, matching reasonable detection scores with a good estimation accuracy.

5.4.3 Visual Results

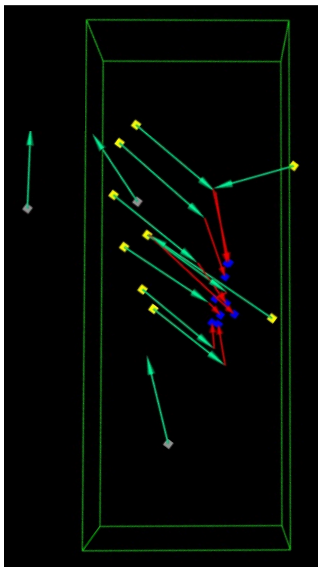
Visual analyses represent an important tool for assessing the correctness of the network operations. In this section, visualizations of the proposed architecture outputs are provided. It has been decided to adopt the model trained in the 4frames setting, as it showed the best performance.

Figure 5.15 contains visual examples about the offset and residual offset predictions of the network in different scenarios. Though the objects used in this analysis have various GT yaw configurations, they have been rotated and aligned with the x-axis (vertical) for visualization purposes. Figure 5.15A shows a common example, where the object has a moderate velocity and the offset predictions from the first stage have small errors. Note how the offset vectors (cyan arrows) already concentrate the center proposal points (yellow) close to the center of the object. Then, the residual offset vectors (red arrows) further improve the object box-center proposal by fine-tuning the location estimate. Note how the final object-centers (blue points) result very close to each other. In fig. 5.15B is depicted an example where the offset predictions produce large errors. Note how, after being shifted by the offset vectors, the center proposals are still distributed along the long box-dimension. Indeed, the offset predictions correctly estimate the shift in the horizontal axis, but have some residual error in the vertical axis. In this case, the residual offset vectors adjust the final prediction, mostly compensating the error along the vertical axis – the main component of most of the red arrows lies along this axis. Figure 5.15C shows an example of fast moving vehicles. Since reflections from previous frames are aggregated in the same input, some of the points produced by the object are found in the rear of the box – it is not possible to compensate for its movement as its velocity is unknown. Note how the network classifies these points as center-proposal, though they are associated with the background class. This is because such points have the radar properties specific of moving vehicles. However, these proposed centers are far away from the true center of the object and, without any offset regression, they will produce BBs with very poor IoU scores – even assuming perfect size and orientation estimations. Note how the offset vectors move the points inside the object box, but the residual offset vectors are necessary to bring all the points together and close to the object center. Furthermore, note how the decision to mask points in the vicinity of the GT boxes helps the network not to get confused in such circumstances – it is not penalized for predicting these points as center-proposal. Another interesting scenario is proposed in fig. 5.15D. It shows how the network operates on slow-moving vehicles – i.e. whose velocity is in the interval $[1, 2.5]$ m/s. Bear in mind that, in order to avoid confusing the network, these instances are masked out during training – as described in section 3.3.1. Yet, the network can produce center-proposals for these objects. Similarly to fig. 5.15A, the residual offset vectors refine the object box-centers, correcting residual errors of the offset vectors. In addition, fig. 5.15D also shows how not all the center-proposal points result in an object box: indeed, only 3 blue points (and 3 red arrows) are present out of the dozens selected points. Finally, it is worth noting how, in all the examples of fig. 5.15, points not classified as vehicle center-proposal (gray), experience offset vectors with directions significantly different than center point-proposals. This suggests that the offset head learns to distinguish between object and background points, pushing these latter away from the object-center.

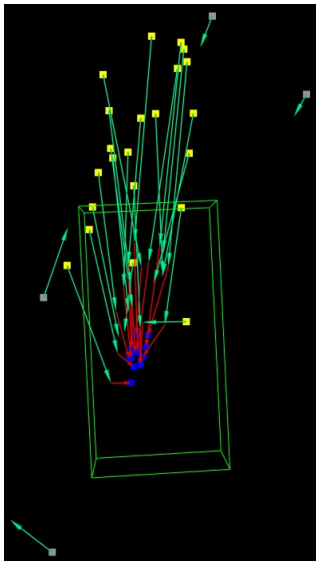
Figure 5.16 contains an example of box-predictions generated by the proposed ar-



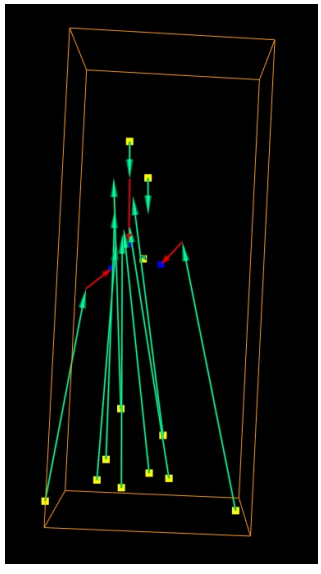
(A) Small errors.



(B) Large errors.



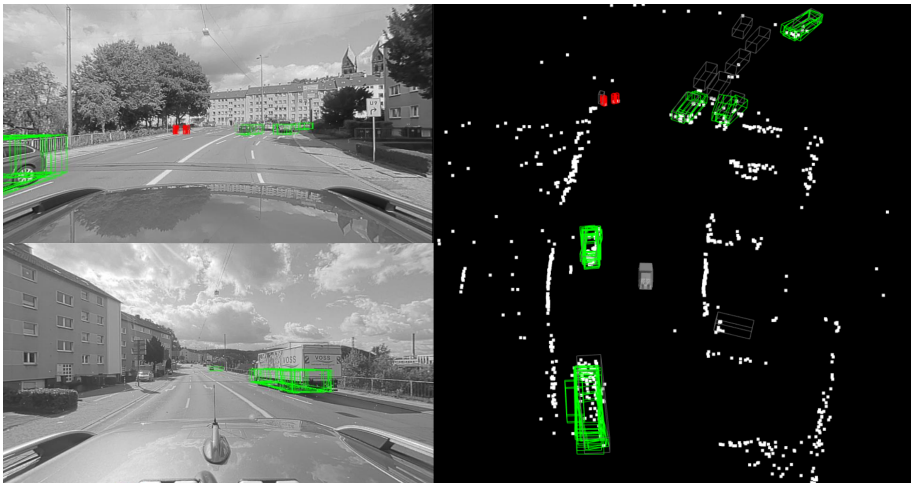
(C) Fast-moving.



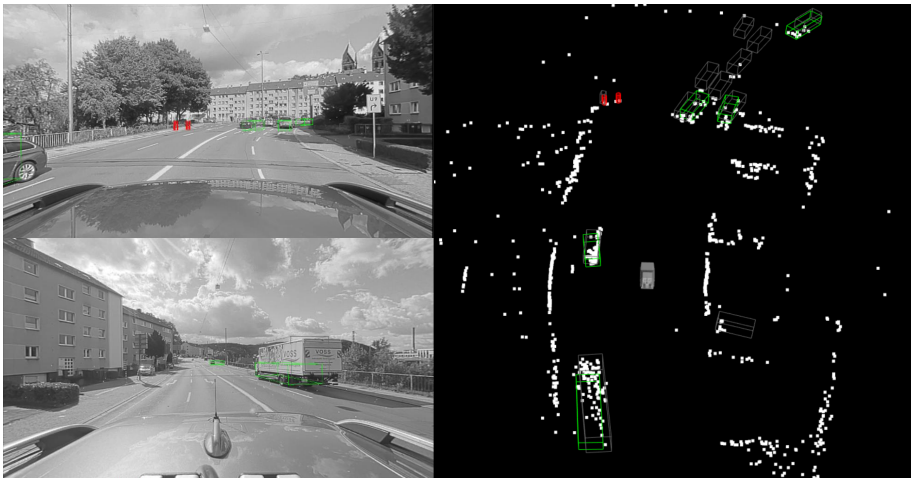
(D) Slow-moving.

Figure 5.15: Visual examples showing the location estimation task performed by the proposed architecture. Cyan arrows represent offset vectors predicted by the first stage. Red arrows show the residual offset vectors estimated by the second stage. Yellow points are center proposals, while gray points are classified as background. The blue color marks the center of object-predictions. Boxes represent GT vehicles: moving (green) and slow-moving (orange). Best viewed in color.

chitecture. In order to gather more insights, for the same scene, the box-predictions before and after application of the NMS algorithm described in section 5.3.2 are reported. To improve visual perception, the estimated boxes are extended along the z-dimension – using a fixed height parameter of $2m$ – and aligned with the road plane – i.e. null elevation. Note how the urban scene depicted in fig. 5.16 is quite miscellaneous. Indeed, it contains moving and stationary vehicles of various sizes, pedestrians, trees and buildings. Figure 5.16A shows the raw network predictions. Here, only boxes with confidence above 0.3 – i.e. the value of γ_{NMS}^{conf} – are displayed, as boxes with confidence below this value will be discarded by the NMS algorithm. Note how the network can recognize most of the objects of interest. Indeed, the vehicles in the incoming lane are detected as well as the pedestrian and some of the vehicles in front of the ego-car. Note that the cluster of vehicles in front are standing behind of a red traffic light. Since the network is trained to detect only moving vehicles, they are not false negatives – i.e. erroneously missed objects. Nevertheless, it is interesting to note how some of these vehicle-objects do not provide any radar-reflection – e.g. the left-most instance at the top of the scene. Therefore, the network would not have any data to support its detection. Figure 5.16B reports the same network predictions filtered out by the NMS algorithm. Note how the scene results much more clean and accurate, as multiple predictions for the same physical instance are reduced to a single one. It is interesting to note how the large vehicle at the bottom of the scene obtains a box-prediction which is larger than the other common-sized vehicles. Though among the box-predictions there are smaller boxes, the network gives a larger confidence score to the large box. This proves that the network does not overfit to the most-represented dimensions but regresses the proper object-size. Furthermore, note how the network is able to detect vehicles oriented in various direction and does not overfit to the most represented angles – i.e. in the interval $[-5°, 5°]$. Finally, it is worth noting how, despite detecting the pedestrian in the scene, the network produces a false positive in its vicinity. Note also how the NMS algorithm cannot filter out this box. Yet, false positives like this one, which occur for a single frame, can be easily filtered out by means of a tracking algorithm. Additional visualizations of the network predictions in different scenarios are provided in appendix A.4.



(A) Network box-predictions before NMS.



(B) Network box-predictions after NMS.

Figure 5.16: Visual example of the box-predictions produced by the proposed architecture. The same scene is proposed before and after NMS. Green boxes show moving vehicle predictions, while red cylinders represent pedestrians. GT boxes are reported in gray to avoid visual confusion. Gray points show the input point cloud fed to the network. Left. Front/rear camera view. Right. 3D visualization. Best viewed in color.

Chapter 6

A Novel Statistical Detector to Defend against Weaknesses of Deep Learning

The previous chapters prove that DL represents a useful tool for effective processing of radar data. However, in order to enable adoption of such technology in safety-critical applications, like autonomous vehicles, their risks and limitations must be established and tackled.

In this chapter, a specific weakness of DL is addressed: adversarial examples, i.e. maliciously manipulated inputs created to force the system to a customized output. After a brief introduction on the attack modalities of such techniques, a novel solution developed for defending against this weakness is presented and validated. As adversarial examples are a well-established threat in the computer vision domain, camera data are used to perform the experiments. Yet, in order to assess the impact of the proposed solutions in autonomous vehicle applications, a traffic-sign classification task is used in the evaluation procedure.

This chapter contains the following own contributions:

- Development of an innovative, cost-effective, attack-agnostic and robust adversarial detector using statistical information to identify malicious samples.
- Assessment of various properties of the detector, such as compatibility with other defensive solutions, usage of distortions and performance under various attack settings.
- Introduction of a new, effective similarity metric for the detection task, exploiting the orthogonality between malicious and legitimate samples.
- Development of a brand new distortion, exposing adversarial examples to detection.
- Introduction of solutions improving the state-of-the-art, such as usage of a signature extraction process and first-order statistics as class representative vectors.

The method presented in this chapter obtained a patent in [91]. Both, method and results were previously published in [106].

6.1 Adversarial Examples

Recent studies have shown that it is possible to force DL methods to produce a custom outcome by imperceptibly altering the input [41, 37, 44, 59]. These perturbed inputs, a.k.a. **adversarial examples**, severely undermine the stability of systems based on DL solutions. For instance, consider an autonomous vehicle that relies on DL to detect and recognize traffic signs. By tampering the sign, an attacker might force the system to misclassify a 30 km/h speed limit sign as a 130 km/h, thus causing the vehicle to take potentially dangerous actions.

Adversarial examples are inputs specifically altered to fool a DL model, while appearing authentic to humans. Generally, they are built by adding to the input a noise signal, a.k.a. **adversarial perturbation**, that results from an optimization process. Figure 6.1 shows an instance of adversarial example. Both, legitimate and perturbed images contain a penguin. However, despite correctly recognizing the legitimate image (leftmost), the DL model is misled by the malicious sample (rightmost) into a wrong prediction.

Formally, given a legitimate input sample $\boldsymbol{x}$, it is possible to generate an adversarial example $\boldsymbol{x'}$ as

$$\boldsymbol{x'} = \boldsymbol{x} + \boldsymbol{\eta} \ , \tag{6.1}$$

where $\boldsymbol{\eta}$ represents the adversarial perturbation. This latter is computed as a solution of the constrained optimization problem in eq. (6.2), where $f(\cdot)$ represents the function approximated by the target model and $\|\cdot\|$ the norm operator

$$\boldsymbol{\eta}^* = \arg\min_{\boldsymbol{\eta}} \|\boldsymbol{x'} - \boldsymbol{x}\| \tag{6.2a}$$

$$s.t.\ f(\boldsymbol{x}) \neq f(\boldsymbol{x'}) \tag{6.2b}$$

$$\boldsymbol{x'} \in [0,1]^N \ . \tag{6.2c}$$

Equation (6.2a) defines the optimization problem, whose objective function consists in the magnitude of $\boldsymbol{\eta}$. Minimizing this quantity is tantamount to pushing $\boldsymbol{x'}$ to be as similar as possible to the legitimate sample $\boldsymbol{x}$. However, the optimization problem in eq. (6.2a) finds a trivial solution in the null perturbation – yielding the lowest objective score of 0. By addressing the constraint in eq. (6.2b), it is possible to avoid the trivial solution, forcing a perturbation that changes the original output

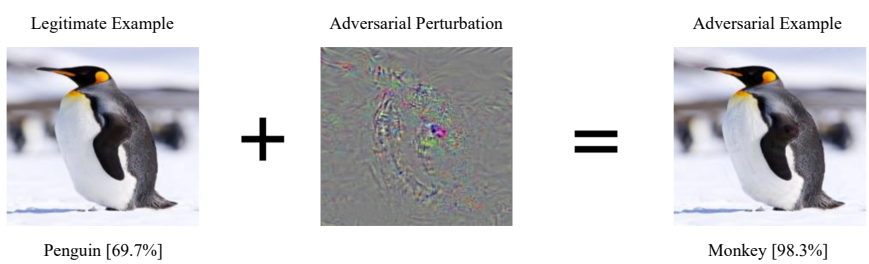


Figure 6.1: Instance of an adversarial example. The legitimate image is correctly classified as penguin. By altering it with an adversarial perturbation, the system is misled to predict the monkey class with high confidence. Yet, both images look natural and identical to a human. Best viewed in color.

of the system. Finally, the constraint in eq. (6.2c), a.k.a. **box-constraint**, limits the search only among valid inputs.

Besides representing a potential threat, adversarial examples unveil unknown characteristics of NNs and their intrinsic operations. Indeed, finding out that it is possible to produce profound variations with slight input alterations, exposes properties of such technology that were concealed before. Therefore, fueled by the opportunity to advance knowledge, scientist have started to develop novel attack and defense strategies. The next sections serve as a background overview, describing the most relevant state-of-the-art in this area.

6.1.1 Attack Methods

Adversarial attacks can be split into two main categories, based on the attack-settings adopted. **White-box** attacks assume full access to the target model. The attacker can query the victim with custom inputs and leverage the back-propagation signal used during training. However, it cannot alter the target network. On the other hand, in **black-box** scenarios, the attacker does not have any access to the model: it has only knowledge about the task the victim is deployed for. Generally, black-box attacks rely on the transferability property of adversarial inputs, according to which, a maliciously-modified sample can be effective against various models, beyond the victim one. Therefore, a common strategy consists in training a substitute model – on the same task solved by the targeted model – and then attack it assuming white-box settings. Evidently, such attacks usually result less effective than white-box ones.

Another factor that strongly characterizes adversarial examples is the process used to generate them. Indeed, since they are the outcome of an optimization problem, it is possible to instantiate various attack procedures with different properties.

The Fast Gradient Sign Method (FGSM) is a simple yet effective single-step attack [41]. It generates malicious samples by altering the input features in the direction of the gradient of the loss function w.r.t. the input. Formally, given the input-label pair $\boldsymbol{x}$-l and the corresponding loss produced by the network $L(\boldsymbol{x}, l)$, the FGSM attack computes the adversarial perturbation as

$$\boldsymbol{\eta} = \epsilon \cdot sign(\nabla_{\boldsymbol{x}} L(\boldsymbol{x}, l)) \ , \tag{6.3}$$

where $\epsilon > 0$ controls the magnitude of the perturbation. FGSM is a very fast and cheap method that enables the generation of malicious samples in real-time. On the other hand, since every feature is altered by the same magnitude, it produces bold perturbations, sometimes even visible to human eyes. Nevertheless, the ability of such a greedy technique to undermine the stability of NNs represents a major finding, capturing the interest of the research community. Indeed, although FGSM is one of the first adversarial attack ever developed, it has inspired several researchers, leading to various attack derivations [59, 52, 84, 71, 60].

Other researchers focused on the generation of malicious samples with the smallest adversarial perturbation. Moosavi-Dezfooli et al. [50] proposed to approach the problem from a geometrical perspective. They noticed that a NN performs its task by separating the input domain into decision regions – e.g. for a classification task, each region is associated to a class. Therefore, it is possible to generate an adversarial example by moving the original input into a different decision region. As

a result, the authors devised DeepFool (DF), an iterative algorithm that generates adversarial examples with the smallest perturbation possible. For this purpose, DF first linearizes the boundaries between decision regions as hyper-planes. Then, at each iteration, it computes the perturbation that moves the current input towards the closest hyper-plane. Since the process stops when the perturbed input achieves misclassification, its outcome consists in an adversarial example which lives *just beyond* the closest decision boundary. DF produces perturbations that are rarely detectable by humans. However, this attack does not provide full-control over its effectiveness: occasionally, it produces adversarial examples that are on the decision boundary edge, thus resulting in ambiguous predictions. Despite being an iterative procedure, it is very computationally efficient, as it takes only few steps to converge[1].

Carlini and Wagner (C&W) [54] is a technique devised to give the attacker full control over the adversarial generation process. The authors analyzed the optimization problem in eq. (6.2) from a mathematical point-of-view. They noticed that the constraint in eq. (6.2b) is highly non-linear. Therefore, they substituted it with a cost function $g(\cdot)$ such that $g(\boldsymbol{x'}) \leq 0$ if and only if the malicious input is misclassified. In addition, the authors devised a reparametrization trick to approach the box-constraint in eq. (6.2c). Specifically, they defined the adversarial perturbation $\boldsymbol{\eta}$ as in eq. (6.4) and optimized over the latent variable $\boldsymbol{v}$, thus ensuring the validity of the solution

$$\boldsymbol{\eta} = \boldsymbol{x'} - \boldsymbol{x} = \frac{1}{2}(\tanh(\boldsymbol{v}) + 1) - \boldsymbol{x} \ . \tag{6.4}$$

The optimization problem used to generate C&W adversarial examples is shown in eq. (6.5)

$$\boldsymbol{v}^* = \arg\min_{\boldsymbol{v}} \ \|\boldsymbol{x'} - \boldsymbol{x}\| + c \cdot g(\boldsymbol{x'}) \ , \tag{6.5}$$

where $\boldsymbol{x'}$ is obtained from eq. (6.4) and $c > 0$ is a constant. Notice how the parameter c and the function $g(\cdot)$ enable the attacker to tune the optimization process. C&W represents the state-of-the-art attack technique and it has been a benchmark for the development of new defensive and attack methods [86, 63, 55, 90].

6.1.2 Defense Strategies

Defensive techniques against adversarial examples include all those methods that aim at reducing the success-rate of adversarial attacks. They can be categorized into two main families: **proactive** and **reactive** [102]. Proactive methods tamper the design or training phase to build robustness into the defended network. On the other hand, reactive methods aim at defending the victim model by detecting malicious samples or reconstructing the original, legitimate input. They are generally deployed after the network has been trained.

Goodfellow et al. [41] assumed that, given the chance, NNs can learn to recognize adversarial perturbation patterns. Therefore, the authors proposed to tamper the training procedure by including manipulated inputs. In particular, at every training step, they extended the mini-batch with malicious samples obtained from the legitimate counterpart. In this way, it is possible to store in the network knowledge about adversarial patterns, thus increasing its robustness. Adversarial training has

[1]NNs produce non-linear decision boundaries that are approximated by linear hyper-planes. The iterations are needed only to ensure crossing over the true non-linear decision boundary.

proven to be an effective solution, considerably decreasing the success-rate of state-of-the-art attack methods. However, as it requires live-computation of adversarial perturbations, it cannot use lengthy, iterative attack methods. Furthermore, it is not attack-agnostic: it is not effective against techniques that produces adversarial perturbations with different patterns than the ones used during training. Finally, adversarial training has shown an interesting side-effect: it increases the model generalization to legitimate data, thus improving its performance [41, 43]. The authors of [41] attributed this effect to the strong similarity between adversarial sampling and hard example mining. Indeed, adversarial examples can be considered as noisy inputs that resist classification. Since the network is trained with noisy, hard-to-classify inputs, it learns better representations of the ground truth.

Other researchers realized that a valid defensive strategy consists in detecting malicious samples. Grosse et al. [58] suggested a variant of adversarial training. They created a manipulated version of the legitimate dataset and trained a binary classifier to predict the nature of the input (legitimate or malicious). In this way, it is possible to perform a screening of the inputs and avoid to use unwanted outputs produced by adversarial examples. Unfortunately, despite proving very efficient against black-box attacks, this method is easily by-passed in white-box attack-settings [53]. In another work, Xu et al. [86] noticed that adversarial attacks exploit the high dimensionality of the input space to find unnatural representations around the legitimate sample. Therefore, they proposed to lower the attacker opportunities by limiting the degree-of-freedom of the input domain. Indeed, the authors observed that, when distorted with simple computer-vision methods – such as spatial smoothing or color bit-depth reduction –, adversarial examples are rarely effective[2]. On the other hand, NNs are known to be very robust to such distortions. As a result, they designed Feature Squeezing (FS), a reactive defensive strategy using distortions to detect malicious samples. Every time the victim network receives a new input, FS applies a distortion and feeds both original and distorted samples to the network. Then, it compares the two outputs by computing the L_1 distance. In case the input has been manipulated to fool the network, the distance between the two outputs will be consistent[3]. Therefore, by thresholding this score, it is possible to determine whether the input is legitimate or malicious. FS has proven effective against both black- and white-box attack-settings. Moreover, it is an attack-agnostic technique and does not impose any limitation during the design/training phase. However, it results in higher computational demands, as it requires to query the victim model multiple times.

Another defense possibility consists in cleaning up the adversarial perturbation to reconstruct the legitimate input. Liao et al. [75] developed High-level representation Guided Denoiser (HGD), a NN module trained to purify the input of any malicious pattern. HGD works as a pre-processing module: it receives the input and produces as output the sample to be fed to the network. It is trained to minimize the input representation at the high-level hidden-layer of the victim network. The authors proved that HGD performs very well against various attack methods and can be successfully transferred to other architectures. Moreover, they showed that this module works in two steps: it removes part of the adversarial perturbation and then applies a pattern that facilitates the classification task. Unfortunately, it is exposed

[2]They live in a very narrow space around the legitimate input.

[3]Assuming that the attack is not robust against the distortion.

to white-box attacks, as it forms, together with the defended network, a new system vulnerable to most attack methods. Finally, it requires adversarial inputs during training, therefore, it is not attack-agnostic.

6.2 A Statistical Scheme to Detect Adversarial Examples

Adversarial detectors represent the most practical solution among the various defensive approaches described in section 6.1.2. These methods defend the victim network by identifying whether an input is manipulated or not. In this way, it is possible to discard the network output forced by a malicious sample and void its effects.

Adversarial detectors are usually distinct modules, detached from the victim network. This enables the system to operate in different modalities, without affecting the defended model. For instance, it is possible to activate the detector only when certain conditions applies. Moreover, since they are independent from the victim network, they do not impose any design nor training constraints. Finally, adversarial detectors are generally attack-agnostic and can be easily deployed along with other defensive strategy, thus combining the advantages of several techniques.

The system developed herein shares the theory from [86] that adversarial examples live in a narrow region around the natural input. One can think of them as outliers: singular data-points that produce abnormal outputs compared to their close neighbors. Statistics provide the means to distinguish between natural and ambiguous data-points. Indeed, by exploring the vicinity of an input, it is possible to build up a signature that characterizes it. One way to explore the input domain consists in using greedy computer-vision distortions. For instance, blurring an input image allows one to move the input in an adjacent region of the space while maintaining its original content. As an additional side effect, these distortions squeeze out the input domain, thus drastically reducing the regions where adversarial examples proliferate.

In the next sections, it is presented a novel solution for defending against adversarial examples: a statistical adversarial detector that leverages dataset statistics and input-level distortions to build an efficient detection scheme [106]. In the rest of the chapter, the term *perturbed input* is used to refer to an adversarially perturbed input, while *distorted input* refers to the input after application of a distortion. Moreover, it is used the term *original* to refer to the input before the application of the distortion, regardless from its nature – i.e. legitimate or malicious.

6.2.1 Detection Concept

Similarly to [86], the proposed detector exploits computer-vision distortions to detect adversarial examples. It has been observed that, usually, legitimate and manipulated samples exhibit different statistics when distorted. Consider, without loss of generality, a natural sample $\boldsymbol{x_1}$ and a manipulated sample $\boldsymbol{x'_2} = \boldsymbol{x_2} + \boldsymbol{\eta_2}$ that, when fed to the classification network with equivalent function $f(\cdot) \triangleq f(\cdot; \boldsymbol{\theta})$, they result in the same output class prediction, namely

$$c_1 = f(\boldsymbol{x_1}) = f(\boldsymbol{x'_2}) \neq f(\boldsymbol{x_2}) = c_2 \ . \tag{6.6}$$

It has been noticed that, given a suitable distortion $\psi : [0,1]^N \to [0,1]^N$, eq. (6.7) holds true: i.e. the adversarial sample is no longer effective, while the legitimate sample is still correctly classified

$$c_1 = f(\psi(\boldsymbol{x_1})) = f(\boldsymbol{x_1}) \ , \tag{6.7a}$$
$$c_2 = f(\psi(\boldsymbol{x'_2})) \neq f(\boldsymbol{x'_2}) \ . \tag{6.7b}$$

Therefore, it is possible to detect adversarial examples by simply monitoring the model prediction on the original and distorted replica of the input[4]. Indeed, by combining eqs. (6.6) and (6.7), it is possible to write

$$f(\psi(\boldsymbol{x_1})) - f(\boldsymbol{x_1}) \neq f(\psi(\boldsymbol{x'_2})) - f(\boldsymbol{x'_2}) \ , \tag{6.8a}$$
$$0 \neq c_2 - c_1 \ . \tag{6.8b}$$

However, eq. (6.7b) does not always hold true and using only the criteria in eq. (6.8) for detection would be really inefficient. In the proposed method, it has been decided to leverage this property in a different way: the network's prediction vectors are compared to each other, instead of the final class predictions. Formally, given the classification network $f(\cdot)$ of a d-classes task, it is called prediction vector $f_{-1}(\boldsymbol{x}) \in [0,1]^d$ the network layer that contains, in each entry, the probability associated with every one of the d classes. It is related to the output class prediction as

$$f(\boldsymbol{x}) = \arg\max_i f_{-1}(\boldsymbol{x}) \quad \text{with } i \in [1\,..\,d] \ . \tag{6.9}$$

Since the distortion generally changes the class prediction of a malicious sample, it is safe to assume that it generates diverging prediction vectors – the highest entry will be at a different position. On the other hand, legitimate samples will maintain the same class prediction, i.e. the position of the highest value will remain unchanged. Consequently, by using eq. (6.9), it is possible to rewrite eq. (6.7) as

$$f_{-1}(\psi(\boldsymbol{x_1})) \approx f_{-1}(\boldsymbol{x_1}) \ , \tag{6.10a}$$
$$f_{-1}(\psi(\boldsymbol{x'_2})) \not\approx f_{-1}(\boldsymbol{x'_2}) \ , \tag{6.10b}$$

where the similar sign substitutes the equal sign to cope with vector-valued quantities[5]. Therefore, there exists a similarity function $S(\cdot,\cdot) : [0,1]^s \times [0,1]^s \to \mathbb{R}$ such that

$$S\big(f_{-1}(\psi(\boldsymbol{x_1})), f_{-1}(\boldsymbol{x_1})\big) > S\big(f_{-1}(\psi(\boldsymbol{x'_2})), f_{-1}(\boldsymbol{x'_2})\big) \ , \tag{6.11}$$

where s accounts for the dimension of the arguments – in eq. (6.11) $s = d$. Indeed, provided that $S(\cdot,\cdot)$ returns a score proportional to the similarity between two vectors, legitimate samples result in high scores, while manipulated ones produce poor similarity rates. Hence, it is possible to use a threshold to distinguish between legitimate and malicious samples. Equation (6.12) contains the detection criteria resulting from eq. (6.11), where $\boldsymbol{x}$ is a generic input, α a suitable threshold, H_0 the hypothesis of legitimate sample and H_1 the hypothesis of malicious sample

$$S\big(f_{-1}(\psi(\boldsymbol{x})), f_{-1}(\boldsymbol{x})\big) \underset{H_1}{\overset{H_0}{\gtrless}} \alpha \ . \tag{6.12}$$

[4]For proper selection of the distortion. More details about distortions in section 6.2.4.

[5]Here, two vectors are considered similar when they have similar element-wise values.

Equation (6.12) suggests to detect adversarial examples by comparing the prediction vectors of original and distorted version of the input. Compared with eq. (6.8), it proposes a more robust detection criteria, as it remains sensible to smaller variations which would not result in a different class prediction.

Despite eq. (6.12) provides an effective detection criteria, it still has some drawbacks. It has been noticed that, since eq. (6.12) has been derived from considerations on the class predictions, it holds particularly true for high-confidence adversarial examples. These samples produce a well-defined prediction vector $f_{-1}(\boldsymbol{x'})$, with a spike at the location of the predicted class and all the other entries close to 0. When distorted, as a result of a variation of the class prediction, they change the position of the highest entry in $f_{-1}(\psi(\boldsymbol{x'}))$, which leads to poor similarity scores. Conversely, low-confidence malicious samples produce a lower peak in $f_{-1}(\boldsymbol{x'})$. After being distorted, they would still result in a different class prediction, but the element-wise difference will be lower. Because of this, they generate higher similarity scores than high-confidence samples, thus increasing the risk of evading detection.

For this reason, the proposed detector substitutes the prediction vectors on the original samples – i.e. $f_{-1}(\boldsymbol{x_1})$ and $f_{-1}(\boldsymbol{x'_2})$ – with class-representative vectors. In particular, it has been decided to use per-class first-order statistics collected on the training set. Indeed, assuming that the network can correctly classify most of the samples in the training set, the class-representative vectors would have the highest value at the corresponding class position. Therefore, since the legitimate sample $\boldsymbol{x}$ of class i produces a prediction vector with the same distribution of the other samples from X_i – i.e. the set containing all the training sample of class i – it is possible to write

$$\mathbb{E}_{\boldsymbol{x}\in X_i}[f_{-1}(\boldsymbol{x})] \approx f_{-1}(\boldsymbol{x}) \ , \tag{6.13}$$

where $\mathbb{E}[\cdot]$ represents the expectation operation. Since eqs. (6.7) and (6.10) hold true, $f_{-1}(\psi(\boldsymbol{x_1}))$ maintains a strong similarity with the corresponding class-representative vector, while $f_{-1}(\psi(\boldsymbol{x'_2}))$ results in a poor similarity score. Consequently, it is possible to substitute eq. (6.13) into eq. (6.11) and write

$$S\big(f_{-1}(\psi(\boldsymbol{x_1})), \mathbb{E}_{\boldsymbol{x}\in X_k}[f_{-1}(\boldsymbol{x})]\big) > S\big(f_{-1}(\psi(\boldsymbol{x'_2})), \mathbb{E}_{\boldsymbol{x}\in X_j}[f_{-1}(\boldsymbol{x})]\big) \ , \tag{6.14}$$

where k and j represent the class prediction of the original, non-distorted samples – i.e. $k = f(\boldsymbol{x_1})$ and $j = f(\boldsymbol{x'_2})$ – and X_k and X_j the sets containing all the legitimate training examples from class k and j, respectively. Equation (6.14) suggests the detection criteria in eq. (6.15), which proposes to detect adversarial examples by comparing the prediction vector on the distorted input replica with the class-representative vector

$$S\big(f_{-1}(\psi(\boldsymbol{x})), \mathbb{E}_{\boldsymbol{x}\in X_i}[f_{-1}(\boldsymbol{x})]\big) \underset{H_1}{\overset{H_0}{\gtrless}} \alpha \ . \tag{6.15}$$

Note how the class prediction of the original, non-distorted version of the input selects the proper class-representative vector to use – i.e. $i = f(\boldsymbol{x})$. This detection criteria results more stable than the one in eq. (6.12), as the representative vectors do not change across different samples of the same class. Moreover, since the representative vectors encompass more samples, they contain cross-class similarity patterns – e.g. natural dogs have 10% resemblance with cats. As a result, in order to evade detection, manipulated samples not only have to change the predicted class,

but must also conserve the similarity with other classes. Finally, by not using the prediction vector on the current sample, it is possible to avoid eventual undesired behavior produced by a malicious input during computation of the similarity score.

At this point, it has been noted that eq. (6.15) proposes to compare two different quantities: the prediction vector on the distorted sample $f_{-1}(\psi(\boldsymbol{x_1}))$ with the prediction vector first-order statistic, but computed on the original, non-distorted samples, i.e. $\mathbb{E}[f_{-1}(\boldsymbol{x})]$. However, although this represents an effective solution, it is not the most efficient one. By exploiting the property in eq. (6.13), it is possible to apply the expectation operation to both sides of eq. (6.10a) to get

$$\mathbb{E}_{\boldsymbol{x}\in X_i}\big[f_{-1}(\psi(\boldsymbol{x}))\big] \approx \mathbb{E}_{\boldsymbol{x}\in X_i}[f_{-1}(\boldsymbol{x})] \ . \tag{6.16}$$

Since the set X_i contains only legitimate samples from the training set, eqs. (6.7a) and (6.10a) hold true, so both sides of eq. (6.16) have a similar distribution – i.e. the highest entry is at the same location. Therefore, eq. (6.14) can be rewritten as

$$S\big(f_{-1}(\psi(\boldsymbol{x_1})), \mathbb{E}_{\boldsymbol{x}\in X_k}\big[f_{-1}(\psi(\boldsymbol{x}))\big]\big) > S\big(f_{-1}(\psi(\boldsymbol{x'_2})), \mathbb{E}_{\boldsymbol{x}\in X_j}\big[f_{-1}(\psi(\boldsymbol{x}))\big]\big) \ , \tag{6.17}$$

where the class-representative vector is changed to contain the first-order statistic of the distorted input replica. Equation (6.17) suggests the detection criteria in eq. (6.18), which proposes to detect adversarial examples by checking whether the distorted input replica produces a prediction vector with the same distribution of the distorted legitimate samples of the original class prediction

$$S\big(f_{-1}(\psi(\boldsymbol{x})), \mathbb{E}_{\boldsymbol{x}\in X_i}\big[f_{-1}(\psi(\boldsymbol{x}))\big]\big) \underset{H_1}{\overset{H_0}{\gtrless}} \alpha \ . \tag{6.18}$$

Note how this criteria reduces the risk of misclassification for legitimate samples, as the representative vector embeds information about how the specific distortion affects other samples of the same class. Yet, manipulated samples are still detected, as they do not show the same properties of natural samples after being distorted. Finally, note that the class predicted for the original sample is still used to select the proper class-representative vector.

Equation (6.18) provides the means to build a robust adversarial detector. However, it does not specify how to use multiple distortions to increase its effectiveness. The most straightforward solution consists in collecting the similarity scores produced by every distortions and retain only the minimum score. In this way, it is possible to detect an adversarial sample if at least one distortion is effective – i.e. results in eq. (6.10) true. On the other hand, this solution increases the amount of false positives – legitimate input classified as malicious –, as the score produced by the most aggressive distortion will be selected. To overcome this limitation, a combination technique has been designed to create a signature for the input, encoding the effects of all the distortions. In particular, given the set of distortions $\Psi = \{\psi_j : j \in [1..m]\}$, where m is the total number of distortions, a function $g(\cdot; f_{-1}, \Psi) : [0,1]^N \to [0,1]^{m\cdot d}$ is defined such that it concatenates the prediction vectors from all the distorted replica of the input, i.e.

$$g\big(\boldsymbol{x}; f_{-1}, \Psi\big) = \text{concat}\big(\{f_{-1}(\psi_j(\boldsymbol{x})) \ \ \forall\ j \in [1..m]\}\big) \ . \tag{6.19}$$

Equation (6.19) produces a signature that does not alter the content of the prediction vector $f_{-1}(\cdot)$, thus maintaining all the properties exploited so far. Consequently, it

is possible to rewrite eqs. (6.11)–(6.18) using the mappings in eq. (6.20), where $\mathcal{I} = \{I_j : j \in [1..m]\}$ is the set containing the identity functions that returns as output the input unmodified, i.e. $\boldsymbol{x} = I_j(\boldsymbol{x})$

$$f_{-1}(\psi(\boldsymbol{x})) \mapsto g\big(\boldsymbol{x}; f_{-1}, \Psi\big) \ , \tag{6.20a}$$

$$f_{-1}(\boldsymbol{x}) \mapsto g\big(\boldsymbol{x}; f_{-1}, \mathcal{I}\big) \ . \tag{6.20b}$$

In particular, it is possible to rewrite eq. (6.17) as

$$S\big(\gamma(\boldsymbol{x_1}), \mathbb{E}_{\boldsymbol{x}\in X_k}\big[\gamma(\boldsymbol{x})\big]\big) > S\big(\gamma(\boldsymbol{x'_2}), \mathbb{E}_{\boldsymbol{x}\in X_j}\big[\gamma(\boldsymbol{x})\big]\big) \ , \tag{6.21}$$

where $\gamma(\boldsymbol{x}) \triangleq g\big(\boldsymbol{x}; f_{-1}, \Psi\big)$ for brevity. Finally, eq. (6.21) suggests the detection criteria in eq. (6.22), which proposes to detect adversarial examples by comparing the signature extracted for the current sample with per-class representative signature vector

$$S\big(\gamma(\boldsymbol{x}), \mathbb{E}_{\boldsymbol{x}\in X_i}\big[\gamma(\boldsymbol{x})\big]\big) \underset{H_1}{\overset{H_0}{\gtrless}} \alpha \ . \tag{6.22}$$

Note how this criteria maintains the properties of the one suggested in eq. (6.18), but benefits from the usage of multiple distortions. Consequently, it results in a more robust detector, as malicious samples need to resist multiple distortions to avoid detection.

6.2.2 Class Representative Vectors

The system proposed herein exploits class representative vectors to detect adversarial examples. In order to produce these vectors, an offline operation is necessary. In particular, the legitimate samples used to train the model are processed to extract the signature according to eq. (6.19). Then, samples belonging to the same ground-truth class are grouped together and their signature averaged to compute the first-order statistic. Equation (6.23) contains the formula to produce the representative vectors $\boldsymbol{\mu}$, where $\gamma(\cdot) \triangleq g\big(\cdot; f_{-1}, \Psi\big)$ is the signature function in eq. (6.19) and $|\cdot|$ the set-size operator returning the number of samples in a set

$$\boldsymbol{\mu_j} = \frac{1}{|X_j|} \sum_{\boldsymbol{x}\in X_j} \gamma(\boldsymbol{x}) \quad \forall\ j \in [1..d] \ . \tag{6.23}$$

As shown in section 6.4.5, class representative vectors increase the stability of the detection process. Moreover, since they are computed offline, they do not affect the inference timing performance, as they are stored in memory, ready to be used. Additionally, the procedure in eq. (6.23) does not require any adversarial examples, thus retaining the attack-agnostic property of the proposed system unaffected.

6.2.3 Similarity Score Function

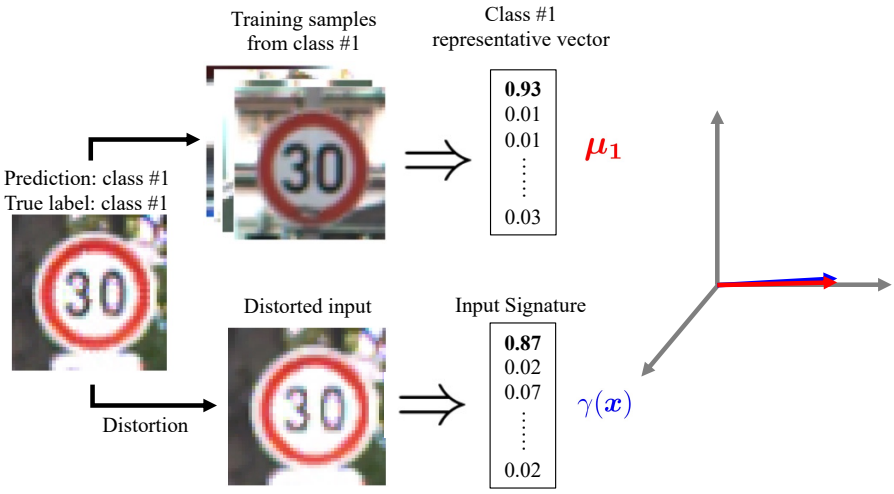
Equation (6.22) proposes to perform the detection task by comparing the input signature with the corresponding class-representative vector. This requires the computation of a detection score, encoding differences and similarities between the two vectors. Although every similarity metric is eligible to fulfill this task, however, accurate selection of this function is a key aspect to achieve superior performance.

The literature suggests to compute the detection score by means of the L_1–distance [86]. This metric captures the discrepancy between two vectors by calculating the sum of the element-wise differences. Formally, given two vector-valued variables $\boldsymbol{a}$ and $\boldsymbol{b}$, the L_1–distance score is defined as

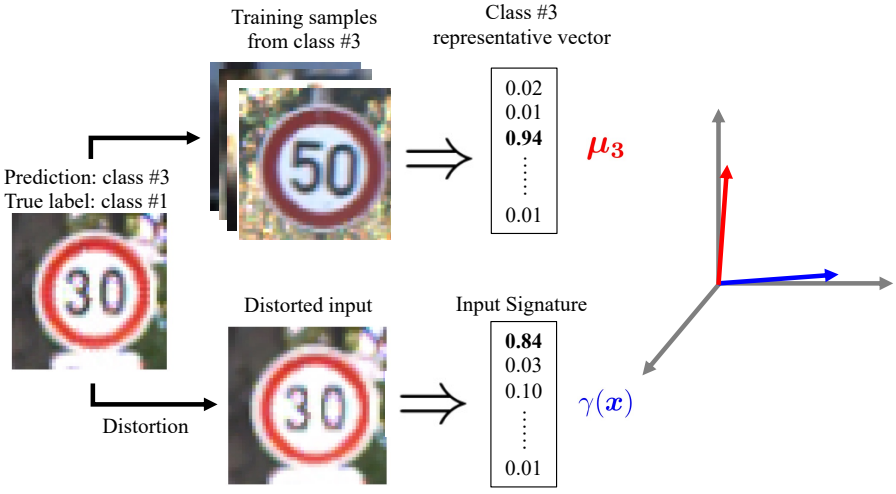
$$L_1(\boldsymbol{a}, \boldsymbol{b}) = \|\boldsymbol{a} - \boldsymbol{b}\|_1 = \sum_i |a_i - b_i| \ , \tag{6.24}$$

where $\|\cdot\|_1$ is the L_1–norm and $|\cdot|$ the absolute value operation. Note that this metric does not compute the similarity but the distance between vectors. Nevertheless, it can be used to produce the detection score, as eqs. (6.12), (6.15), (6.18) and (6.22) continue to hold true for $S(\cdot, \cdot) = -L_1(\cdot, \cdot)$. The L_1–distance score results particularly effective when comparing two well-shaped prediction vectors with high confidence for the predicted class. However, they do not efficiently tackle poor confident prediction vectors, especially when affected by cross-class confusion. Finally, a main drawback of this metric consists in the effect produced by entries with low values in both vectors: they need to be exactly the same, otherwise they will still provide a contribution, increasing the detection score. Despite this limitation has minor impact on small classification tasks (small d), it is very critical for high dimensional prediction vectors, where many low entries are present – e.g. German Traffic Sign Recognition Benchmark (GTSRB) classification task with 43 exclusive output classes.

By analyzing the characteristics of the vectors in eq. (6.22), it has been noted an interesting property: adversarial examples tend to generate signatures *orthogonal* to the corresponding class-representative vector. At the same time, legitimate samples result in signatures that are aligned with their class-representative vector. This effect is illustrated in fig. 6.2. Consider, without loss of generality, to use a single distortion during the detection process. In this case, a legitimate input produces a signature with the highest feature at the true class position. Since the selected class-representative vector contains statistics of prediction vectors generated by samples of the same class, they share the highest entry location. Therefore, the two vectors result aligned. An example is provided in fig. 6.2A, where the legitimate 30 km/h speed-limit sign is correctly classified as class 1. Since the representative vector is built-up using training images containing the same traffic sign, it still produces the highest entry for class 1. Contrarily, a malicious sample yields a signature that has its highest entry at a different location than the original input's class prediction. Since this latter is used to select the class-representative vector, the two vectors have the highest entry at different locations. Moreover, bearing in mind that prediction vectors sum-up to 1, the remaining entries of the two vectors contain values close to 0, hence they are orthogonal to each other. Figure 6.2B contains an example. The malicious 30 km/h speed-limit sign sample generates an adversarial class prediction in class 3. Since the distortion changes the network prediction, its signature contains the highest value for class 1. Yet, it is compared with the representative vector of class 3, which has the highest score at class 3. Therefore, the two vectors align along different axis and, as a consequence, they are perpendicular to each other. Finally, note how the signature function in eq. (6.19) does not change the position of the highest entry but just appends another prediction vector with similar characteristics. As this operation does not affect the orthogonality property, this derivation can be generalized to configurations using multiple distortions.



(A) Legitimate samples alignment.



(B) Adversarial examples orthogonality.

Figure 6.2: Orthogonality property of eq. (6.22). Legitimate samples produce signatures aligned with their class-representative vector. Adversarial examples result in signatures perpendicular to the class-representative vector selected. Best viewed in color.

To exploit the orthogonality property, it is important to adopt a similarity metric that takes into account the relative geometrical position of the two vectors. For this purpose, it has been decided to use the *normalized projection score*, a.k.a. *cosine similarity score*. This metric function performs a dot-product between the two vectors and normalizes the result so that it is in the interval $[0, 1]$. Equation (6.25)

contains the formal definition, where $\boldsymbol{a}$ and $\boldsymbol{b}$ are column vectors and $\|\cdot\|_2$ is the L_2–norm

$$proj(\boldsymbol{a}, \boldsymbol{b}) = \frac{\boldsymbol{a}^T \boldsymbol{b}}{\|\boldsymbol{a}\|_2 \cdot \|\boldsymbol{b}\|_2} = \frac{\sum_i a_i \cdot b_i}{\sqrt{\sum_i a_i^2} \cdot \sqrt{\sum_i b_i^2}} \ . \tag{6.25}$$

Technically, the dot-product projects the vector $\boldsymbol{a}$ onto $\boldsymbol{b}$, providing a summary about the common components of the two vectors. A different interpretation consists in considering the score as the cosine of the angle between the two vectors. For this reason, the normalized projection score encodes the orthogonality property straightforwardly. In addition, it solves the shortcoming of the L_1–distance score with low-value entries. Indeed, it multiplies together – rather than subtracting – values close 0, thus drastically limiting their contribution to the final score.

Equation (6.26) contains the detection score computed by the proposed detector, where $\boldsymbol{a}$ and $\boldsymbol{b}$ are replaced with $\gamma(\boldsymbol{x})$ and $\boldsymbol{\mu_j}$, i.e., respectively, the input signature and the representative vector of class $j = f(\boldsymbol{x})$ – the class prediction on the original input $\boldsymbol{x}$

$$s(\boldsymbol{x}) = proj(\gamma(\boldsymbol{x}), \boldsymbol{\mu_j}) = \frac{\gamma(\boldsymbol{x})^T \boldsymbol{\mu_j}}{\|\gamma(\boldsymbol{x})\|_2 \cdot \|\boldsymbol{\mu_j}\|_2} \ . \tag{6.26}$$

As a consequence, legitimate samples, producing signatures that align with the corresponding class-representative vector, result in high detection scores. Conversely, the closer to 0 the score is, the more likely it is that the input is malicious. This is illustrated in fig. 6.3. It contains the histogram of detection scores generated using eq. (6.26) on the GTSRB test-set [29]. Adversarial examples – generated using the C&W method – result in poor scores, with more than 60% of them showing a score close to 0. At the same time, more than 80% of legitimate samples exhibit a score close to 1, thus proving the validity of the proposed detection framework.

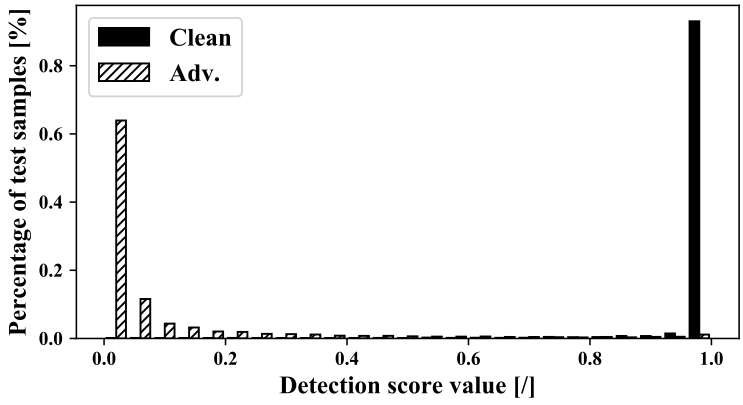


Figure 6.3: Histogram of detection scores extracted using eq. (6.26) for legitimate (black) and adversarial (hatched fill) samples. Inputs are sampled from the GTSRB test-set and perturbed using the C&W method. A single distortion is used (median smoothing).

6.2.4 Distortions

The formal derivation proposed in section 6.2.1 assumes the effectiveness of the distortions involved in the detection process. In order to be effective, a distortion must respect the properties in eq. (6.7), i.e. it must be such that:

- distorted legitimate samples are still correctly classified.
- distorted adversarial examples are no longer malicious, resulting in the correct class prediction.

Note that, to avoid trivial solutions, it is important that the distortion exhibits both properties. As an example, consider the identity function that outputs its input unmodified. Although this distortion complies with the first property, it has no effects on adversarial examples, thus resulting in missed detections.

Unfortunately, however, it is difficult to find distortions which possess the properties in eq. (6.7). Nonetheless, since the proposed detector utilizes prediction vectors to detect adversarial examples, it is possible to relax these properties. Indeed, it requires that eq. (6.10) holds true, therefore, the distortion must be such that:

- distorted legitimate samples produce prediction vectors similar to their original (non-distorted) version
- distorted adversarial examples produce a prediction vector considerably different than the original sample

Note how the new properties are less strict. Indeed, the first condition requires that the distortion has minimal effects on the outcome produced by legitimate samples. Meanwhile, the second property demands that the distortion substantially changes the prediction vector produced by adversarial examples. Since it is not expected to recover the original class prediction, the introduction of high element-wise difference would suffice to enable detection of malicious samples.

The new properties broaden the set of distortions well-suited for the detection task. Good candidates are distortions that NNs are robust to. At the same time, however, it is important that they attack the adversarial perturbation pattern. The remainder of this section describes the distortions used in the proposed approach: i.e. spatial smoothing, reduction of bit-depth resolution and gray-scaling. Figure 6.4 shows their effect on a legitimate example from the GTSRB dataset.

(A) Original sample. (B) Spatial smoothing. (C) Reduction of bit-depth resolution. (D) Gray-scaling.

Figure 6.4: Visual effects of the distortions on a legitimate sample of the GTSRB dataset. Best viewed in color.

Spatial Smoothing A well-known characteristic of NNs is their robustness to greedy computer vision distortions. Therefore, *spatial smoothing* represents the most simple candidate for the proposed detection framework. This distortion, also known as *image blurring* or *low-pass filtering*, is widely used to reduce the noise present in an input image. Since the adversarial perturbation can be considered as a special noise added to the input, it results particularly effective for attacking malicious patterns. Spatial smoothing operates by setting the value of every pixel according to statistics collected over the neighboring pixels. In this way, it is possible to achieve a low-pass filtering effect, smoothing out high-frequency components present in the adversarial perturbation in the form of abrupt changes across neighboring pixel values.

There exist several methods to achieve the smoothing effect. The most simple implementation consists in setting the pixel intensity to the mean value of the nearby pixels. *Gaussian smoothing* performs the same operation by weighting the neighboring pixels with a 2D Gaussian filter. *Median smoothing* collects the pixels in a given window and set the intensity of the center pixel to the median ranking value. Although every technique has its own advantages and disadvantages, median smoothing proved to be the most effective against adversarial examples [86]. This particular distortion is well-suited for removing local outlier pixel values while retaining important details like edges and corners. Figure 6.4B shows the effect of median smoothing on a legitimate sample from the GTSRB dataset. The proposed detection framework uses median smoothing with square window of size 2 pixels.

Reduction of Bit-Depth Resolution NNs assume their inputs to be continuous. However, real-world systems have memory constraints that force the use of discretized data. For instance, images are represented as arrays of pixels with associated (discrete) intensity values. Common configurations are 8-bit (gray-scale) and 24-bit (color, 8-bit per channel) intensity resolution. Therefore, every pixel of a gray-scale image hold a discrete value in the integer interval $[0\,..\,255]$, encoding a specific shade of gray – ranging from black (0) to white ($255 = 2^8 - 1$). Similarly, color images encoded using the RGB-format, use 3 color channels (red, green and blue) of 8-bit each. Studies have shown that, from a visual perspective, small variations of the pixels intensity do not affect the input content, thus suggesting that the bit-depth resolutions used are unnecessary large [86]. Indeed, although high color-resolutions provide more natural visualizations, they are not crucial to make sense of images – e.g. to recognize objects of interest. Therefore, it is possible to reduce the bit-depth resolution without significantly hurting the performance of NNs. At the same time, by limiting the input resolution, it is possible to reduce the adversarial opportunity, thus disrupting eventual malicious patterns.

Following the suggestions in [86], it has been implemented the distortion as in eq. (6.27), where $\boldsymbol{x} \in [0,1]^N$ is the input with original bit-resolution normalized to the interval $[0,1]$, b the output bit-depth resolution and $int(\cdot)$ the integer-rounding operation

$$\widetilde{\boldsymbol{x}_{\boldsymbol{b}}} = \frac{int(\boldsymbol{x} \cdot (2^b - 1))}{2^b - 1} \ . \tag{6.27}$$

As a result, the distorted input $\widetilde{\boldsymbol{x}_{\boldsymbol{b}}}$ is re-quantized, reducing the information capacity to b bits. In the proposed detection framework b is set to 4 for every color-channel. Figure 6.4C shows an example of legitimate input from the GTSRB dataset with bit-depth resolution reduced from 8 to 4 bit per channel.

Gray-scaling Humans do not need colors to recognize objects of interest. Indeed, a gray-scale version of a colored image still retains prominent patterns like texture, corners and edges. Gray-scale images are the result of a combination of the color channels. Over the years, several different methods have been developed to perform this operation [25]. The most simple solution consists in reducing the three RGB color channels by averaging out the corresponding pixel intensity values. More advanced techniques perform a weighted combination with special weighting factors in order to maintain the most salient information for a given task. They are designed to focus on specific characteristics of the gray-scale image such as perception to human eye or optimization for digital visualization. As an example, ITU-R Recommendation BT. 709 [47] defines a standard for gray-scale image visualization on high-definition television. This standards accentuates the contribution of the green channel as humans are more sensible to this color spectrum.

Similarly to a reduction of the bit-depth resolution, by gray-scaling an image it is possible to limit the adversarial opportunity while maintaining the input content. Therefore, the proposed framework adopts the color-to-gray-scale distortion using the implementation in eq. (6.28), where k_r, k_g and k_b are constants and $\boldsymbol{R}$, $\boldsymbol{G}$ and $\boldsymbol{B}$ are the red, green and blue color channels, respectively

$$\widetilde{\boldsymbol{x}} = k_r\boldsymbol{R} + k_g\boldsymbol{G} + k_b\boldsymbol{B} \ . \tag{6.28}$$

Note how, despite eq. (6.28) also performs a reduction of the bit-depth resolution, it differs from eq. (6.27), as it combines the color channels rather than re-quantize them. It has been decided to follow the recommendations in [47], setting $k_r = 0.2125$, $k_g = 0.7154$ and $k_b = 0.0721$. In addition, the gray-scaled image is replicated along the three color channels to retain the input dimensionality and, consequently, the information flow within the NN. Figure 6.4D shows the effect of this distortion on a legitimate sample of the GTSRB dataset.

6.2.5 Detection Framework

Figure 6.5 contains the detection framework proposed herein, implementing the criteria derived in eq. (6.22). When the defended network receives an input, a set of distortions are applied to generate the distorted replicas. At this stage, the system has to generate the input signature. Therefore, it queries the model with the distorted replicas, collects the prediction vectors and combines them using eq. (6.19). At the same time, the original version of the input is fed to the model to obtain the class prediction. This is used to select the class representative vector which will be compared with the current input signature. The normalized projection score is used to compute the detection score. Finally, the score is thresholded to infer the input's nature and decide whether to discard or maintain the predicted class.

6.3 Experimental Setup

This section describes the setup used to conduct the experiments. Throughout the examination, the guidelines for a correct and robust evaluation provided in [90] are rigorously followed.

The detection results are reported in terms of area under the curve (AUC) in order to be independent from the threshold level. In particular, for each experiment,

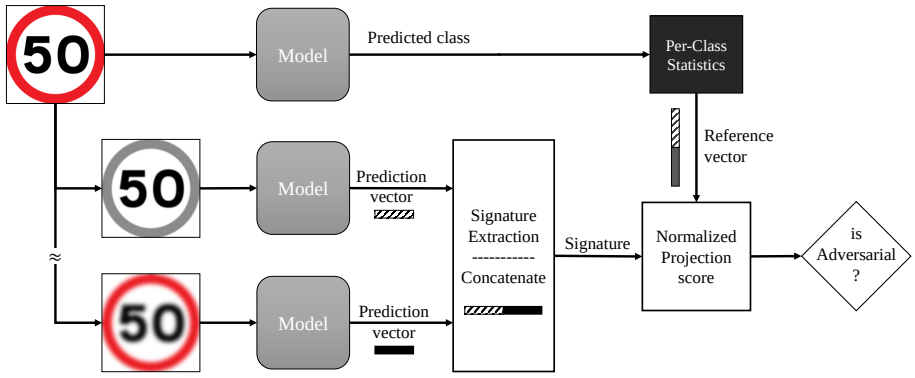


Figure 6.5: Scheme of the proposed detection framework. The input is distorted to extract its signature. The original input class prediction is used to select the class-representative vector. The normalized projection metric computes the detection score. A thresholding operation determines whether the input is legitimate or adversarial.

the attack-set – containing only successful malicious samples – is paired with an equal number of correctly predicted samples from the legitimate test-set. Then, the Receiver Operating Curve (ROC), plotting true positive rate – adversarial examples successfully detected – versus false positive rate – legitimate samples wrongly classified as malicious –, are extracted to collect the AUC scores. Due to the limited amount of samples, for black-box experiments the AUC scores would be an inaccurate metric, therefore, detection rates are used. They report the percentage of successful malicious samples predicted at a given threshold value. The classification performance of the models is measured in terms of accuracy.

Unless otherwise stated, every detector is tested in a two-distortions configuration, using the distortions found in literature – i.e. median smoothing and reduction of the bit-depth resolution. Section 6.4.3 reports the results collected using different distortion-configurations, validating the gray-scale distortion proposed herein. The next sections describe the datasets and attack-methods used, as well as the method from literature adopted as benchmark and the victim classification models.

6.3.1 Datasets

The main application of the method proposed herein consists in effectively defending an autonomous vehicle from adversarial attacks. However, it is important to evaluate it on a more broad and general task to fully assess its performance. Therefore, two diverse datasets are used to validate the proposed solution: GTSRB [29] and CIFAR10 [26].

GTSRB Traffic sign recognition is a fundamental task for autonomous vehicles. However, traffic signs are exposed to adversarial perturbations which might deceive the classification system [83, 69]. It is, therefore, crucial to have in place solutions that can defend against eventual attacks. To evaluate the performance of the proposed adversarial detector on such tasks, the German Traffic Sign Recognition

Benchmark (GTSRB) dataset is used. It features more than 60k traffic sign images under various conditions. Every image contains a single traffic-sign and is labeled according to one of 43 classes – such as stop, 30 km/h speed-limit, no entry, etc. Therefore, it enables one to address a multi-class classification problem. The dataset is split into training and test set, consisting of 40k and 13k samples, respectively. The samples are represented using the RGB format. Since the inputs have varying sizes, every image is resized to 46×46 pixels before feeding it to the model.

CIFAR10 ML researchers rely on publicly available datasets to develop and evaluate algorithms. Due to the low-resolution of its samples – enabling fast experimentation –, CIFAR10 has been used as benchmark for many ML solutions. It consists in a collection of images featuring ordinary objects such as cars, birds, dogs, etc. Every image contains a single object from 10 different classes. The dataset is divided into a training split of 50k samples and a test split of 10k samples. The images are encoded using the RGB-format and have a fixed size of 32×32 pixels.

6.3.2 Adversarial Attacks

The selection of the attack techniques represents a crucial phase for the correct evaluation of any defensive strategy. Adversarial attacks have different characteristics, therefore, it is important to use diverse methods to thoroughly assess the performance of the system. For instance, a defense solution can prove effective against aggressive attacks but vulnerable and powerless to finely perturbed inputs. For these reasons, it has been decided to validate the proposed detection framework using three attacks: C&W, DF and FGSM. Figure 6.6 showcases a legitimate sample form the GTSRB dataset perturbed with every attack-method.

C&W Carlini and Wagner (C&W) represents the state-of-the-art among adversarial attacks and it is used as benchmark to assess the performance of new defensive solutions. The malicious perturbations produced by this method are very fine and results invisible to humans. In addition, C&W provides the attacker with full control over the generation process. During experimentation, it has been used the untargeted L_2–norm attack, which uses eq. (6.5) with the L_2–norm and function $g(\cdot)$ defined as in eq. (6.29), where $f_{-1}(\boldsymbol{x'})_i$ is the i-th entry of the prediction vector and $j = f(\boldsymbol{x})$ – i.e. the class predicted for the clean version of the input

$$g(\boldsymbol{x'}) = \max(f_{-1}(\boldsymbol{x'})_j - \max(\{f_{-1}(\boldsymbol{x'})_i : i \neq j\}), -\kappa) \ . \tag{6.29}$$

(A) Legitimate sample. (B) C&W. (C) DF. (D) FGSM.

Figure 6.6: A GTSRB sample attacked with the methods used for evaluation. Best viewed in color.

Note how the function in eq. (6.29) finds its minimum in $-\kappa$ when there is at least a class with prediction scores higher – by a margin of κ – than the original predicted class. Therefore, the parameter κ allows tuning of the attack outcome, as higher values produce more confident predictions. Different configurations of the C&W attack are adopted, using different values of κ. The value used for this parameter is reported with a number beside C&W – e.g. C&W5 has $\kappa = 0.5$. No number stands for $\kappa = 0$. Figure 6.6B contains a GTSRB sample attacked with the C&W method.

DF DeepFool (DF) is a very powerful attack method. It generates adversarial examples with the smallest perturbation possible. It has been decided to use this attack because of its ability to produce very fine malicious samples, altering the original input imperceptibly. The parameters are set as suggested in the original paper [50]. As DF does not allow for customization, a single configuration of this attack is used. Figure 6.6C shows a GTSRB sample perturbed using the DF method.

FGSM The Fast Gradient Sign Method (FGSM) is a fast and greedy attack technique. It generates malicious samples often visible to humans, because of a very aggressive perturbation pattern. Yet, it features different characteristics than the previous methods, thus providing a good testing ground for defensive strategies. The attack has been implemented as indicated in eq. (6.3), setting the step size parameter ϵ to 0.01 and 0.04 – FGSM1 and FGSM4, respectively. Figure 6.6D displays the effect of the FGSM attack on a GTSRB sample.

To obtain a comprehensive measure of the proposed detector performance, all the adversarial attacks are deployed in both white- and black-box settings. In white-box scenario, it is assumed that the attacker has access to the classification model, but not to the defensive technique. Instead, to get black-box malicious samples, it is attacked a substitute model trained on the same task. Then, the resulting adversarial examples are used to attack the original classifier. Table 6.1 reports the performance of victim and substitute models on all the attack-sets.

6.3.3 Prior Art Benchmark

Comparison with other prior art methods is critical to assess the full potential of the proposed defensive strategy. However, it is important to use methods that perform the same task with similar resources, in order to enable fair comparisons. For this reason, it has been decided to use Feature Squeezing (FS) [86] in the evaluation process, as it represents the closest solution to the proposed approach. Indeed, they both are low-cost, attack-agnostic adversarial detectors that are well-suited to be deployed along with other defensive solutions – such as adversarial training [41]. Technically, FS adopts a structure similar to the proposed detection framework. It uses distortions to attack the adversarial pattern and compares the prediction vectors of the distorted replicas with the prediction vector of the original sample, as in eq. (6.12). Differently than the proposed approach, it uses the L_1–distance to compute the detection score. Finally, it does not have a signature extraction function: it fuses the effects of multiple distortions by using the maximum operator – thus selecting the most destructive distortion. The interested reader is referred to section 6.1.2 or the original paper for further details.

6.3.4 Classification Models

In this section are described the victim models defended by the proposed detector.

The model deployed for the GTSRB classification task is a CNN. It has two convolutional layers with kernel size 3×3, stride 2 and 8, 16 output channels, respectively. Rectified Linear Unit (ReLU) is used as activation function. Finally, a dropout layer with rate 0.5 is applied to the output feature-map and an FC-layer is used to regress the (43) class scores with the softmax activation function. The model used to craft the black-box adversarial examples has the same structure, but with an additional convolution layer – 3×3 kernel, stride 1 and 32 channels.

Concerning CIFAR10, it has been decided to use a less accurate classifier than the state-of-the-art. In this way, it is possible to evaluate the proposed model under different conditions. Indeed, the more uncertain the victim is, the less accurate its statistics are, thus potentially affecting the detector operations. It has been decided to use a CNN model, with three 3×3 convolutional layers: the first with 8 channels and stride 1, the remaining two with stride 2 and 32, 64 channels, respectively. ReLU is used as activation function. A dropout layer followed by an FC-layer conclude the model, which provides as output 10 units holding the class scores – through the softmax function. The black-box version of the model exhibits the same structure, with the addition of a convolution layer with stride 1 and 128 channels.

All models are trained using the cross-entropy loss function with learning rate of 0.001, iterating over the dataset until convergence. Table 6.1 reports the performance of victim and substitute models on the legitimate test-set.

6.4 Results

Table 6.1 reports the performance of the victim classifiers on both GTSRB and CIFAR10. The top section contains the results against the white-box attack-sets. Note how the networks performance are heavily affected by the adversarial attacks. A very accurate network trained on the GTSRB dataset, achieves only 4% circa of accuracy when queried with the C&W or DF test-set. It means that most of the samples are able to deceive the network into predicting a wrong class. The FGSM attack is also able to reduce the model accuracy, but its effectiveness is decisively lower. On the other hand, FGSM produces highly confident predictions, as stated by the large average prediction confidence values. This is due to the fact that FGSM is a very aggressive attack: it does not always succeed, but when it does it produces high confidence scores. Another observation that can be collected from table 6.1 concerns the parameter κ of the C&W attack. Indeed, the table results confirm that this parameter enables tuning of the attack uncertainty, as the higher it is, the higher is the average prediction confidence value.

Similar considerations can be repeated for the CIFAR10 model. However, here the FGSM attacks result more effective, producing the lowest accuracy score. This effect attributed to two reasons: the lower model accuracy and the lower input resolution. Indeed, less accurate models are more easily fooled, due to the blurrier decision boundaries. Moreover, as the input resolution becomes smaller, FGSM approaches to the other more advanced methods that exploit the high input dimensionality to find "fooling" regions[6].

[6]On a single pixel input, FGSM is equivalent to a DF attack, up to a scaling factor.

Table 6.1: Performance of the tested classifiers against every test-set (legitimate and adversarial) with no defense deployed. Accuracy and average prediction confidence are reported in percentage. Top: White-box results. Bottom: Black-box results.

	Metrics	Legitimate Samples	White-box C&W	C&W5	C&W9	DF	FGSM1	FGSM4
CIFAR10	Accuracy (victim)	76.83	13.32	13.21	13.21	13.97	21.08	12.66
	Avg. Prediction Confidence (victim)	84.14	44.86	55.79	63.88	44.58	74.83	78.87
GTSRB	Accuracy (victim)	92.88	3.66	3.66	3.64	3.52	42.19	20.58
	Avg. Prediction Confidence (victim)	96.88	51.55	63.17	71.41	48.45	83.90	91.58

	Metrics	Legitimate Samples	Black-box C&W	DF	FGSM4
CIFAR10	Accuracy (victim)	76.83	75.68	75.63	38.50
	Avg. Prediction Confidence (victim)	84.14	59.30	59.31	65.39
	Accuracy (substitute)	79.11	11.80	12.84	11.89
GTSRB	Accuracy (victim)	92.88	92.71	92.66	78.97
	Avg. Prediction Confidence (victim)	96.88	63.58	63.35	66.95
	Accuracy (substitute)	96.72	1.72	2.03	20.47

The bottom part of the table reports the results on the black-box attack-sets. Note how the attacks are very effective against the substitute model. However, only FGSM shows good transferability properties – reducing its accuracy – whilst C&W and DF fail to generalize to the victim model – producing less than 2% accuracy drop. Yet, although C&W and DF cannot deceive the model into a wrong classification output, they are not completely ineffective. Indeed, they considerably reduce the average prediction confidence, thus resulting in very uncertain predictions.

6.4.1 White-box Results

Figure 6.7 contains a comparison between the proposed detector and FS in terms of ROC curves. The white-box C&W attack-set on the CIFAR10 dataset is used to collect the results. The plot shows that the proposed method considerably outperforms FS, generating a ROC curve that passes closer to the top-left corner. In addition, the figure reveals that the proposed detector possesses better convergence properties than FS, as shown in the top-right section of the plot, highlighting a flat behavior of the curve. This means that it is possible to achieve 100% detection rate, while producing a lower amount of false positives. Finally, the AUC score in the legend numerically quantifies the superiority of the proposed approach on this specific configuration, outperforming FS by more than 15%.

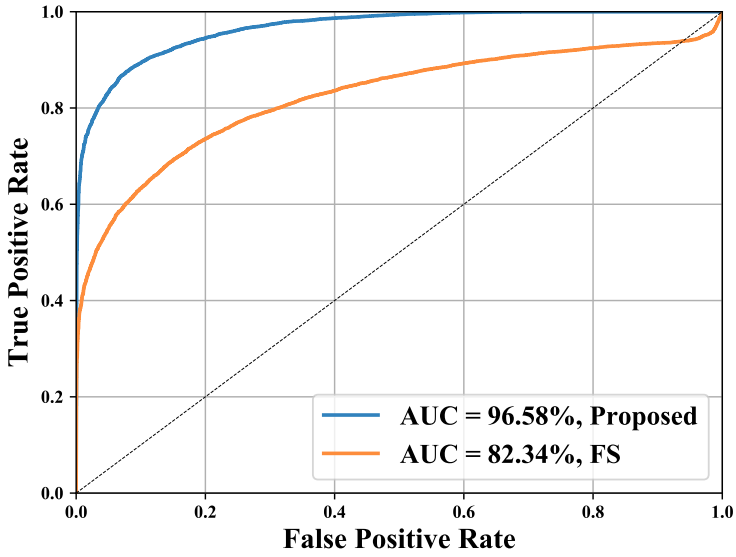


Figure 6.7: ROC curves of the proposed detector (blue) and FS (orange) against the C&W attack on the CIFAR10 test-set. Both the detectors use median filtering and reduction of the bit-depth as distortions. Best viewed in color.

Table 6.2 reports the AUC scores generated on all the white-box attack-sets by the two methods. It supports the conclusion drawn from fig. 6.7, confirming the superior performance of the proposed solution in detecting the advanced attacks. Note, indeed, how the AUC score for the C&W and DF attack-sets are always above 95%. Another interesting observation can be made from the results collected on the different datasets. Though the proposed detector performance are almost unchanged, FS experiences a significant drop when defending the CIFAR10 classifier. This indicates that FS is severely affected by the model accuracy: it can successfully detect high-confidence adversarial examples but struggles to cope with uncertain ones. On the other hand, the proposed detector proves to be reasonably robust to the model accuracy, hence resulting well-suited to defend a broader variety of classifiers. A similar pattern can be found in the performance of the two methods against the different C&W attack-configurations. Indeed, notice how FS improves its performance as the value of the parameter κ increases, while the proposed detector remains almost unaffected by it. Bearing in mind that the higher κ is, the higher is the malicious samples confidence – as shown in table 6.1 –, this effect supports the previous considerations. In addition, it suggests that the proposed solution is more effective than FS to defend against various configurations of the C&W attack.

Finally, it is worth commenting the performance on the FGSM attack-sets. Table 6.2 shows that FS is more effective against such greedy techniques than the proposed method. This can be partially attributed to the higher prediction confidence generated by FGSM adversarial samples. Moreover, it is surprising how the FGSM4 attack-set generates the worst detection performance in both methods. This is due to the high aggressiveness of the attack-technique, which often

Table 6.2: AUC scores of FS and the proposed detector against the various attack-sets (%).

	Method	C&W	C&W5	C&W9	DF	FGSM1	FGSM4
CIFAR10	FS	82.34	86.62	89.45	82.21	**88.89**	**74.90**
	Proposed	**96.58**	**95.57**	**95.31**	**97.00**	84.17	67.52
GTSRB	FS	96.55	97.76	98.39	96.16	**93.74**	73.60
	Proposed	**99.77**	**99.69**	**99.61**	**99.62**	92.54	**74.08**

produces perturbations visible to human eyes and, thus, robust to the distortions adopted. Nevertheless, it is possible to improve the detection performance on the FGSM attack-sets by combining other complementary defensive solutions, as shown in section 6.4.2.

6.4.2 Compatibility with other Defensive Solutions

Section 6.4.1 proved the ability of the proposed solution to detect malicious samples generated by advanced techniques such as C&W and DF. It also turned out that it suffers when attacked with more aggressive methods – like FGSM –, which are sometimes able to evade detection. However, there exists a large number of defensive solutions that can successfully handle such adversarial attacks. This section shows how it is possible to leverage the compatibility property of the proposed method to solve this shortcoming and improve its robustness.

The proposed method possesses the property of being an adversarial detector that does not alter the classifier inference procedure. As such, it can be considered an external module that performs a task detached to that victim's one. This characteristic enables the proposed detector to be deployed with other complementary defensive strategies, operating on different weaknesses of adversarial examples. An example can be found in the HGD denoiser developed in [75]. It can be employed to clean the input sample of the malicious perturbation. Yet, the denoiser-classifier system can be interpreted as the new victim model and the proposed adversarial detector can be deployed to defend it. In this way, if the malicious input evades the denoiser, this can still be recognized by the external detector. Therefore, it is possible to achieve a defensive strategy that combines the qualities of two different techniques, thus enhancing both effectiveness and robustness.

In this section, it has been decided to experiment with the adversarial training defensive method [41]. It defends the victim model by including adversarial examples during the training process. In this way, the victim can store knowledge about them and, consequently, be less exposed to adversarial perturbation patterns. Further details are provided in section 6.1.2 or the original paper [41].

Since the detector proved strong against both C&W and DF, it has been decided to use adversarial training to improve the detection performance on the FGSM attack. Therefore, once trained the classifier until convergence, it has been fine-tuned

for few epochs using the adversarial training procedure. Specifically, the mini-batch of legitimate samples used during training has been extended with the same amount of malicious samples. These samples are generated by attacking legitimate inputs with the FGSM4 method. Note that the malicious perturbation is crafted using the current network configuration as target model. The fine-tuning process has been set to last 20% of the initial training procedure. Finally, to comply with the white-box requirements, every adversarial attack-set has been re-computed on the finalized version of the victim network – i.e. after adversarial training.

Table 6.3 shows the effect of adversarial training on the victim models. It can be noted that this defensive solution has the same effect on both tasks (CIFAR10 and GTSRB). As expected, adversarial training has the strongest impact on the FGSM attack-sets, significantly reducing their success-rate. This suggests that it succeeds at teaching the network the perturbation pattern of the FGSM method. With this knowledge, the network can smooth out the input domain around the legitimate samples – in the direction of the FGSM perturbation – and limit its exposure to adversarial examples. Interestingly, adversarial training also reduces the fooling-rates of C&W and DF, though the effects are marginal. This supports the previous consideration: C&W or DF perturbations that resemble FGSM ones have a reduced effect on the network. Finally, it is worth noting that the performance of the classifier on the legitimate test-set is slightly harmed by adversarial training. However, this effect can be prevented by carefully fine-tuning the adversarial training process.

Table 6.4 reports the results of both FS and the proposed detector deployed together with adversarial training. As reference, it also contains the results – from table 6.2 – of the two methods without adversarial training the victim network. It is interesting to notice the different behavior of the two methods. Indeed, FS produces almost always poorer detection performance when defending the adversarially trained network. Contrarily, the proposed solution benefits from the cooperation with adversarial training, often improving its detection capabilities. This effects can be attributed to two reasons. On one hand, it has to be considered that adversarial training diminishes the effectiveness of the attacks. This results not only in a lower number of samples fooling the network, but also in lower prediction confidence values. Since section 6.4.1 proved that FS is sensitive to this setting, it explains the poor compatibility property of FS. On the other hand, by using the first-order statis-

Table 6.3: Performance of the victim models with and without adversarial training in terms of accuracy (%).

	Advers. Training	Legitim. Samples	C&W	DF	FGSM1	FGSM4
CIFAR10	without	76.83	13.32	13.97	21.08	12.66
	with	73.58	14.18	14.29	46.65	28.16
GTSRB	without	92.88	3.66	3.52	42.19	20.58
	with	91.95	4.31	4.22	73.49	31.80

Table 6.4: AUC scores of the proposed detector and FS with and without adversarial training (%).

	Method	Advers. Training	C&W	DF	FGSM1	FGSM4
CIFAR10	FS	without	82.34	82.21	**88.89**	74.90
		with	81.40	80.73	79.81	72.50
	Proposed	without	96.58	**97.00**	84.17	67.52
		with	**96.65**	96.73	86.48	**78.41**
GTSRB	FS	without	96.55	96.16	93.74	73.60
		with	88.39	88.61	92.20	89.45
	Proposed	without	**99.77**	**99.62**	92.54	74.08
		with	93.26	93.26	**95.77**	**92.81**

tic as class representative vector, the proposed method incorporates knowledge of the victim into the detection process. This enables it to encode information about the decision region instantiated by the network and leverage the effects produced by adversarial training. In this way, the proposed method takes advantage from the cooperation, improving the detection performance.

Another intriguing result can be found in the performance of the proposed solution against the FGSM attack-sets. Note how adversarial training, despite reducing the effectiveness of the FGSM attacks, also improves the detection performance. In this way, the proposed method is able to fill the gaps with FS, performing better or on-par with it. Finally, it is worth noting the slight performance degradation against both C&W and DF attack-sets. Following the consideration in section 6.4.1, it is possible to attribute this effect to the drop in accuracy of the adversarially trained networks. Yet, the proposed method still outperforms FS, achieving better detection scores.

6.4.3 Distortions Configuration

The detectors tested so far always used two distortions. However, they are not limited to this configuration. In this section, various distortion-configurations are tested to assess their impact on the performance and study the ability of the methods to combine them. Moreover, a brand new distortion is introduced and tested, i.e. gray-scaling – described in section 6.2.4.

Table 6.5 contains the results of this experiment. The defended classification network is shared across different configurations. However, the distortions used by the detectors to identify malicious samples are changed. In particular, median, bit-depth and gray-scale refer to single-distortion configurations using, respectively, median smoothing, reduction of bit-depth resolution and gray-scaling. These configurations do not require any technique for merging the contribution of different distortions – such as the signature extraction function in eq. (6.19). Indeed, they provide de-

tails about the effectiveness of each single distortion against different adversarial examples and/or different detection frameworks. The 2-distortions configuration uses median smoothing and reduction of bit-depth resolution, i.e. the same configuration used in the experiment of section 6.4.1. The 3-distortions configuration, instead, adopts all three different distortions.

The results in table 6.5 suggest the same insights collected from table 6.2: the proposed detector outperforms FS on the more advanced attacks, while they compete with each other on the FGSM attack-set. Moreover, it is possible to note how FS struggles to efficiently combine multiple distortions. Indeed, it always achieves its best performance when using a single distortion configuration, with the 3-distortions configuration being almost always the worst. This indicates that using the most destructive distortion is not an efficient combination technique. Instead, the proposed method seems to benefit from the usage of several distortions. Indeed, besides producing competitive results with single distortion settings, it often achieves its best performance leveraging the effects of all three distortions. This implies that the combination technique adopted by the proposed method – i.e. eq. (6.19) – has the ability to generate strong, meaningful signatures, retaining the most relevant information for the detection of adversarial examples.

Focusing on the single-distortion configurations, it is possible to note how the effectiveness of each distortion varies across different datasets and/or attack techniques. In particular, median smoothing performs better on the GTSRB dataset, while bit-depth reduction is more effective on CIFAR10. The FGSM attack is more sensitive to the median smoothing distortion in both tasks, while resulting quite robust against reduction of the bit-depth resolution. The proposed gray-scaling dis-

Table 6.5: AUC scores of FS and the proposed detector for different choices of the distortions (%). The configuration with two distortions uses median smoothing and reduction of bit-depth resolution.

		C&W		DF		FGSM4	
	Distortions	FS	Prop.	FS	Prop.	FS	Prop.
CIFAR10	Median	80.69	93.68	80.73	93.72	**76.15**	69.01
	Bit-depth	**87.96**	94.41	**86.84**	94.71	52.64	56.83
	Gray-scale	75.16	88.85	74.62	88.32	65.09	65.76
	2–Distortions	82.34	96.58	82.21	97.00	74.90	67.52
	3–Distortions	75.63	**96.79**	75.37	**97.08**	73.88	**72.50**
GTSRB	Median	**97.86**	99.51	**97.63**	99.46	74.95	**75.63**
	Bit-depth	97.00	99.05	96.19	97.36	61.75	66.95
	Gray-scale	91.35	96.98	90.84	96.60	**77.19**	73.84
	2–Distortions	96.55	99.77	96.16	99.62	73.60	74.08
	3–Distortions	89.98	**99.81**	89.31	**99.72**	76.08	73.67

tortion shows good performance, surprisingly achieving the best results when used in the FS framework to detect FGSM samples on the GTSRB dataset. This proves that it represents a valid alternative, especially when used in multiple-distortions configurations. Finally, note how the proposed detector continues to outperform FS, even in single-distortion configurations. This suggests that it uses a more efficient detection scheme, regardless of the combination technique.

6.4.4 Black-box Results

The previous sections proved the superior performance of the proposed detector under white-box attack-settings. However, in order to achieve an exhaustive evaluation, it is important to include black-box experiments. Black-box attacks have peculiar characteristics, differentiating them from the white-box counterparts. For instance, they do not possess a strong relationship with the victim, as they are computed using substitute networks, hence using signals that differ from the victim's gradient. This characteristic might potentially impact the proposed detection framework, as it has a more intimate relationship with the victim model than FS.

Bearing in mind that only a limited amount of black-box malicious samples are successful – see table 6.1 –, it has been decided to use a different performance metric. In particular, AUC scores have been substituted with detection rates. As this metric requires a threshold value, similarly to [86], it has been decided to use the threshold value which rejects only 5% of the CIFAR10 legitimate samples (10% on the GTSRB dataset). The detection rates collected during the experiment are reported in table 6.6. Note how the proposed method results more robust and effective than FS to detect black-box adversarial examples. This is attributed to the usage of the first-order statistic as class representative vectors, providing a stable signature, regardless of the effectiveness of the attack. Finally, it is worth noting that, in this configuration, the proposed detector is also able to outperform FS on the FGSM attack.

6.4.5 Ablation Studies

The results proposed so far proved that the adversarial detector developed herein represents a valid technique for defending against adversarial examples, showing state-of-the-art detection capabilities. However, the role played by the solutions

Table 6.6: Detection rates of FS and the proposed detector in black-box settings (%).

	Method	Threshold	Legitim.	C&W	DF	FGSM4
CIFAR10	FS	1.2119	95.42	7.68	7.58	25.83
	Proposed	0.6894	94.59	**22.82**	**23.06**	**31.27**
GTSRB	FS	0.4097	90.13	69.27	70.15	68.51
	Proposed	0.9739	89.59	**78.28**	**77.59**	**72.95**

proposed in section 6.2 is yet to be assessed. This section provides an investigation on the effects of every one of the proposed solutions.

Figure 6.8 contains the ROC curves collected using the white-box C&W attack-set on the CIFAR10 dataset. The orange curve reports the performance of the prior art, Feature Squeezing (FS) [86]. It adopts the detection criteria derived in eq. (6.12), using the minimum operator to combine the effect of multiple distortions, i.e.

$$\min_j \left(S\big(f_{-1}(\psi_j(\boldsymbol{x})), f_{-1}(\boldsymbol{x})\big) \right) \underset{H_1}{\overset{H_0}{\gtrless}} \alpha \ . \tag{6.30}$$

The negative L_1–distance score metric in eq. (6.24) is used as similarity metric. To assess the benefits introduced by the normalized projection score metric in eq. (6.26), the same detection criteria in eq. (6.30) is evaluated, but using the normalized projection score as similarity metric (blue curve). It is possible to note the improvement introduced by the new metric, increasing the AUC score by more than 3 percentage points. The green curve has been generated to assess the benefits of the first order statistic as class representative vector. It is produced using the detection criteria in eq. (6.18), with the minimum operator as distortions combination technique, i.e.

$$\min_j \left(S\big(f_{-1}(\psi_j(\boldsymbol{x})), \mathbb{E}_{\boldsymbol{x}\in X_i}\big[f_{-1}(\psi_j(\boldsymbol{x}))\big]\big) \right) \underset{H_1}{\overset{H_0}{\gtrless}} \alpha \ . \tag{6.31}$$

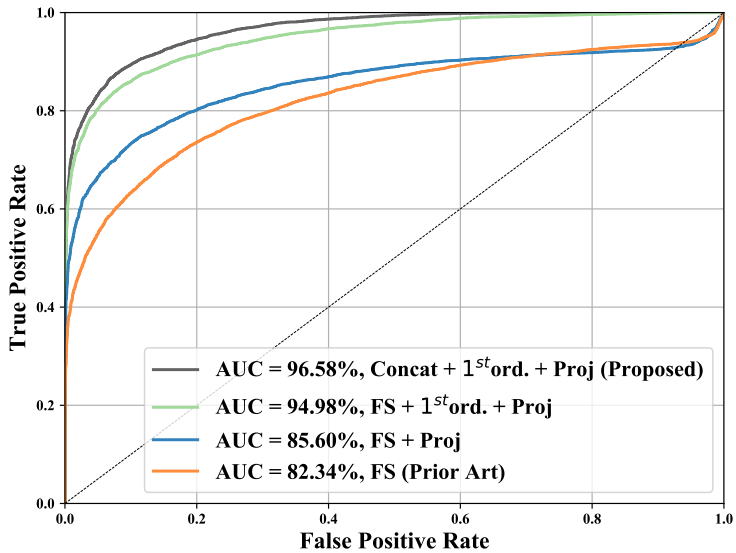


Figure 6.8: ROC curves showing the effects of the proposed solutions against the C&W attack on the CIFAR10 dataset. The orange curve is produced by FS – eq. (6.30) – and the gray one by the proposed detector (using all the solutions) – eq. (6.22). The blue curve shows the effects of the proposed normalized projection score metric in eq. (6.26). The green curve is produced using the detection criteria in eq. (6.31) and shows the improvement due to the first-order statistics as class representative vectors. Median filtering and reduction of the bit-depth resolution are used as distortions. Best viewed in color.

The normalized projection score is still used as similarity metric. Note the big improvement introduced by this solution, increasing the AUC score by nearly 10%. In addition, it is interesting to notice that this solution is responsible for the better convergence property discussed in section 6.4.1. This suggests that the usage of first-order statistics as class representative vectors brings stability to the detection framework, allowing it to achieve 100% detection rates with much lower false detection rates than the previous two configurations. Finally, the gray curve reports the performance of the detector using all the proposed solutions. It adopts the detection criteria in eq. (6.22) with the normalized projection score as similarity metric. It differs from eq. (6.31) as it uses the signature extraction process in eq. (6.19) – concatenation operation – to combine the effect of multiple distortions. Note how, by incorporating the combination technique into the score computation, it is possible to enhance even more the detection performance, increasing the AUC score by almost 2%. Ultimately, the proposed solutions address a more effective detection framework, achieving an improvement of nearly 15% over the prior art – i.e FS.

Chapter 7

Conclusions

This document proves the effectiveness of DL techniques to process radar data for solving various environment perception problems. Chapter 3 proposes innovative solutions that enable both, effective and efficient processing of radar data. It has been shown that it is possible to improve the performance of point-wise processing techniques by encoding radar-related information into high-dimensional point-descriptors. The proposed pre-processing module proved to be effective in extracting the radar-object-perception in various point-wise architectures. In addition, it has been shown that, in order to achieve an efficient processing, it is crucial to respect the intrinsic nature of radar. The usage of MS instead of FPS and the adoption of a non-hierarchical architecture proved to optimize the usage of spatial context information, taking into account the low-density of the data. By combining together all the proposed solutions, it has been designed RadarPCNN, a novel point-wise processing architecture for semantic segmentation of radar point clouds. It has been proved that this network can achieve state-of-the-art performance, using less capacity than previous DL approaches. Results of ablation studies have been reported, highlighting specific characteristics of RadarPCNN. In particular, it has been shown that SA layers do not require an ideal grouping process, as they can filter the content of local neighborhoods. Additionally, it has been proved that, in order to achieve superior performance, it is critical to ego-compensate the Doppler feature. It has also been shown that the model benefits from its usage as spatial coordinate. Finally, following a visual investigation on the predictions of RadarPCNN, it has been shown that the network confuses bushes for pedestrians. Yet, it has been proved that this effect is not due to the algorithm, but rather to their strong similarity from the radar perspective. In future, it would be interesting to evaluate the ability of the proposed pre-processing module to generalize to other point-wise processing architectures. Another interesting experiment consists in the comparison with grid-based approaches. Finally, since it has been found out that most of the network errors are present just for few frames, it would be possible to improve the network performance by including temporal information in the processing scheme.

Chapter 4 provides a survey on the ability of point-wise processing techniques to detect pedestrians. In particular, it has been proposed to study how these models use the radar features. The PointNet++ [64] and PointNet [65] architectures are analyzed. Shared FC and RF [20], instead, are used as benchmark. The experiments have proved that PointNet++ performs a different, more complex processing of the radar feature. It has been shown that Doppler represents the most informative

feature for the task of pedestrian detection. However, PointNet++ also leverages the local spatial structure to infer the output class. Although the RCS feature has proved to contain little information for the task at hand, NN-based techniques have demonstrated their ability to learn to include it in the decision process, extracting insightful point-properties. By analyzing the Doppler domain, it has been shown that GT pedestrians are characterized by both low and null Doppler values. Yet, PointNet++ is the only architecture that produce a Doppler distribution of pedestrian predictions that approximate the GT one. By succeeding in the detection of points with Doppler values close to 0, indeed, it demonstrated its ability to use all the radar features, while the other models depended almost uniquely on the velocity information. Finally, it has been shown how a perturbation of Doppler and XY-coordinates affect PointNet++ differently: it has been proved that the former impacts the precision score, while the latter influences the recall rate. A promising future work direction consists in the inclusion of other families of techniques in the survey. For instance, it would be fascinating to study how grid-based approaches use the radar features, compared to point-wise processing methods. Another interesting experiment consists in performing a deeper investigation on the prediction errors of PointNet++, especially for points with Doppler values close to 0, to provide further insights about the properties of missed detections.

Chapter 5 introduces a system that uses raw radar point clouds to perform the ultimate perception task: object detection. The proposed architecture has been designed to optimize the processing executed by the prior art model [92]. The results proved that it performs a more effective processing than the prior art, achieving better detection performance. Moreover, it resulted in less FLOPS, while using more parameters, thus proving the efficiency of the proposed solutions. By analyzing the box-regression performance, it turned out that the proposed method produces more accurate box-predictions. Indeed, the proposed regression-aided formulation results more effective than the template-based one adopted in the prior art. In addition, the spherical regression task, addressed for the orientation estimation, increased the accuracy of the angle predictions. Finally, the proposed architecture has been evaluated on a multi-class task, proving its ability to detect small objects like pedestrians. Despite the proposed system has outperformed the prior art, it is still possible to improve its performance by pursuing several work directions. One of them follows the observations gathered during the visual analysis in section 5.4.3. Indeed, since the proposed architecture uses point-data, it struggles to detect objects that are scarcely represented – providing none or few reflections. Consequently, it is possible to considerably improve the detection performance by producing point clouds that better represent the scenes. This can be achieved, e.g. by leveraging low-level radar data. Other promising directions entail an upgrade of the proposed architecture. For instance, the inclusion of an RNN module would provide the network with the means for leveraging temporal information, thus producing more consistent and confident predictions. Finally, both, detection performance and box-accuracy can be improved by advanced post-processing steps. Indeed, it is possible to perform a better filtering of the boxes by developing an enhanced NMS procedure and/or using a tracking algorithm. For this latter, the network could be equipped with a further output head that regresses the velocity of the objects, thus providing additional information for the tracking task.

Although ML/DL has proved to represent a valid solution for performing envi-

ronment perception tasks, it has some limitations. Chapter 6 introduces a system that helps defeating a specific weakness of ML techniques: adversarial examples. The proposed adversarial detector proved to be a low-cost, robust tool for detecting malicious samples. The system showed impressive detection capabilities, outperforming the state-of-the-art technique. It proved its ability to detect adversarial samples in different tasks/attack-settings. In particular, it showed its ability to identify samples computed with the most advanced attack-procedures. Though it proved slightly less robust against more aggressive attacks, it has been demonstrated that it is possible to annihilate them by combining multiple defensive strategies. In this context, it showed its remarkable compatibility with other defensive solutions, enhancing its detection performance. Moreover, it proved its ability to proficiently fuse the effects of multiple distortions as well as detect samples in black-box attack-settings. Ultimately, it was shown that all the solutions introduced in the design of the detector play a central role, bringing improvements to the overall system. The work presented in chapter 6 opens various research directions. Indeed, it would be interesting to perform a deeper investigation into the compatibility property of the proposed detector, experimenting with other defensive strategies. Another intriguing experiment would consist in studying the robustness of the classifier-detector system when attacked in white-box settings (giving the attacker access to the defense). Indeed, Sharma et al. proved that an adversarial detector can be bypassed in this settings [82]. However, other solutions can be put in place to avoid this effect, such as the introduction of randomness in the system. An example would be, for every input query, to select the distortions to use in the detection framework from a pool at random. Finally, it would be also interesting to investigate the usage of data-driven distortions, even though they might result more exposed to an attack due to the back-propagation process.

Bibliography

[1] Harold W. Kuhn. "The Hungarian Method for the Assignment Problem". In: *Naval Research Logistics Quarterly* 2.1–2 (1955), pp. 83–97. DOI: 10.1002/nav.3800020109.

[2] D. R. Cox. "The Regression Analysis of Binary Sequences". In: *Journal of the Royal Statistical Society. Series B (Methodological)* 20.2 (1958), pp. 215–242. ISSN: 00359246. URL: http://www.jstor.org/stable/2983890.

[3] Peter J. Huber. "Robust Estimation of a Location Parameter". In: *The Annals of Mathematical Statistics* 35.1 (1964), pp. 73 –101. DOI: 10.1214/aoms/1177703732. URL: https://doi.org/10.1214/aoms/1177703732.

[4] Thomas M. Cover and Peter E. Hart. "Nearest neighbor pattern classification". In: *IEEE Trans. Inf. Theory* 13.1 (1967), pp. 21–27. DOI: 10.1109/TIT.1967.1053964. URL: https://doi.org/10.1109/TIT.1967.1053964.

[5] Keinosuke Fukunaga and Larry D. Hostetler. "The estimation of the gradient of a density function, with applications in pattern recognition". In: *IEEE Trans. Information Theory* 21.1 (1975), pp. 32–40. DOI: 10.1109/TIT.1975.1055330.

[6] L. Breiman et al. *Classification and Regression Trees*. Monterey, CA: Wadsworth and Brooks, 1984.

[7] Teofilo F. Gonzalez. "Clustering to Minimize the Maximum Intercluster Distance". In: *Theor. Comput. Sci.* 38 (1985), pp. 293–306.

[8] J. R. Quinlan. "Induction of Decision Trees". In: *MACH. LEARN* 1 (1986), pp. 81–106.

[9] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning Representations by Back-propagating Errors". In: *Nature* 323.6088 (1986), pp. 533–536. DOI: 10.1038/323533a0. URL: http://www.nature.com/articles/323533a0.

[10] R. Schmidt. "Multiple emitter location and signal parameter estimation". In: *IEEE Transactions on Antennas and Propagation* 34.3 (1986), pp. 276–280. DOI: 10.1109/TAP.1986.1143830.

[11] J. Y. Chen and I. S. Reed. "A Detection Algorithm for Optical Targets in Clutter". In: *IEEE Transactions on Aerospace and Electronic Systems* AES-23.1 (Jan. 1987), pp. 46–59. ISSN: 1557-9603. DOI: 10.1109/TAES.1987.313335.

[12] A. Elfes. “Using Occupancy Grids for Mobile Robot Perception and Navigation”. In: *Computer* 22.6 (June 1989), 46–57. ISSN: 0018-9162. DOI: 10.1109/2.30720. URL: https://doi.org/10.1109/2.30720.

[13] Yann Lecun. “Generalization and network design strategies”. English (US). In: *Connectionism in perspective*. Ed. by R. Pfeifer et al. Elsevier, 1989.

[14] R. Roy and T. Kailath. “ESPRIT-estimation of signal parameters via rotational invariance techniques”. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 37.7 (1989), pp. 984–995. DOI: 10.1109/29.32276.

[15] Corinna Cortes and Vladimir Vapnik. “Support-Vector Networks”. In: *Mach. Learn.* 20.3 (Sept. 1995), 273–297. ISSN: 0885-6125. DOI: 10.1023/A:1022627411411. URL: https://doi.org/10.1023/A:1022627411411.

[16] Martin Ester et al. “A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise”. In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), Portland, Oregon, USA*. Ed. by Evangelos Simoudis, Jiawei Han, and Usama M. Fayyad. AAAI Press, 1996, pp. 226–231. URL: http://www.aaai.org/Library/KDD/1996/kdd96-037.php.

[17] L. H. Eriksson and B. . As. “Automotive radar for adaptive cruise control and collision warning/avoidance”. In: *Radar 97 (Conf. Publ. No. 449)*. 1997, pp. 16–20. DOI: 10.1049/cp:19971623.

[18] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory”. In: *Neural Computation* 9.8 (1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

[19] Thomas M. Mitchell. *Machine Learning*. 1st ed. USA: McGraw-Hill, Inc., 1997. ISBN: 0070428077.

[20] Leo Breiman. “Random Forests”. In: *Machine Learning* 45.1 (2001), pp. 5–32. DOI: 10.1023/A:1010933404324.

[21] Constantine A. Balanis. *Antenna Theory: Analysis and Design*. 3rd ed. USA: Wiley-Interscience, 2005, pp. 34, 72. ISBN: 0471714623.

[22] Giancarlo Alessandretti, Alberto Broggi, and Pietro Cerri. “Vehicle and Guard Rail Detection Using Radar and Vision Data Fusion”. In: *IEEE Trans. Intell. Transp. Syst.* 8.1 (2007), pp. 95–105. DOI: 10.1109/TITS.2006.888597. URL: https://doi.org/10.1109/TITS.2006.888597.

[23] Sean Gillies et al. *Shapely: manipulation and analysis of geometric objects*. toblerity.org, 2007–. URL: https://github.com/Toblerity/Shapely.

[24] Henning Ritter and Hermann Rohling. “Pedestrian detection based on automotive radar”. In: Nov. 2007, pp. 1 –4. DOI: 10.1049/cp:20070656.

[25] Martin Cadík. “Perceptual Evaluation of Color-to-Grayscale Image Conversions”. In: *Computer Graphics Forum* 27 (2008).

[26] Alex Krizhevsky. *Learning multiple layers of features from tiny images*. Tech. rep. 2009.

[27] E.H. Choi and National Highway Traffic Safety Administration. *Crash Factors in Intersection-Related Crashes: An On-Scene Perspective*. Nhtsa Technical Report Dot Hs 811 366. Createspace Independent Pub, 2010. ISBN: 9781493507139. URL: https://books.google.de/books?id=Pxc2nQAACAAJ.

[28] Erik Coelingh, Andreas Eidehall, and Mattias Bengtsson. "Collision Warning with Full Auto Brake and Pedestrian Detection - a practical example of Automatic Emergency Braking". In: *13th International IEEE Conference on Intelligent Transportation Systems*. 2010, pp. 155–160. DOI: 10.1109/ITSC.2010.5625077.

[29] Johannes Stallkamp et al. "The German Traffic Sign Recognition Benchmark: A multi-class classification competition". In: *The 2011 International Joint Conference on Neural Networks, IJCNN 2011, San Jose, California, USA, July 31 - August 5, 2011*. 2011, pp. 1453–1460. DOI: 10.1109/IJCNN.2011.6033395. URL: https://doi.org/10.1109/IJCNN.2011.6033395.

[30] M. Adams et al. *Robotic Navigation and Mapping with Radar*. Artech House, 2012.

[31] A. Bartsch, F. Fitzek, and R. Rasshofer. "Pedestrian recognition using automotive radar sensors". In: *Advances in Radio Science* 10 (Sept. 2012), pp. 45–55. DOI: 10.5194/ars-10-45-2012.

[32] Steffen Heuel and Hermann Rohling. "Pedestrian Classification in Automotive Radar Systems". In: (May 2012), pp. 39–44. DOI: 10.1109/IRS.2012.6233285.

[33] Michael Brown, Michael J. Pelosi, and Henry Dirska. "Dynamic-Radius Species-Conserving Genetic Algorithm for the Financial Forecasting of Dow Jones Index Stocks". In: vol. 7988. July 2013. DOI: 10.1007/978-3-642-39712-7_3.

[34] Cyril Chauvel et al. "Automatic Emergency Braking for Pedestrians: Effective Target Population and Expected Safety Benefits". In: 2013.

[35] R. Dubé et al. "Detection of parked vehicles from a radar based occupancy grid". In: *2014 IEEE Intelligent Vehicles Symposium Proceedings*. 2014, pp. 1415–1420. DOI: 10.1109/IVS.2014.6856568.

[36] Yan Gao et al. "An Improved Music Algorithm for DOA Estimation of Coherent Signals". In: *Sensors and Transducers* 175 (July 2014), pp. 75–82.

[37] Christian Szegedy et al. "Intriguing properties of neural networks". In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*. 2014. URL: http://arxiv.org/abs/1312.6199.

[38] Chenyi Chen et al. "DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving". In: *CoRR* abs/1505.00256 (2015). arXiv: 1505.00256. URL: http://arxiv.org/abs/1505.00256.

[39] Brian N. Fildes et al. "Effectiveness of low speed autonomous emergency braking in real-world rear-end crashes." In: *Accident; analysis and prevention* 81 (2015), pp. 24–9.

[40] Ross B. Girshick. “Fast R-CNN”. In: *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*. IEEE Computer Society, 2015, pp. 1440–1448. DOI: 10.1109/ICCV.2015.169. URL: https://doi.org/10.1109/ICCV.2015.169.

[41] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. “Explaining and Harnessing Adversarial Examples”. In: *International Conference on Learning Representations*. 2015. URL: http://arxiv.org/abs/1412.6572.

[42] F. Maxwell Harper and Joseph A. Konstan. “The MovieLens Datasets: History and Context”. In: *ACM Trans. Interact. Intell. Syst.* 5.4 (Dec. 2015). ISSN: 2160-6455. DOI: 10.1145/2827872. URL: https://doi.org/10.1145/2827872.

[43] Ruitong Huang et al. “Learning with a Strong Adversary”. In: *CoRR* abs/1511.03034 (2015). arXiv: 1511.03034. URL: http://arxiv.org/abs/1511.03034.

[44] Anh Mai Nguyen, Jason Yosinski, and Jeff Clune. “Deep neural networks are easily fooled: High confidence predictions for unrecognizable images”. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. 2015, pp. 427–436. DOI: 10.1109/CVPR.2015.7298640. URL: https://doi.org/10.1109/CVPR.2015.7298640.

[45] Olga Russakovsky et al. “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision (IJCV)* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[46] Ziqiang Tong, Ralf Renter, and Masahiko Fujimoto. “Fast chirp FMCW radar in automotive applications”. In: *IET International Radar Conference 2015*. 2015, pp. 1–4. DOI: 10.1049/cp.2015.1362.

[47] International Communication Union. *Recommendation ITU-R BT.709-6: Parameter values for the HDTV standards for production and international programme exchange*. en. Tech. rep. BT.709-6. International Organization for Standardization, 2015. URL: https://www.itu.int/rec/R-REC-BT.709.

[48] K. Werber et al. “Automotive radar gridmap representations”. In: *2015 IEEE MTT-S International Conference on Microwaves for Intelligent Mobility (ICMIM)*. 2015, pp. 1–4.

[49] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[50] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. “DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. 2016, pp. 2574–2582. DOI: 10.1109/CVPR.2016.282. URL: https://doi.org/10.1109/CVPR.2016.282.

[51] Abhay Prakash, Chris Brockett, and Puneet Agrawal. “Emulating Human Conversations using Convolutional Neural Network-based IR”. In: *ArXiv* abs/1606.07056 (2016).

[52] Andras Rozsa, Ethan M. Rudd, and Terrance E. Boult. "Adversarial Diversity and Hard Positive Generation". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2016, Las Vegas, NV, USA, June 26 - July 1, 2016*. 2016, pp. 410–417. DOI: 10.1109/CVPRW.2016.58. URL: https://doi.org/10.1109/CVPRW.2016.58.

[53] Nicholas Carlini and David Wagner. "Adversarial Examples Are Not Easily Detected: Bypassing Ten Detection Methods". In: *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*. AISec '17. Dallas, Texas, USA: Association for Computing Machinery, 2017, 3–14. ISBN: 9781450352024. DOI: 10.1145/3128572.3140444. URL: https://doi.org/10.1145/3128572.3140444.

[54] Nicholas Carlini and David A. Wagner. "Towards Evaluating the Robustness of Neural Networks". In: *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*. 2017, pp. 39–57. DOI: 10.1109/SP.2017.49. URL: https://doi.org/10.1109/SP.2017.49.

[55] Pin-Yu Chen et al. "ZOO: Zeroth Order Optimization Based Black-box Attacks to Deep Neural Networks without Training Substitute Models". In: *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, AISec@CCS 2017, Dallas, TX, USA, November 3, 2017*. 2017, pp. 15–26. DOI: 10.1145/3128572.3140448. URL: https://doi.org/10.1145/3128572.3140448.

[56] Martin Engelcke et al. "Vote3Deep: Fast object detection in 3D point clouds using efficient convolutional neural networks". In: *2017 IEEE International Conference on Robotics and Automation, ICRA 2017, Singapore, Singapore, May 29 - June 3, 2017*. IEEE, 2017, pp. 1355–1361. DOI: 10.1109/ICRA.2017.7989161. URL: https://doi.org/10.1109/ICRA.2017.7989161.

[57] Andre Esteva et al. "Dermatologist-level classification of skin cancer with deep neural networks". In: *Nat.* 542.7639 (2017), pp. 115–118. DOI: 10.1038/nature21056. URL: https://doi.org/10.1038/nature21056.

[58] Kathrin Grosse et al. "On the (Statistical) Detection of Adversarial Examples". In: *CoRR* abs/17 (2017). URL: https://publications.cispa.saarland/1142/.

[59] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. "Adversarial examples in the physical world". In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. 2017. URL: https://openreview.net/forum?id=HJGU3Rodl.

[60] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. "Adversarial Machine Learning at Scale". In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. 2017. URL: https://openreview.net/forum?id=BJm4T4Kgx.

[61] J. Lombacher et al. "Object classification in radar using ensemble methods". In: *2017 IEEE MTT-S International Conference on Microwaves for Intelligent Mobility (ICMIM)*. 2017, pp. 87–90.

[62] J. Lombacher et al. "Semantic radar grids". In: *2017 IEEE Intelligent Vehicles Symposium (IV)*. 2017, pp. 1170–1175.

[63] Dongyu Meng and Hao Chen. "MagNet: A Two-Pronged Defense against Adversarial Examples". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*. 2017, pp. 135–147. DOI: 10.1145/3133956.3134057. URL: https://doi.org/10.1145/3133956.3134057.

[64] Charles Ruizhongtai Qi et al. "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space". In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*. Ed. by Isabelle Guyon et al. 2017, pp. 5099–5108. URL: http://papers.nips.cc/paper/7095-pointnet-deep-hierarchical-feature-learning-on-point-sets-in-a-metric-space.

[65] Charles Ruizhongtai Qi et al. "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*. IEEE Computer Society, 2017, pp. 77–85. DOI: 10.1109/CVPR.2017.16. URL: https://doi.org/10.1109/CVPR.2017.16.

[66] Evan Shelhamer, Jonathan Long, and Trevor Darrell. "Fully Convolutional Networks for Semantic Segmentation". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 39.4 (2017), pp. 640–651. DOI: 10.1109/TPAMI.2016.2572683. URL: https://doi.org/10.1109/TPAMI.2016.2572683.

[67] Christian Wöhler et al. "Comparison of random forest and long short-term memory network performances in classification tasks using radar". In: *Sensor Data Fusion: Trends, Solutions, Applications, SDF 2017, Bonn, Germany, October 10-12, 2017*. IEEE, 2017, pp. 1–6. DOI: 10.1109/SDF.2017.8126350. URL: https://doi.org/10.1109/SDF.2017.8126350.

[68] Manzil Zaheer et al. "Deep Sets". In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*. Ed. by Isabelle Guyon et al. 2017, pp. 3391–3401. URL: http://papers.nips.cc/paper/6931-deep-sets.

[69] Shang-Tse Chen et al. "ShapeShifter: Robust Physical Adversarial Attack on Faster R-CNN Object Detector". In: *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2018, Dublin, Ireland, September 10-14, 2018, Proceedings, Part I*. Ed. by Michele Berlingerio et al. Vol. 11051. Lecture Notes in Computer Science. Springer, 2018, pp. 52–68. DOI: 10.1007/978-3-030-10925-7_4. URL: https://doi.org/10.1007/978-3-030-10925-7_4.

[70] Verena Distler, Carine Lallemand, and Thierry Bellet. "Acceptability and Acceptance of Autonomous Mobility on Demand: The Impact of an Immersive Experience". In: *CHI '18*. 2018.

[71] Yinpeng Dong et al. "Boosting Adversarial Attacks With Momentum". In: *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*. 2018, pp. 9185–9193. DOI: 10.1109/CVPR.2018.00957. URL: http://openaccess.thecvf.com/

content_cvpr_2018/html/Dong_Boosting_Adversarial_Attacks_CVPR_2018_paper.html.

[72] Yifan Feng et al. "GVCNN: Group-View Convolutional Neural Networks for 3D Shape Recognition". In: *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*. IEEE Computer Society, 2018, pp. 264–272. DOI: 10.1109/CVPR.2018.00035. URL: http://openaccess.thecvf.com/content_cvpr_2018/html/Feng_GVCNN_Group-View_Convolutional_CVPR_2018_paper.html.

[73] Neder Karmous, Mohamed Ould El Hassan, and Fethi Choubeni. "An Improved Esprit Algorithm for DOA Estimation of Coherent Signals". In: *International Conference on Smart Communications and Networking, SmartNets 2018, Yasmine Hammamet, Tunisia, November 16-17, 2018*. IEEE, 2018, pp. 1–4. DOI: 10.1109/SMARTNETS.2018.8707432. URL: https://doi.org/10.1109/SMARTNETS.2018.8707432.

[74] Yangyan Li et al. "PointCNN: Convolution On X-Transformed Points". In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*. Ed. by Samy Bengio et al. 2018, pp. 828–838. URL: http://papers.nips.cc/paper/7362-pointcnn-convolution-on-x-transformed-points.

[75] Fangzhou Liao et al. "Defense Against Adversarial Attacks Using High-Level Representation Guided Denoiser". In: *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*. 2018, pp. 1778–1787. DOI: 10.1109/CVPR.2018.00191. URL: http://openaccess.thecvf.com/content_cvpr_2018/html/Liao_Defense_Against_Adversarial_CVPR_2018_paper.html.

[76] Rosanne Liu et al. "An intriguing failing of convolutional neural networks and the CoordConv solution". In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. Ed. by Samy Bengio et al. 2018, pp. 9628–9639. URL: https://proceedings.neurips.cc/paper/2018/hash/60106888f8977b71e1f15db7bc9a88d1-Abstract.html.

[77] R. Prophet et al. "Pedestrian Classification with a 79 GHz Automotive Radar Sensor". In: *2018 19th International Radar Symposium (IRS)*. 2018, pp. 1–6. DOI: 10.23919/IRS.2018.8448161.

[78] Robert Prophet et al. "Pedestrian Classification for 79 GHz Automotive Radar Systems". In: *2018 IEEE Intelligent Vehicles Symposium, IV 2018, Changshu, Suzhou, China, June 26-30, 2018*. IEEE, 2018, pp. 1265–1270. DOI: 10.1109/IVS.2018.8500554. URL: https://doi.org/10.1109/IVS.2018.8500554.

[79] Charles R. Qi et al. "Frustum PointNets for 3D Object Detection From RGB-D Data". In: *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*. Computer Vision Foundation / IEEE Computer Society, 2018, pp. 918–927. DOI: 10.1109/CVPR.2018.00102. URL: http://openaccess.thecvf.com/

content_cvpr_2018/html/Qi_Frustum_PointNets_for_CVPR_2018_paper.html.

[80] Joseph Redmon and Ali Farhadi. "YOLOv3: An Incremental Improvement". In: *CoRR* abs/1804.02767 (2018). arXiv: 1804.02767. URL: http://arxiv.org/abs/1804.02767.

[81] Ole Schumann et al. "Semantic Segmentation on Radar Point Clouds". In: *21st International Conference on Information Fusion, FUSION 2018, Cambridge, UK, July 10-13, 2018*. IEEE, 2018, pp. 2179–2186. DOI: 10.23919/ICIF.2018.8455344. URL: https://doi.org/10.23919/ICIF.2018.8455344.

[82] Yash Sharma and Pin-Yu Chen. "Bypassing Feature Squeezing by Increasing Adversary Strength". In: *CoRR* abs/1803.09868 (2018). arXiv: 1803.09868. URL: http://arxiv.org/abs/1803.09868.

[83] Chawin Sitawarin et al. "DARTS: Deceiving Autonomous Cars with Toxic Signs". In: *CoRR* abs/1802.06430 (2018). arXiv: 1802.06430. URL: http://arxiv.org/abs/1802.06430.

[84] Florian Tramèr et al. "Ensemble Adversarial Training: Attacks and Defenses". In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. 2018. URL: https://openreview.net/forum?id=rkZvSe-RZ.

[85] Lachlan Tychsen-Smith and Lars Petersson. "Improving Object Localization With Fitness NMS and Bounded IoU Loss". In: *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*. Computer Vision Foundation / IEEE Computer Society, 2018, pp. 6877–6885. DOI: 10.1109/CVPR.2018.00719. URL: http://openaccess.thecvf.com/content_cvpr_2018/html/Tychsen-Smith_Improving_Object_Localization_CVPR_2018_paper.html.

[86] Weilin Xu, David Evans, and Yanjun Qi. "Feature Squeezing: Detecting Adversarial Examples in Deep Neural Networks". In: *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*. 2018. URL: http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2018/02/ndss2018_03A-4_Xu_paper.pdf.

[87] Bo Yang et al. "Attentional Aggregation of Deep Feature Sets for Multi-view 3D Reconstruction". In: *CoRR* abs/1808.00758 (2018). arXiv: 1808.00758. URL: http://arxiv.org/abs/1808.00758.

[88] Yin Zhou and Oncel Tuzel. "VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection". In: *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR*. 2018, pp. 4490–4499. DOI: 10.1109/CVPR.2018.00472.

[89] Daniel Brodeski, Igal Bilik, and Raja Giryes. "Deep Radar Detector". In: *CoRR* abs/1906.12187 (2019). arXiv: 1906.12187. URL: http://arxiv.org/abs/1906.12187.

[90] Nicholas Carlini et al. “On Evaluating Adversarial Robustness”. In: *CoRR* abs/1902.06705 (2019). arXiv: 1902.06705. URL: http://arxiv.org/abs/1902.06705.

[91] Alessandro Cennamo et al. “Method and System for Determining whether Input Data to be Classified is Manipulated Input Data”. European pat. 19182110.7. Aptiv Technologies Ltd. June 24, 2019. URL: https://data.epo.org/publication-server/document?iDocId=6430743&iFormat=0.

[92] Andreas Danzer et al. “2D Car Detection in Radar Data with PointNets”. In: *2019 IEEE Intelligent Transportation Systems Conference, ITSC 2019, Auckland, New Zealand, October 27-30, 2019*. IEEE, 2019, pp. 61–66. DOI: 10.1109/ITSC.2019.8917000. URL: https://doi.org/10.1109/ITSC.2019.8917000.

[93] Z. Feng et al. “Point Cloud Segmentation with a High-Resolution Automotive Radar”. In: *AmE 2019 - Automotive meets Electronics; 10th GMM-Symposium*. 2019, pp. 1–5.

[94] Justin M. Johnson and Taghi M. Khoshgoftaar. “Survey on deep learning with class imbalance”. In: *J. Big Data* 6 (2019), p. 27. DOI: 10.1186/s40537-019-0192-5. URL: https://doi.org/10.1186/s40537-019-0192-5.

[95] Ajay Jose, Harish Thodupunoori, and Binoy Nair. “A Novel Traffic Sign Recognition System Combining Viola–Jones Framework and Deep Learning: Methods and Protocols”. In: Jan. 2019, pp. 507–517. ISBN: 978-1-4939-8936-2. DOI: 10.1007/978-981-13-3600-3_48.

[96] Shuai Liao, Efstratios Gavves, and Cees G. M. Snoek. “Spherical Regression: Learning Viewpoints, Surface Normals and 3D Rotations on N-Spheres”. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 2019, pp. 9759–9767. DOI: 10.1109/CVPR.2019.00999. URL: http://openaccess.thecvf.com/content_CVPR_2019/html/Liao_Spherical_Regression_Learning_Viewpoints_Surface_Normals_and_3D_Rotations_on_CVPR_2019_paper.html.

[97] Bence Major et al. “Vehicle Detection With Automotive Radar Using Deep Learning on Range-Azimuth-Doppler Tensors”. In: *2019 IEEE/CVF International Conference on Computer Vision Workshops, ICCV Workshops 2019, Seoul, Korea (South), October 27-28, 2019*. IEEE, 2019, pp. 924–932. DOI: 10.1109/ICCVW.2019.00121. URL: https://doi.org/10.1109/ICCVW.2019.00121.

[98] Cristiano Premebida, Gledson Melotti, and Alireza Asvadi. “RGB-D Object Classification for Autonomous Driving Perception”. In: Oct. 2019, pp. 377–395. ISBN: 978-3-030-28602-6. DOI: 10.1007/978-3-030-28603-3_17.

[99] Robert Prophet et al. “Semantic Segmentation on Automotive Radar Maps”. In: *2019 IEEE Intelligent Vehicles Symposium, IV 2019, Paris, France, June 9-12, 2019*. IEEE, 2019, pp. 756–763. DOI: 10.1109/IVS.2019.8813808. URL: https://doi.org/10.1109/IVS.2019.8813808.

[100] Farhana Sultana, Abu Sufian, and Paramartha Dutta. "Advancements in Image Classification using Convolutional Neural Network". In: *CoRR* abs/1905.03288 (2019). arXiv: 1905.03288. URL: http://arxiv.org/abs/1905.03288.

[101] Jiaqi Wang et al. "Region Proposal by Guided Anchoring". In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 2019, pp. 2965–2974. DOI: 10.1109/CVPR.2019.00308. URL: http://openaccess.thecvf.com/content_CVPR_2019/html/Wang_Region_Proposal_by_Guided_Anchoring_CVPR_2019_paper.html.

[102] Xiaoyong Yuan et al. "Adversarial Examples: Attacks and Defenses for Deep Learning". In: *IEEE Transactions on Neural Networks and Learning Systems* 30.9 (2019), pp. 2805–2824. DOI: 10.1109/TNNLS.2018.2886017.

[103] Shizhe Zang et al. "The Impact of Adverse Weather Conditions on Autonomous Vehicles: Examining how rain, snow, fog, and hail affect the performance of a self-driving car". In: *IEEE Vehicular Technology Magazine* PP (Mar. 2019), pp. 1–1. DOI: 10.1109/MVT.2019.2892497.

[104] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. "Objects as Points". In: *CoRR* abs/1904.07850 (2019). arXiv: 1904.07850. URL: http://arxiv.org/abs/1904.07850.

[105] Holger Caesar et al. "nuScenes: A Multimodal Dataset for Autonomous Driving". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13-19, 2020*. IEEE, 2020, pp. 11618–11628. DOI: 10.1109/CVPR42600.2020.01164. URL: https://doi.org/10.1109/CVPR42600.2020.01164.

[106] Alessandro Cennamo, Ido Freeman, and Anton Kummert. "A Statistical Defense Approach for Detecting Adversarial Examples". In: *Proceedings of the 2020 International Conference on Pattern Recognition and Intelligent Systems*. New York, NY, USA: Association for Computing Machinery, 2020. ISBN: 9781450387699. URL: https://doi.org/10.1145/3415048.3416103.

[107] Alessandro Cennamo and Florian Kaestner. "Method and device for detecting objects". European pat. 20202567.2. Aptiv Technologies Ltd. Oct. 19, 2020.

[108] Alessandro Cennamo, Florian Kästner, and Anton Kummert. "Leveraging Radar Features to Improve Point Clouds Segmentation with Neural Networks". In: *Proceedings of the 21st EANN (Engineering Applications of Neural Networks) 2020 Conference - Proceedings of the EANN 2020, Halkidiki, Greece, June 5-7, 2020*. Ed. by Lazaros Iliadis et al. Vol. 2. Proceedings of the International Neural Networks Society. Springer, 2020, pp. 119–131. DOI: 10.1007/978-3-030-48791-1_8. URL: https://doi.org/10.1007/978-3-030-48791-1_8.

[109] Maria Dreher et al. "Radar-based 2D Car Detection Using Deep Neural Networks". In: *23rd IEEE International Conference on Intelligent Transportation Systems, ITSC 2020, Rhodes, Greece, September 20-23, 2020*. IEEE, 2020, pp. 1–8. DOI: 10.1109/ITSC45102.2020.9294546. URL: https://doi.org/10.1109/ITSC45102.2020.9294546.

[110] Tsung-Yi Lin et al. “Focal Loss for Dense Object Detection”. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 42.2 (2020), pp. 318–327. DOI: 10.1109/TPAMI.2018.2858826. URL: https://doi.org/10.1109/TPAMI.2018.2858826.

[111] Mirko Meuter et al. “Methods and System for Detection of Objects in the vicinity of a Vehicle”. European pat. 20187674.5. Aptiv Technologies Ltd. Sept. 16, 2020.

[112] Emmanuel Pintelas and et al. “Explainable Machine Learning Framework for Image Classification Problems: Case Study on Glioma Cancer Prediction”. In: *Journal of Imaging* 6 (May 2020). DOI: 10.3390/jimaging6060037.

[113] Azad Raheem and Ali Mahmood. “Accurate DOA Estimation Using Modified ESPRIT Algorithm”. In: *International Journal of Intelligent Engineering and Systems* 13 (June 2020), pp. 358–366. DOI: 10.22266/ijies2020.0831.31.

[114] Ole Schumann et al. “Scene Understanding With Automotive Radar”. In: *IEEE Trans. Intell. Veh.* 5.2 (2020), pp. 188–203. DOI: 10.1109/TIV.2019.2955853. URL: https://doi.org/10.1109/TIV.2019.2955853.

[115] Jun Yu et al. “Spatial Pyramid-Enhanced NetVLAD With Weighted Triplet Loss for Place Recognition”. In: *IEEE Trans. Neural Networks Learn. Syst.* 31.2 (2020), pp. 661–674. DOI: 10.1109/TNNLS.2019.2908982. URL: https://doi.org/10.1109/TNNLS.2019.2908982.

[116] Wanwan Zheng and Mingzhe Jin. “The Effects of Class Imbalance and Training Data Size on Classifier Learning: An Empirical Study”. In: *SN Comput. Sci.* 1.2 (2020), p. 71. DOI: 10.1007/s42979-020-0074-0. URL: https://doi.org/10.1007/s42979-020-0074-0.

[117] Xunfei Zhou et al. “Development of a Camera-Based Driver State Monitoring System for Cost-Effective Embedded Solution”. In: *SAE Technical Paper*. SAE International, Apr. 2020. DOI: 10.4271/2020-01-1210. URL: https://doi.org/10.4271/2020-01-1210.

[118] Marco Braun et al. “Semantic Segmentation of Radar Detections using Convolutions on Point Clouds”. In: *Journal of Physics: Conference Series* 1924 (May 2021), p. 012003. DOI: 10.1088/1742-6596/1924/1/012003.

[119] Alessandro Cennamo and Florian Kaestner. “Computer Implemented Method, Computer System and Computer Readable Medium for Object Detection in a Vehicle”. European pat. 21195945.7. Aptiv Technologies Ltd. Sept. 10, 2021.

[120] Alessandro Cennamo, Florian Kaestner, and Anton Kummert. “A Neural Network Based System for Efficient Semantic Segmentation of Radar Point Clouds”. In: *Neural Processing Letters* (2021). DOI: 10.1007/s11063-021-10544-4. URL: https://doi.org/10.1007%2Fs11063-021-10544-4.

[121] Alessandro Cennamo, Florian Kaestner, and Anton Kummert. “Towards Pedestrian Detection in Radar Point Clouds with Pointnets”. In: *2021 International Conference on Machine Vision and Applications*. New York, NY, USA: Association for Computing Machinery, 2021, 1–7. ISBN: 9781450389556. URL: https://doi.org/10.1145/3459066.3459067.

[122] Mirko Meuter et al. “Device and method for determining objects around a vehicle”. European pat. 21190164.0. Aptiv Technologies Ltd. Aug. 6, 2021.

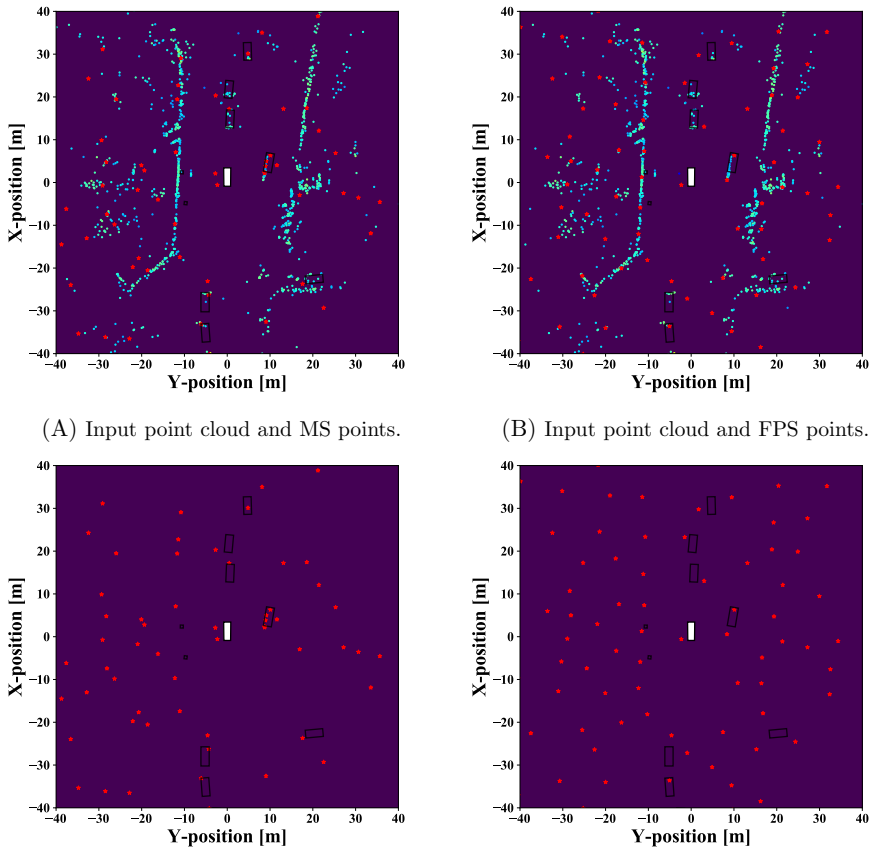
[123] Tianwei Yin, Xingyi Zhou, and Philipp Krähenbühl. "Center-Based 3D Object Detection and Tracking". In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*. Computer Vision Foundation / IEEE, 2021, pp. 11784–11793. URL: https://openaccess.thecvf.com/content/CVPR2021/html/Yin_Center-Based_3D_Object_Detection_and_Tracking_CVPR_2021_paper.html.

[124] Jun Yu et al. "SPRNet: Single-Pixel Reconstruction for One-Stage Instance Segmentation". In: *IEEE Transactions on Cybernetics* 51.4 (2021), pp. 1731–1742. DOI: 10.1109/TCYB.2020.2969046.

Appendix A

A.1 Point-Sampling with MS and FPS

This section contains support results for the comparison between MS and FPS sampling strategy in section 6.4.5. Radar reflections are represented with points, colored based on their RCS value. Red stars show the sampling algorithm output. Objects present in the scene are signaled by black boxes. Best viewed in color.



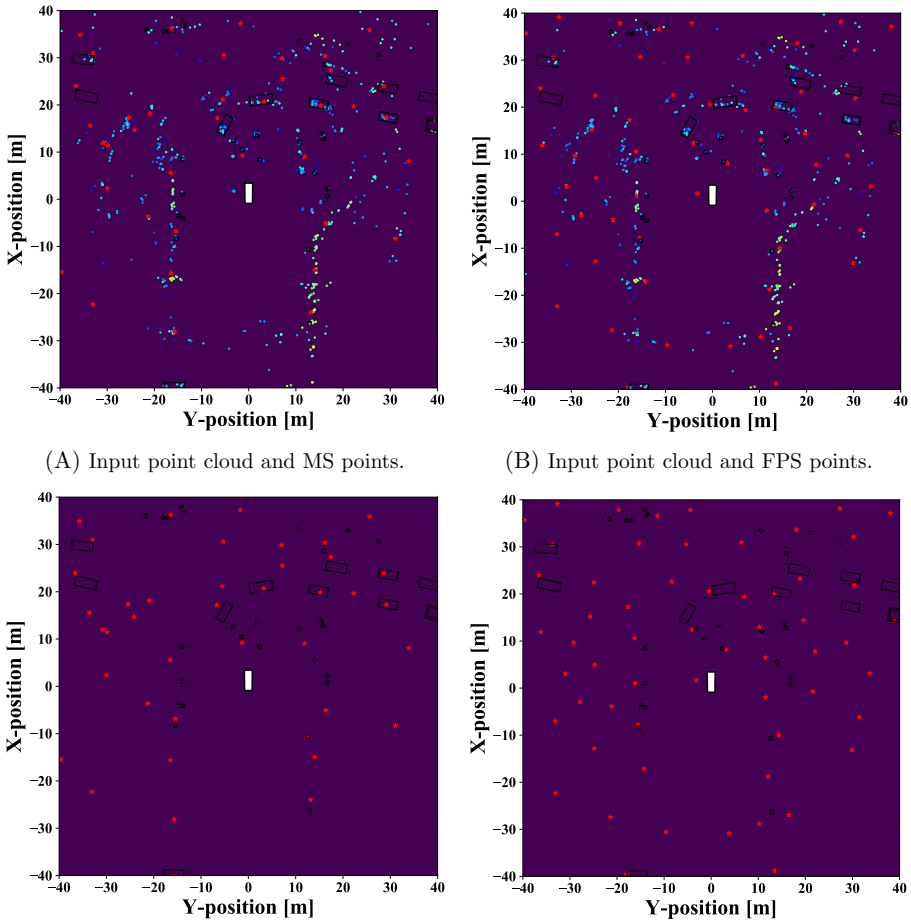
(A) Input point cloud and MS points.

(B) Input point cloud and FPS points.

(C) MS representative points only.

(D) FPS representative points only.

Figure A.1: Both methods sample most of the objects of interest. Yet, MS provides a better description of the space, while FPS results more uniform.



(A) Input point cloud and MS points.

(B) Input point cloud and FPS points.

(C) MS representative points only.

(D) FPS representative points only.

Figure A.2: MS produces a very clean scene with the sampled points, while FPS provides a corrupted representation. Moreover, MS respects the scene, sampling points for all vehicles. FPS misses some of them. Pedestrians are not all sampled due to the sparse sampling strategy – another layer of the network focuses on such smaller objects.

A.2 Visual Examples of RadarPCNN Predictions

This section contains visualizations about the predictions of RadarPCNN. It supports the discussion in section 3.4.3. The 3D visualization as well as the corresponding camera-view are displayed. Both show the input radar point cloud and the ground-truth BBs, colored based on the network predictions. In particular, yellow accounts for moving vehicles and red for moving pedestrians. Orange BBs denote slow-moving vehicles, whose speed belong to the interval $[1, 2.5]$ m/s. Gray BBs shows stationary vehicles – speed lower than 1 m/s – while purple BBs denote

stationary pedestrians. Gray points represent predictions of the negative class. The z-coordinate of the points has been zeroed and aligned with the road for visualization clarity. Best viewed in color.

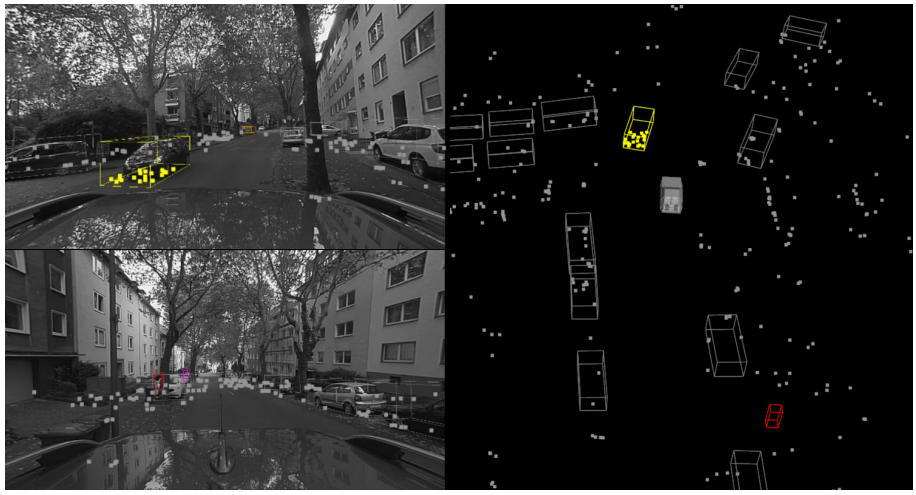


Figure A.3: Incoming moving vehicle correctly detected by the network. Points from all the other parked vehicles are classified as the negative class. The pedestrian on the rear does not have any associated reflection. Since it is occluded by a parked car, the sensor could not see this instance. Top-left: front camera. Bottom-left: rear camera. Right: 3D view.

Figure A.4: The scene contains only static objects and the network can correctly predict the negative class for every point. Top-left: front camera. Bottom-left: rear camera. Right: 3D view.

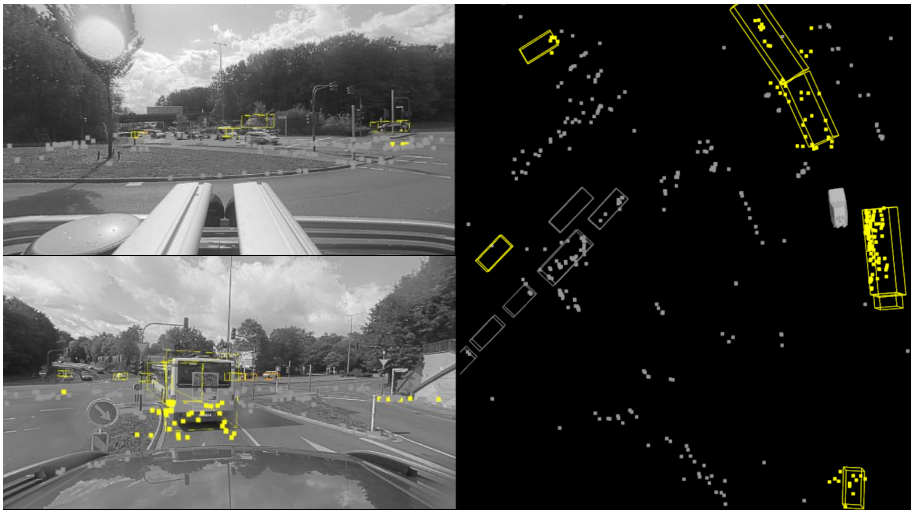


Figure A.5: A complex scene containing both moving and stationary vehicles at a big cross-road. The network can correctly recognize as static the cars in front of the traffic light (on the left). Both large and small moving instances are classified as moving vehicles. The different altitude levels generate visual-errors in the camera views. Top-left: left facing camera. Bottom-left: front camera. Right: 3D view.

Figure A.6: A pedestrian in front of the ego-vehicle, near parked cars, correctly detected by the network. The rest of the scene is classified as static. Top-left: front camera. Bottom-left: rear camera. Right: 3D view.

A.3 RadarPCNN Predictions: Bushes as Pedestrians

This section contains visualizations about the predictions of RadarPCNN, where bushes are erroneously classified as pedestrians. It supports the analysis in section 3.4.3. The figures adopt the visualization-format described in appendix A.2.

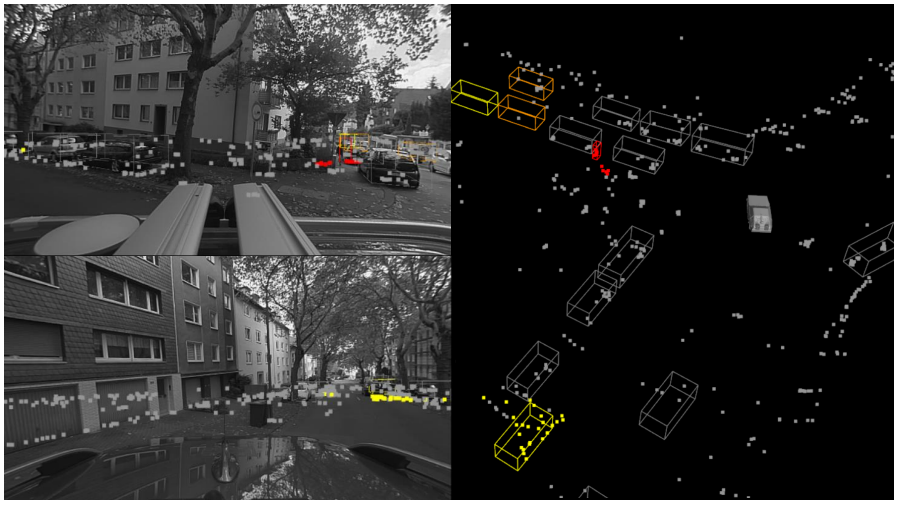
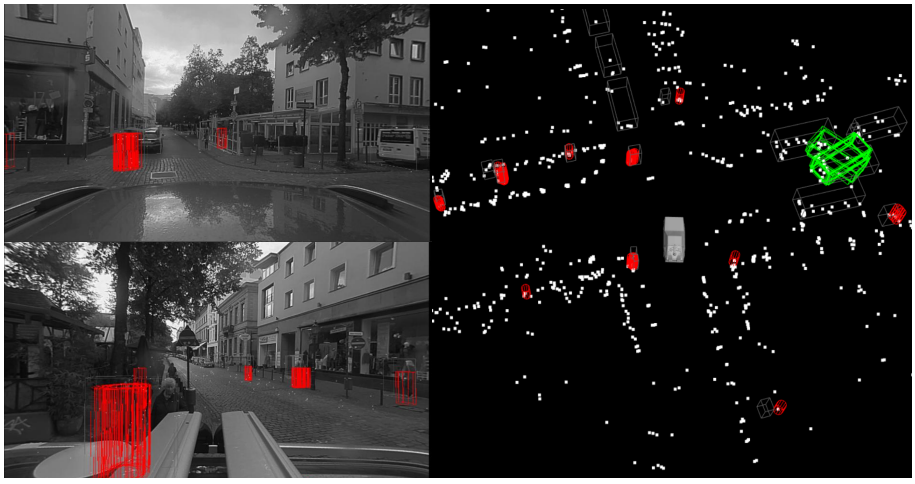


Figure A.7: Bushes from a hedge near the pedestrian are erroneously classified. They share the spatial properties of the nearby pedestrian: both instances are found next to stationary objects. Top-left: left facing camera. Bottom-left: front camera.

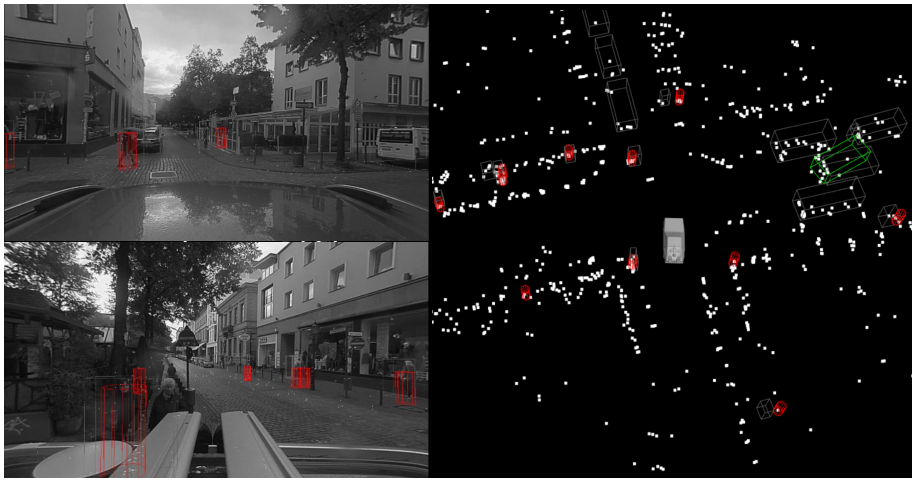
Figure A.8: Bushes on the left and right of the ego-vehicle are mistakenly predicted as pedestrians. Top-left: left facing camera. Bottom-left: right facing camera.

A.4 Visual Examples of Object Box-Predictions

This section contains visualizations about box-predictions of the architecture proposed in chapter 5. Green boxes represent moving vehicle predictions while red cylinders are pedestrians. Gray boxes show GT objects while gray points show the input radar point cloud. The boxes are extended along the z-axis, using a fixed height of $2m$, and aligned with the road plane. Predictions for the same scene before and after NMS are reported. Best viewed in color.

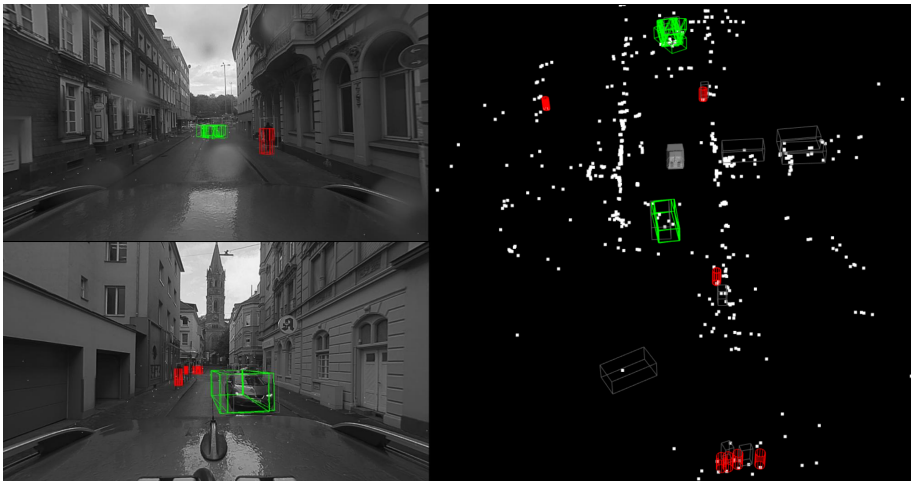


(A) Network box-predictions before NMS.

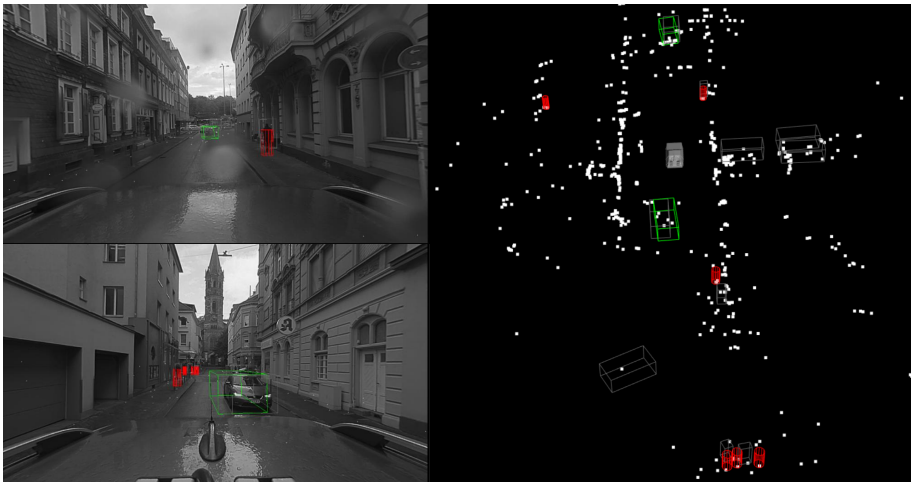


(B) Network box-predictions after NMS.

Figure A.9: Downtown dense scene. The only moving vehicle is detected. Though there are some false positives, most of the pedestrians are detected. The other vehicles in the scene are stationary parked cars. Top-left: front camera. Bottom-left: left-facing camera. Right: 3D view.

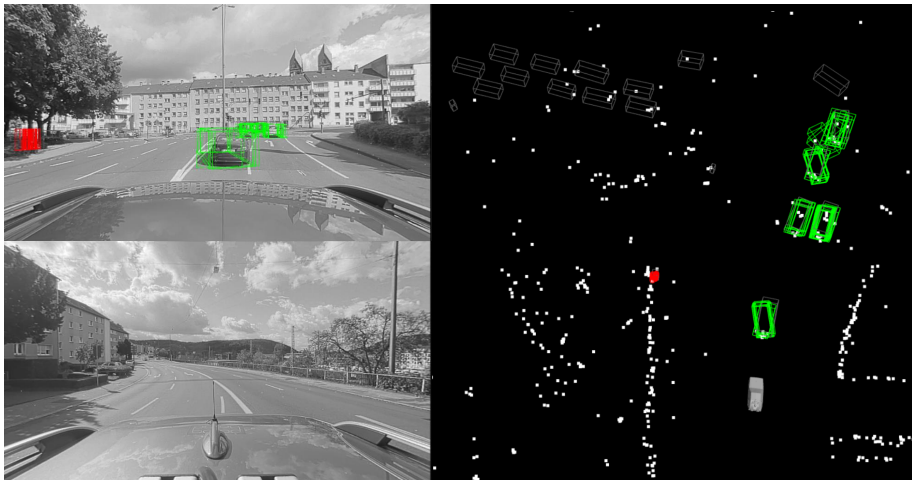


(A) Network box-predictions before NMS.



(B) Network box-predictions after NMS.

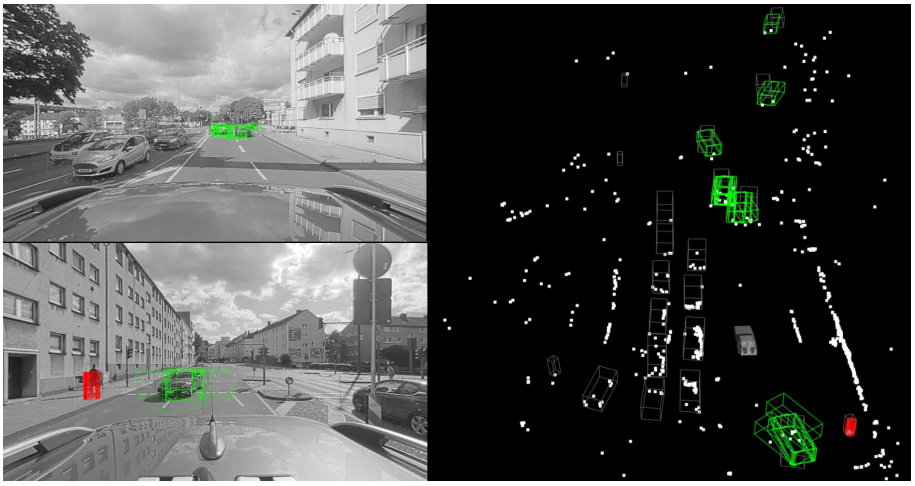
Figure A.10: Downtown scene. Two moving vehicles are correctly recognized. The third moving vehicle in the rear is not detected because it provides a single radar reflection. The pedestrians along the curbstone on the right are detected, but the one in the rear has a location error of few meters. The group of pedestrians in the rear are also correctly recognized, although they are quite far away. The three vehicles on the right represent stationary cars parked in a private area. Top-left: front camera. Bottom-left: rear camera. Right: 3D view.



(A) Network box-predictions before NMS.

(B) Network box-predictions after NMS.

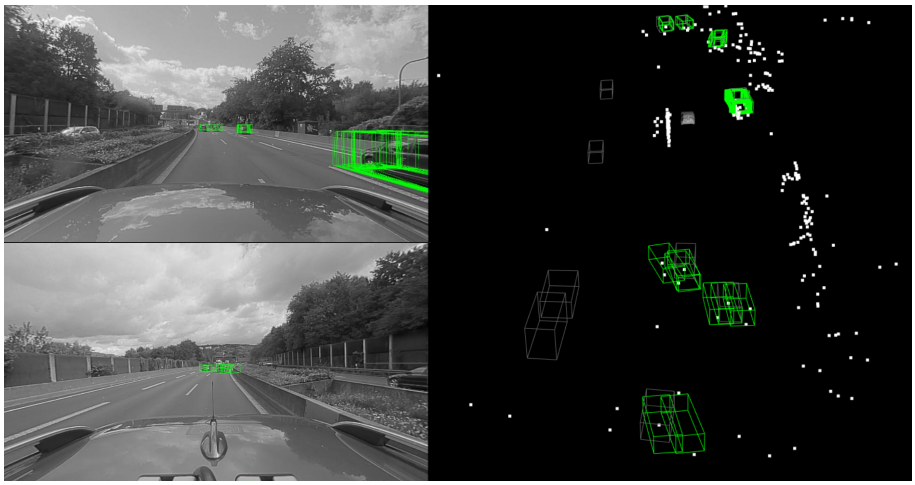
Figure A.11: Urban scene. Moving vehicles and the pedestrian in front are correctly recognized. The two vehicles on the top of the scene are not detected because they have none or few radar reflections. Predictions with correct orientation have higher confidence. The vehicles on the left are standing behind a red traffic light. Top-left: front camera. Bottom-left: rear camera. Right: 3D view.



(A) Network box-predictions before NMS.

(B) Network box-predictions after NMS.

Figure A.12: The pedestrian on the rear-right size and the moving vehicles in the ego-car lane are detected. The vehicles on the left are standing behind a red traffic light. The pedestrian on the rear-left side is missed. The pedestrians on the front-left side cannot be detected because they have no reflections. The network can detect objects very far away, such as the vehicle at the top of the scene. Top-left: front camera. Bottom-left: rear camera. Right: 3D view.



(A) Network box-predictions before NMS.

(B) Network box-predictions after NMS.

Figure A.13: Highway scene. Vehicles in the ego-car roadway are recognized. Vehicles in the incoming roadway are not detected because they do not provide any radar reflections. The network can predict boxes for objects far away. Though multiple boxes are estimated for the vehicles in the rear, the most accurate ones obtain larger confidence values. Top-left: front camera. Bottom-left: rear camera. Right: 3D view.