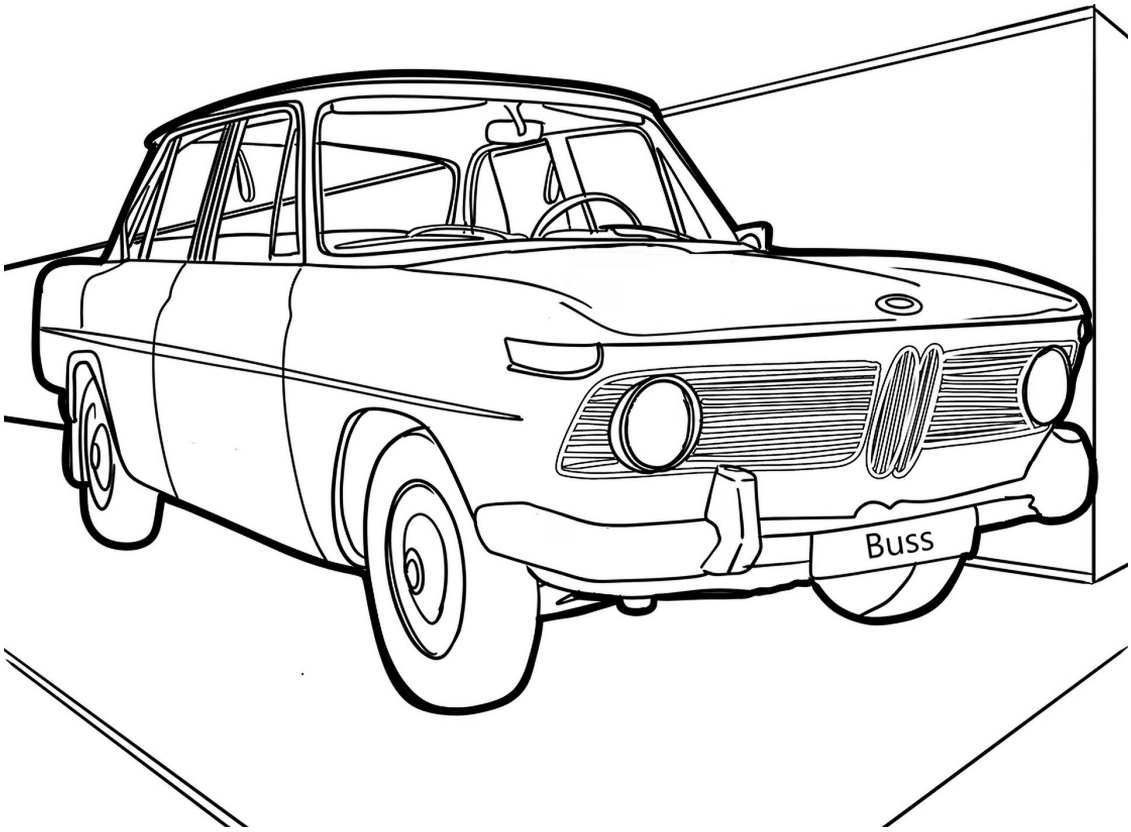
CHALMERS
UNIVERSITY OF TECHNOLOGY



Attention on the road!

Multi-object detection in the radar domain leveraging deep neural networks

Master's thesis in Master Engineering Mathematics

Peter Svenningsson

Department of Mathematical Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

Master's thesis 2020

Attention on the road!

Multi-object detection in the radar domain leveraging deep neural networks

Peter Svenningsson

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mathematical Sciences
Chalmers University of Technology
Gothenburg, Sweden 2020

Attention on the road!
Multi-object detection in the radar domain leveraging deep neural networks
Peter Svenningsson

Supervisors:
Samuel Scheidegger, Gothenburg Research Center, Huawei Technologies Sweden AB
Hossein Nemati, Gothenburg Research Center, Huawei Technologies Sweden AB

Examiner:
Marina Axelson-Fisk, Department of Mathematical Sciences, CTH

Master's Thesis 2020
Department of Mathematical Sciences
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2020

Attention on the road!
Multi-object detection in the radar domain leveraging deep neural networks
Peter Svenningsson
Department of Mathematical Sciences
Chalmers University of Technology

Abstract

Autonomous driving and advanced driver-assistance systems require the perception of the surrounding environment. A subtask in perception is the detection and classification of objects in the environment, commonly referred to as object detection. To aid in this task an autonomous system is outfitted with a sensor suite which commonly include camera, LiDAR and radar sensor modalities.

The contribution of this work is the construction of a novel object detection pipline using a sensor suite of five radar sensors which are capable of detecting objects in a complete field of view. The model constructed is an end-to-end deep learning model which utilizes graph convolutions over radar points to generate a contextualized representation of the sensor data.

It is shown that the model presented is able detect the most commonly occurring classes in the dataset and performs particularly well on objects in motion. It is explored why the model performs poorly on the uncommon classes which stems from limitations in the non-maximum suppression algorithm as well as the low efficacy of the object classifier.

Keywords: Object detection, Radar, Automotive, Geometric deep learning, Attention, nuScenes.

Acknowledgements

I thank the members of the Huawei R&D Gothenburg office for their continued support during this project. In particular, I would like to emphasize the encouragement and expertise provided by Samuel Scheidegger, Hossein Nemati as well as the faculty of the department of mathematical sciences Marina Axelsson-Fisk and Johan Jonasson. I also thank my colleagues Di Xue and Peizheng Yang for the camaraderie and their continued interest in the work.

Peter Svenningsson, Gothenburg, 2020

Contents

1
Introduction

In this chapter one is introduced to the subject of object detection and the aim of this work. Limitations and the scope of the project is discussed as well as the general background for the work.

1.1 Background

For applications in advanced driver-assistance systems (ADAS) and autonomous driving systems (AD) the task of perceiving the environment and modeling the surroundings is paramount. A subset of this task is commonly referred to as object detection - the instantiation and classification of objects. ADAS and AD systems are assisted by three sensor modalities: cameras in the visible spectrum, light imaging based on LiDAR and frequency modulated continuous wave radar sensors. To increase the robustness of downstream utilization it is of interest to create systems for object detection without fusion of these three sensor modalities. Therefore there exists an urgent need for object detection solutions in the radar domain.

Given a set of measurements one is interested in what objects caused those measurements. Object detection is an example of an inverse problem with the related direct problem of finding what measurements a set of objects would cause. Indirect problems are generally difficult to solve because the solution may not be unique as a consequence of measurement noise and ambiguity in the sensor modality. Such problems are often considered ill-posed [1] and may require prior domain knowledge to distinguish a desirable solution.

State of the art object detection systems in the camera and LiDAR modalities are dominated by approaches based on deep learning techniques [2]. Such techniques aim to learn useful and often high dimensional representations of the input data. Clustering or proposal based heuristics are then used to generate object detections represented by a class denomination and a bounding box which specifies the object's physical extent.

Frequency modulated continuous-wave (FMCW) radar sensors measure position and the radial velocity of points in space by sending out radar signals and measuring the radar echo reflected by surfaces. Radio waves are able to penetrate non-conductive materials and can therefore measure objects which are not in line-of-sight, in addition to robustness in various lighting and weather conditions such as rain or snow.

Recent advances in automotive radars, namely a move from the 24 GHz band to the 77 GHz band, has increased the resolution of the radar measurements [3]. The increased resolution makes radar a strong candidate for deep learning methods which benefit from dense representations such as images or dense point clouds.

A common data representation of a radar measurement is a three dimensional tensor with axis specifying discretized values of radial distance, azimuth angle and radial velocity, and elements specifying the strength of the radar echo. Adaptive thresholding algorithms like Constant False Alarm Rate (CFAR) detection can then be used to extract a sparse point cloud in the spatial and radial velocity space. The input for the object detection model defined in this work will be the sparse point cloud extracted by CFAR.

1.2 Aim

The aim of this work is to create a novel model for object detection in the automotive radar setting using deep neural networks. Specifically, the work aims to locate and classify objects such cars, pedestrians and traffic obstacles from a point cloud data representation of radar measurements. The dataset [4] to be used in this work has been recorded and annotated by the company nuTonomy who also hosts an object detection leaderboard for the dataset. It is a target of this work to submit the first object detector model which uses only the radar sensor modality.

1.3 Limitations

Low level sensor data such as radar data tensors is not available in the dataset to be used in this work. This prohibits the use of feature extraction from the tensor data which has shown promising results [5].

FMCW radar units are able to measure velocity in the radial direction with respect to the sensor. Therefore the measured velocity is not representative of the velocity of the measured object. Furthermore, velocity is measured in the vehicles inertial frame and is then compensated by the sensor vehicle's velocity as measured by the vehicle's internal sensors. The efficacy of the vehicle's internal sensors is not known and may introduce an additive bias to the measurements.

1.4 Previous work

Object detection using radar data is currently an open research challenge with only minimal previous work in the field. Earlier work such as by Scheiner et a. [6] clusters the radar point cloud and utilizes a deep neural network (DNN) to classify the clusters. However, this study will focus on end-to-end deep learning solutions for object detection.

Models for object detections using LiDAR data has seen success on the automotive datasets KITTI [7] and nuScenes [4], outperforming camera based detection models. Similarly to the radar sensor modality, the LiDAR data representation is a point cloud. Therefore it is of interest to investigate relevant deep learning architectures for object detection in the LiDAR sensor modality.

Many DNN architectures utilizes the structure in the data representation to extract local patterns in the data, typical examples are convolutional filters [8] used in image analysis and natural language processing (NLP). A point cloud can be considered an unstructured data representation and architectures designed to extract local patterns from point clouds generally use one of three methods to structure the input data: discretizing the input data to two or three dimensional grids suitable for CNN feature extraction, graph construction between proximal points, and extracting the local features of a point by pooling the features of points in its proximity [9].

1.4.1 PointNet

The PointNet architecture presented by Qi et al. [10] utilizes point-wise fully connected layers which takes as input the features of a single point and embeds it in a new feature space. By convention a set of such point-wise feed forward layers separated by activation functions are named a shared multi-layered perception (MLP).

The PointNet architecture encodes a point cloud in two steps. First the points are passed through a shared MLP to embed them in a large feature space. Then global features are extracted from the embeddings by taking the maximum element in each feature dimension (max-pooling). The global features are then concatenated to each point's embedding which forms the complete encoding of the point cloud. The PointNet performs well [10] on 3D model recognition tasks using the ShapeNet-Part dataset [11] which consists of 31963 CAD models covering 16 different shape classes.

PointNet^{++} is an improvement of the PointNet architecture presented by Qi et al. [12]. PointNet^{++} is able to extract local features by using the PointNet architecture to pool the input features of points in small spatial regions. Specifically, a small set of key-points are sampled from the input point cloud. A key-point is then embedded by first generating local features by using the PointNet architecture to pool points within distance ϵ of the key-point for $\epsilon \in \mathcal{E}$. The key-point embedding is then formed by concatenating the generated local features at each distance in $\mathcal{E}$. Non-keypoints are embedded by copying the features of the spatially closest key-point.

PointNet and PointNet^{++} serve as a part of the backbone encoder for many object detection architectures. Models such as Point-GNN [13] and SECOND [14] use the PointNet architecture to project LiDAR point cloud into a three-dimensional grid - a method referred to as voxelization. In contrast, the PointPillar architecture [15] uses PointNet^{++} to project the point cloud to a two dimensional image.

1.4.2 Point-GNN

The Point Graph Neural Network (Point-GNN) [13] is an object detection model for LiDAR input data in which the LiDAR point cloud is first voxelized using a PointNet encoder. A graph is then constructed with edges connecting voxel-nodes within a fixed radius of each other. A deep neural network then further embeds the voxels by performing message passing operations on each voxel-pair connected by an edge. In a message passing operation the embeddings of the transmitting node and the receiving nodes is passed through an MLP to generate a message. A new embedding of the receiving node is generated by max-pooling the received messages.

Point-GNN is a proposal based object detector. Each voxel generates one proposal for an object which includes a class prediction and a prediction for the physical extent of the object. The proposals are then merged or suppressed based on heuristic algorithms similar to non-maximum suppression to generate the set predicted objects.

1.4.3 Object detection for radar tensor data

Some work has explored object detection by leveraging the dense information in the radar data tensor. In work by Palffy et al. [5] CNN filters are used to extract features from the radar data tensor which are then appended to the points generated by CFAR. The extracted features are shown to improve classification metrics in object detection. Work by Major et al. [16] has explored applying object detection architectures popularized in image analysis such as the Single Shot multibox Detector (SSD) on the range-azimuth 2D tensor.

2

Theory

This chapter covers the theoretical framework required to answer the posed research question of object detection in the radar domain. An overview of radar metrology and signal processing is presented. The methodology of deep learning techniques is then presented with an overview of geometric deep learning, attention and object detection techniques.

A FMCW radar operates by transmitting a radar signal which is subsequently mixed with the recieved radar echo. By observing how the radar echo changes over time as measured by a small set of receiver antennas the signal strength in a spatial-velocity space is measured. These measurements are subsequently transformed into a point cloud by finding locally strong signals.

2.1 Radar signal processing

A radar sensor is composed of transmitters which emit electromagnetic waves and receivers which measure the electromagnetic waves reflected by surfaces. A conventional signal processing chain for frequency modulated continuous wave (FMCW) radars generates a point cloud with spatial coordinates $x \in \mathbb{R}^2$ marked with object-velocity in the heading direction of the sensor $v \in \mathbb{R}$, and radar cross section $\sigma \in \mathbb{R}$. By convention these points $p = (x, v, \sigma)$ are named radar detection points and are in this work referred to as radar points to avoid ambiguity. Multiple radar points may be generated by one object instance depending on the size, material properties of the object and the distance to the object.

The radar cross section σ is a measure of the targets' ability to reflect radar signals in the direction of the sensor. In the ideal case this value depends only on the material and the geometry of the target object. However, in practice it is a measure of signal strength normalized by the distance to the object.

A conventional signal processing chain for FMCW radar generates a point cloud. Formally we define a point cloud as a set $\mathbb{P} = \{p_1, \ldots, p_n\}$ where $p_i = (x_i, m_i)$ is a point with spatial coordinates $x_i \in \mathbb{R}^d$ marked with the state vector $m_i \in \mathbb{R}^k$ representing additional point properties. The mark m may include properties measured by a sensor such as signal strength or embedding features generated by a neural network.

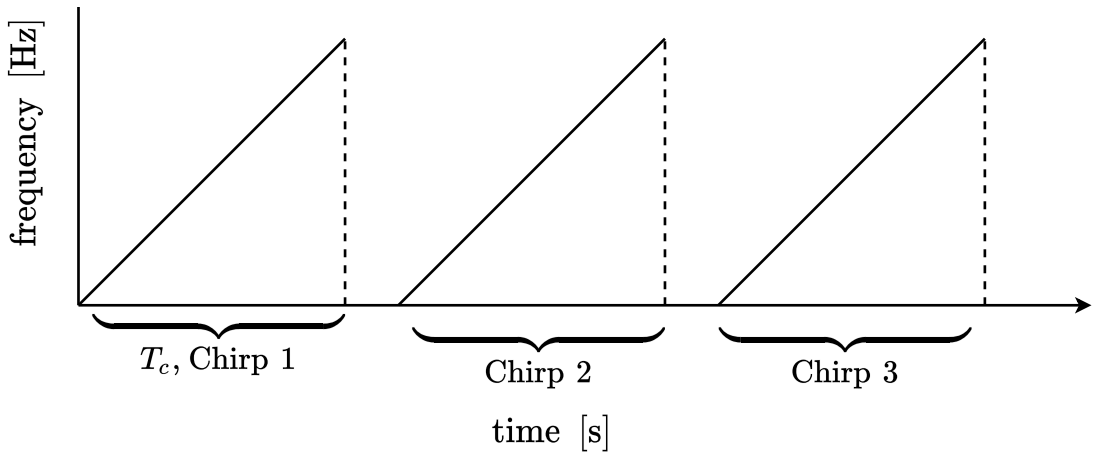


Figure 2.1: A visualization of the transmitter signal of a FMCW radar unit. By convention one linearly increasing segment is named a chirp. The image displays three chirps separated by idle time which allows the signal synthesizer to ramp down after each chirp.

2.1.1 FMCW radar

A FMCW radar unit capable of measuring velocity, range and angular position is composed of one transmitter and multiple receivers. The transmitter outputs a series of short signals which frequencies increases linearly in time as visualized in Figure 2.1; one monotone signal is named a chirp. The signal recorded by a receiver is mixed with the currently output transmitter signal to generate the Intermediate Frequency (IF) signal

$$s_{IF}(t) = \sin\left((w_1 - w_2)t + (\Phi_1 - \Phi_2)\right) = \sin(w_{IF}t + \Phi_{IF}), \tag{2.1}$$

for a transmitter signal $s_1 = \sin(w_1 t + \Phi_1)$ and a received radar echo $s_2 = \sin(w_2 t + \Phi_2)$ where w denotes the signal's frequency and Φ denotes the signal's phase. To enable digital post-processing the signal s_{IF} is sampled and recorded as analog-to-digital converter (ADC) samples.

2.1.2 Radar data tensor

One radar measurement consists of ADC samples recorded over a number of chirps from all of the receiver antennas. The ADC data can be visualized as a three-dimensional tensor as seen in Figure 2.2. The radar data tensor is processed by three fast Fourier transforms (FFT), by convention named 3D-FFT, to extract the range, angle and velocity information.

An FFT is applied across the short-time dimension as visualized in Figure 2.2 to reconstruct the IF signal defined in (2.1). The measured distance

$$d = \frac{w_{IF} c T_c}{2B},$$

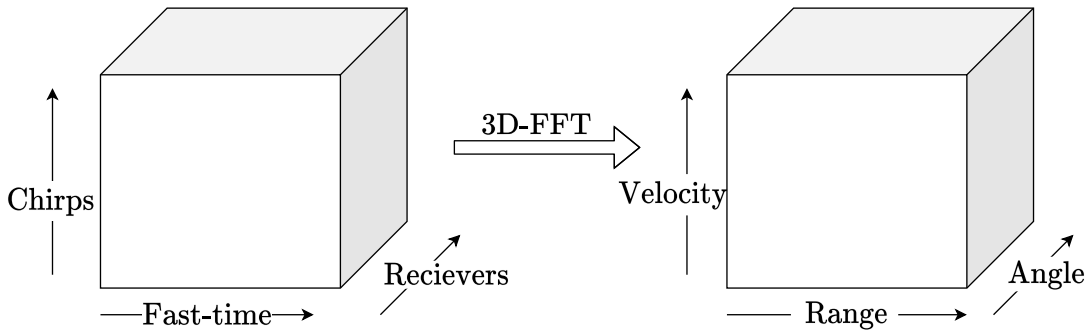


Figure 2.2: An abstract visualization of the three-dimensional data tensor before and after 3D-FFT transformation. The fast-time dimension refers to ADC samples taken within one chirp. In contrast, the chirp dimension is sometimes referred to as slow-time.

where c denotes the speed of light, B denotes the bandwidth of a chirp and T_c denotes the chirp duration [3].

The velocity dimension is recovered by stating that in the time between chirps the position of a measured object in motion has changed only a small amount. It follows that between chirps the frequency of (2.1) is approximately constant for a measured object and the difference in phase of (2.1) between two chirps indicates the velocity of the object relative to the sensor. An FFT is applied across the chirps in each range-bin and the velocity is given by

$$v = \frac{\lambda \Phi_{\Delta c}}{4\pi T_c},$$

where T_c denotes the time between two chirps, λ denotes the wavelength of the signal and $\Phi_{\Delta c}$ denotes the difference in phase between two consecutive chirps [3].

The arrival angle of received radar signals is estimated by observing that receivers separated by a small distance d will receive the radar echo at different phases, see Figure 2.3. Therefore an FFT is applied across the receiver dimension to recover the phase difference $\Phi_{\Delta r}$ across neighboring antennas. The signal's angle of arrival becomes

$$\theta = \sin^{-1} \frac{\lambda \Phi_{\Delta r}}{2\pi d},$$

where d denotes the separation distance between two receiver antennas and λ denotes the wavelength of the signal [3].

2.1.3 Radar point generation

The velocity, range, angle radar data tensor visualized in Figure 2.2 has complex elements with absolute values corresponding to the strength of the signal from a point in the spatial-velocity space. In general there is a large amount of noise in the radar data tensor which stems from signals reflecting from multiple surfaces, background

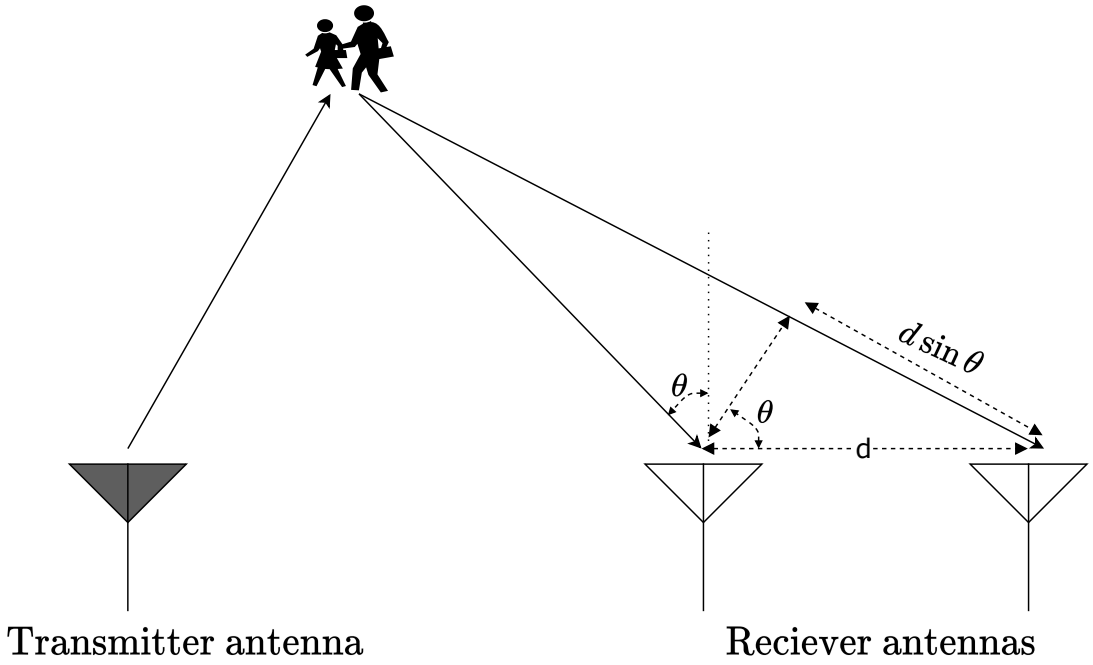


Figure 2.3: A visualization of the geometric properties used to estimate the arrival angle of a radar echo. Note the assumption that the radar signal has not been reflected by multiple surfaces.

electromagnetic radiation and other sources such as reflections from rain particles [3].

The adaptive thresholding algorithm CFAR is used to filter out the noise and extract the informative measurements which comprises the radar point cloud. CFAR estimates the local signal strength s_l by sampling in the neighborhood of a measurement s_p. If the strength of the signal s_p exceeds the local signal's strength s_l by a factor of a threshold τ then the point at s_p is included as a radar point. The threshold τ is set as an algorithm parameter [17].

2.2 Deep neural networks

Supervised learning is a machine learning paradigm in which labeled examples for some task are used to produce hypotheses which generalize to unlabeled data. Commonly, a sequence of parameterized non-linear functions are used to model the label distribution in terms of some covariates. Such models are named deep neural networks (DNN).

To aid in the optimization of a DNN, an objective function is constructed which compares the output of the model to the true label for an example. The objective function may consist of a linear combination of loss functions suitable for classification and regression tasks. The choice of loss functions reflects the multiple tasks a model may perform as well as enable the convergence of the model optimization to a desirable minimum. The object function averaged over all the examples in the

dataset reflects the performance of the algorithm.

2.2.1 ANN

An artificial neural network (ANN) is a computational model consisting of a sequence of linear models and non-linear activation functions. One such linear model is named a neuron and an activation function commonly used is the rectified linear unit (ReLU) as it has a well behaved gradient. The parameterized linear models are fit to minimize an objective function, in this context named a loss function.

In this work, the artificial neural networks considered are feed forward neural networks. A feed forward network consists of layers of neurons. The first layer maps the input covariates to some feature space. Common operations used by a layer are linear transformations and convolutional operations. The second layer takes the embedded input and maps it to another features space. The last layer is named the output layer and outputs a representation useful for some task. Any layer between the input layer and output layer is named a hidden layer. A deep neural network (DNN) is an ANN with many such hidden layers.

Convolutional neural networks (CNNs) utilize parametrized kernels which are convolved across the input. Convolutions are capable of identifying patterns in the input that are invariant under translation. In image analysis CNNs commonly consists of parametrized linear filters which activate on specific patterns and output a response map. Convolutions may also be defined on graphs by defining an operation which acts along the edges of a graph. A general framework for defining graph convolutions is as message passing operations.

To capture a wide range of patterns multiple convolutional operations are performed in parallel. The information captured in the response maps is then summarized by pooling the response maps. A common pooling function is max-pooling which extracts the maximum element in each feature dimension.

A typical use case for artificial neural networks are classification tasks. For such tasks, the ANN is used to approximate a probability distribution over the classes. The distribution is generated by passing the non-normalized output of a network through a normalized exponential function, here named the softmax function

$$\text{Softmax}(z) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}},$$

where z denotes the non-normalized network output.

A perceptron is a single-layered ANN consisting of neurons which are linear functions over the complete set of inputs. The output of the linear models is then passed through an activation function. A multi-layered perceptron (MLP) is a sequence of such perceptrons. In this work a shared-MLP is defined as an MLP for which the

input elements $x \in X$ are passed independently through the MLP to produce the output set X' which contains the same number of elements as X. A use case for a shared-MLP would be to independently embed the points in a radar point cloud.

2.2.2 Optimizer

The parameters of a deep neural network are commonly fit to training examples by variations of the optimization algorithm stochastic gradient descent (SGD) [18]. The SGD algorithm constructs a noisy estimate of the objective function and its gradient based on a single example or a small set of examples. Each model parameter is then updated in the negative gradient direction according to some specified step size.

Adam is a variant of the SGD algorithm in which estimates of the first and second raw moment of the gradient informs the parameter update. In effect, Adam decreases the step size in regions in the loss landscape where the gradient may be large [19] while allowing previous parameter updates to inform the step direction. It has been shown empirically that Adam compares favorably to other stochastic optimization methods for fitting the parameters of a DNN [19].

2.2.3 Loss functions

Loss functions used for classification tasks compare how much the predicted class probability diverges from the actual label. For a set of classes C the cross-entropy loss, given by

$$\mathcal{L}_{\mathbf{ce}} = -\sum_{i \in C} w_i \, y_i \log p_i, \quad y_i \in \{0, 1\}, \quad p_i \in \{x \mid 0 < x < 1\}, \quad w \in \mathbb{R}^+ \tag{2.2}$$

for predicted class probabilities p and label y, is zero-valued for a correct prediction and grows unbounded as $p_j \to 0$, $j : y_j = 1$. For datasets with a large class imbalance the weights w_i are set to downweigh the most common classes [20].

A loss function suitable for segmentation tasks with n predictions with large class imbalances is the class-averaged soft Dice loss [21]

$$\mathcal{L}_{\text{DICE}} = \frac{1}{|C|} \sum_{i \in C} \frac{2 \sum_{j=1}^{n} p_i^{(j)} y_i^{(j)} + \epsilon}{\sum_{j=1}^{n} p_i^{(j)} + \sum_{j=1}^{n} y_i^{(j)} + \epsilon}, \quad \epsilon << 1. \tag{2.3}$$

The soft Dice loss shares similarities with the Dice coefficient [22]

$$\text{Dice}(A, B) = \frac{2|AB|}{|A| + |B|},$$

used to measure similarity of sets A and B.

For regression tasks, two widely used loss functions are the squared error and the absolute error. The squared error has a gradient which is well behaved close to zero

while the absolute error is robust against outliers, i.e. abnormally large errors. The Huber loss [23]

$$\mathcal{L}_{\text{Huber}} = \begin{cases} \frac{1}{2}(y-\hat{y})^2 & \text{if} \quad |y-\hat{y}| \leq \delta \\ \delta|y-\hat{y}| - \frac{1}{2}\delta^2 & \text{else} \end{cases}, \quad \delta \in \mathbb{R}^+, \tag{2.4}$$

is a piecewise function of the squared loss and absolute loss leveraging these two properties.

2.3 Encoder-decoder architecture

Some neural network architectures can be segmented into an encoder-decoder hierarchy. An encoder network maps an input signal to a feature space, and the decoder takes this feature map as input to produce an output, such as a probability distribution.

Convolutional encoder-decoder architectures are commonly used to solve inverse problems such as object detection [24], superresolution [25] and monocular depth estimation [26], visualized in Figure 2.4, outperforming analytical methods on these tasks [27]. Inverse problems are reconstructions of unknown signals, images or sets from observations. The observations may be noisy or generated by a non-invertable process. A solution to an inverse problem is often not unique and analytical approaches leverage prior domain knowledge to generate a desirable solution. In contrast, deep learning models learns to provide the most probable solution based on the training data.

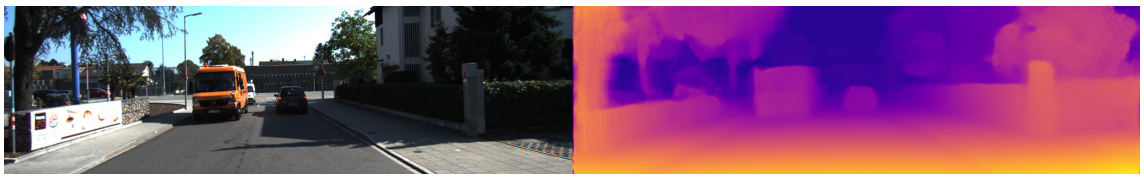


Figure 2.4: A visualization of monocular depth estimation which is a example of an inverse problem solved by a encoder-decoder architecture. The recorded image (left) is used as input to estimate a depth heatmap (right). The figure is adapted from [26].

The encoder, here considered as a CNN, embeds the input in a large feature space. Structure in the data representation, such as grids or sequences, may be used to embed a data point with regards to its local context. Examples of encoders include the U-net architecture [28] for image processing, the BERT architecture [29] for use in natural language processing (NLP) and PointNet^{++} [12] for use on point cloud representation.

A decoder head often consists of an MLP that takes as input an embedded signal and outputs a set of values related to a specific task. For classification tasks

these values are passed through a softmax function to generate a probability distribution over the predicted output classes. For regression tasks the output is often used in a parametrization of the intended output. For example, a regression output may be scaled and centered by the mean annotated value in the training set. In a multi-task setting, the decoder architecture is often comprises several decoder heads.

2.3.1 Geometric deep learning

Conventionally deep learning techniques have been applied on data represented in a grid-like structure - such as a sequence or an image. Convolutions and other operators are then used to extract local patterns. Many interesting applications in deep learning have data which is ill-suited for Euclidian representations such as matrices. Therefore, geometric deep learning has arised as a collective term for deep learning techniques applied on non-Euclidian data such as graphs [30].

The sparsity of radar point cloud measurements makes it ill-suited to be discretized into a grid-like structure. A convolutional operation would encounter many non-informative zero-valued elements at a large computational cost. Therefore, a graph-based data representation may be more suitable for the application of deep learning methods on radar data. In this context, the radar points may be considered as graph nodes and edges may be constructed based on the spatial distance between them. The sparsity of the radar point cloud is further discussed in Chapter 3 and visualized in Figure 3.3.

2.3.2 Message passing

By convention, convolutional operations on graphs are defined in the abstraction of message passing. A message signifies an interaction between two graph nodes. A message passing operation consists of first generating a message along every edge in the graph. A node's embedding is then updated based on the messages it received. Message passing is a general framework capable of capturing many different types of graph convolutions [31]. An encoder can be constructed as a sequence of such message passing operations.

Consider a graph $G = (\mathbb{P}, E)$, with nodes $v \in \mathbb{P}$ and directed edges $e_{i,j} \in E$ defining a connection from node v_i to node v_j. The node $v_i = (x_i, m_i)$ consists of spatial coordinates $x_i \in \mathbb{R}^2$ and is marked by a embedding vector m_i.

A message function

$$b_{i,j} = f(v_i, v_j),$$

constructs a message $b_{i,j}$ along edge $e_{i,j}$. All messages $b_{i,\cdot}$ directed to a node v_i are pooled by a pooling function $\rho(\cdot)$. A new embedding m_i for the node v_i is generated as

$$m_i \leftarrow g\left(\rho\left(\{b_{i,j} \mid e_{i,j} \in E\}\right), v_i\right),$$

where $g(\cdot)$ is some function which further embeds the pooled messages.

2.3.3 Graph attention networks

An attention mechanism is an operation which identifies relevant context and in some way pools the contextual information. Graph attention networks [32] use attention mechanisms to pool the messages received by a node during a message passing operation. A benefit of using an attention mechanisms to pool information is that the operation has a more informative gradient than other pooling functions such as max-pooling. Also, the network may also learn to ignore messages from neighboring nodes which are not informative.

The scaled dot product self-attention mechanism presented in [33] is used in state of the art models for many NLP benchmarks [34] and has also seen use in computer vision [35]. The attention mechanism presented generates a query vector q_i, a key vector k_i and a value vector v_i for each data point $x_i \in X$ using three shared MLPs. A self-attention mechanism gathers context and pools information from the same data representation while a cross-attention mechanism uses different representations for the two tasks [36].

Relevant context with respect to point x_i is quantified by the attention scores

$$s_{i,j} = k_i \cdot q_j, \quad q_j \in \{\mathrm{MLP}_q(x) \mid x \in X\}.$$

The scores $s_{i,\cdot}$ are normalized to non-negative weights w using the softmax function

$$w_{i,j} = \frac{e^{s_{i,j}}}{\sum_{k=1}^{n} e^{s_{i,k}}}.$$

The output embedding z_i is generated as scalar products of the weights $w_{i,\cdot}$ and the value vectors v

$$z_i = \sum_j w_{i,j} v_j.$$

The interaction of key and query vectors allows the mechanism to gather contextualized information without using structures in the data such as grids or sequences [33].

If the input embeddings x_i are first segmented into k segments and the attention mechanism described above is performed independently across the k segments the mechanism is called multi-headed attention with k heads. Multi-headed attention allows the model to pool information from different representation subspaces [33].

In the context of message passing, attention may be used to pool the messages received by node v_i. The set X then comprises the messages received by a node v_i and z_i denotes the new node embedding.

2.4 Object detection

The task of finding all the objects in the input data and assigning them to an object class is commonly referred to as object detection. Many models for object detection are based on first generating a large number of proposals, regions where there might exists an object, which are then classified by a neural network and further filtered heuristically.

Popularized in image analysis, object detectors such as YOLO [37] and SSD [38] are proposal based detectors which generate a large number of proposed objects in a grid-pattern across the image. In contrast, two-stage object detectors such as Faster R-CNN [39] utilizes a second convolutional neural network (CNN) or some other heuristic algorithm which generates region predictions in which objects of interests may reside. The network is then tasked to classify the proposals as some class in C or as part of the image background as well as to adjust the predicted physical extent of the object. LiDAR object detectors such as PointPillar [15] project the point cloud to a two dimensional grid and utilizes image object detectors such as SSD to identify objects in birds-eye-view (BEV).

The Point-GNN [13] LiDAR detector extends proposal based object detection by embedding the point cloud using message passing operations and generates one proposal from every point in the point cloud. Similarly, the proposals are then classified as one of the classes in C or as part of the background.

Proposals which have been classified as belonging to some class in C may be referred to as *detections*. Often one particular object may be covered by many detections. Algorithms such as non-maximum suppression are used to filter out the most informative detection for every object. An overview of the non-maximum suppression algorithm is found in Algorithm 1.

Algorithm 1 The non-maximum suppression algorithm conventionally used to filter out overlapping detections.

iou($\cdot$): The intersection over union of the physical extent of two detection.
Input: $\mathcal{B} = \{b_1, \ldots, b_n\}$, $\mathcal{D} = \{d_1, \ldots, d_n\}$, T
$\mathcal{B}$ is the set of detections
$\mathcal{D}$ is the corresponding detection scores
T is the overlap threshold value
Output: $\mathcal{M}$ is the output set of filtered detections

```
1:  function NMS(B : detections, D : scores)
2:      M ← {}
3:      while B ≠ {} do
4:          i ← argmax(D)
5:          M ← M + b_i
6:          for b_j ∈ B do
7:              if iou(b_i, b_j) > T then
8:                  B ← B − b_j
9:                  D ← D − d_j
10:     return M
```

3
nuScenes dataset

The nuScenes dataset [4] produced by nuTonomy comprises measurements from radar, camera and LiDAR sensors as well as the sensor vehicle's odometry. The data has been collected in the city of Singapore and Boston and consists in total of 5.5 hours of driving divided into 20 s continuous driving sequences.

The dataset is annotated with three-dimensional bounding boxes which inscribe the physical extent of an object. The set of object classes annotated in the dataset can be found in Table 3.1. For a selection of the classes additional attributes are also annotated such as if a car is parked, temporally stopped or moving. However, these properties are not used in this work.

Table 3.1: The number of annotations for each class in the dataset nuScenes [4]. The dataset exhibits class imbalance with the car and pedestrian classes making up a majority of the total annotations.

class	number of annotations
Barrier	152087
Bicycle	11859
Bus	16321
Car	493322
Construction vehicle	14671
Motorcycle	12617
Pedestrian	220194
Trailer	24860
Truck	88519

Camera images are captured at a frequency of 12 Hz, radar and LiDAR measurements are taken at 13 Hz and 20 Hz respectively. Objects in the data have been annotated by a human at a frequency of 2 Hz. A visualization of the available data can be found in Figure 3.1. In this work, a linear interpolation of the position and orientation of the annotating bounding boxes has been used to acquire continuous annotations in the dataset.

The sensors do not capture measurements at the same time. The *frame* at time τ is defined as the collection of measurements from each sensor which was captured closest in time to τ. A frame which coincides with the annotation frequency is named

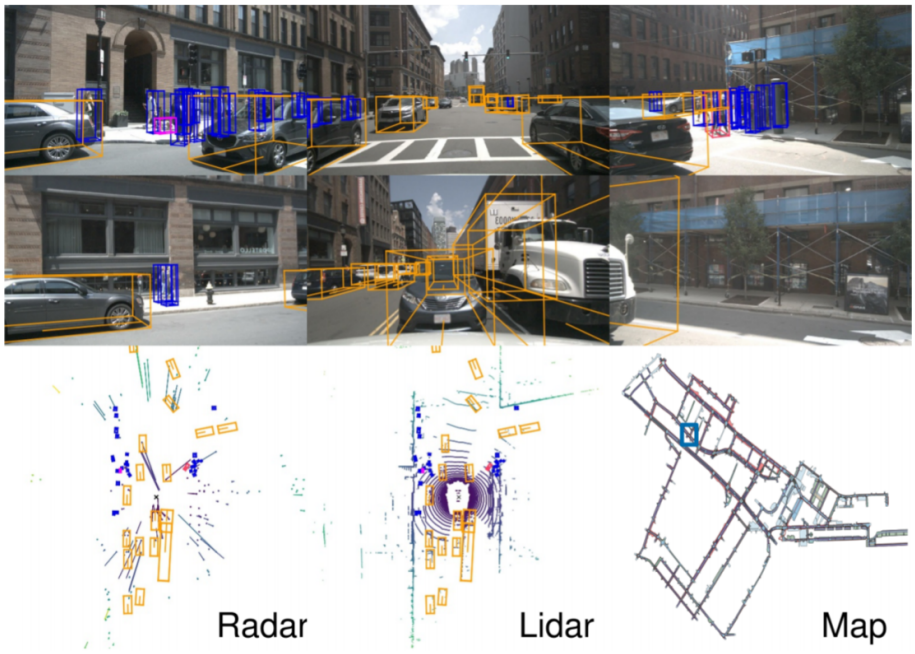
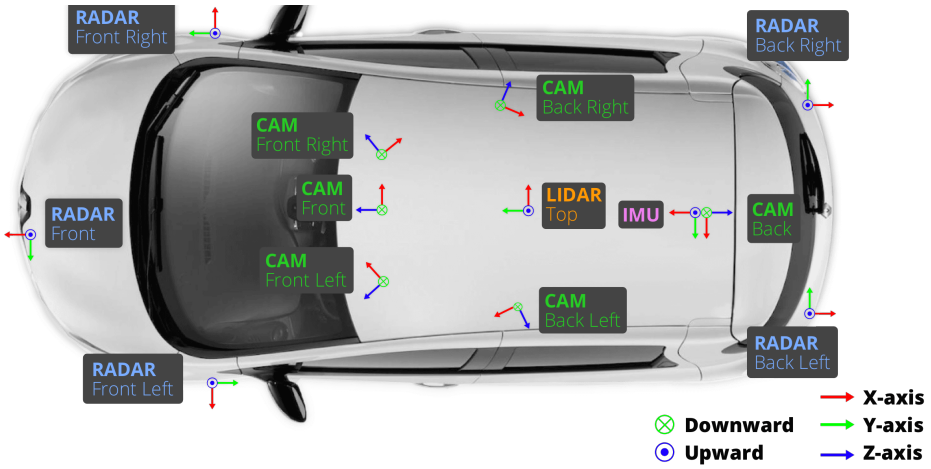


Figure 3.1: A visualization of the three sensor modalities. The radar points are plotted with a vector indicating the measured velocity of the radar point, annotated vehicles and pedestrians are shown in this visualization.

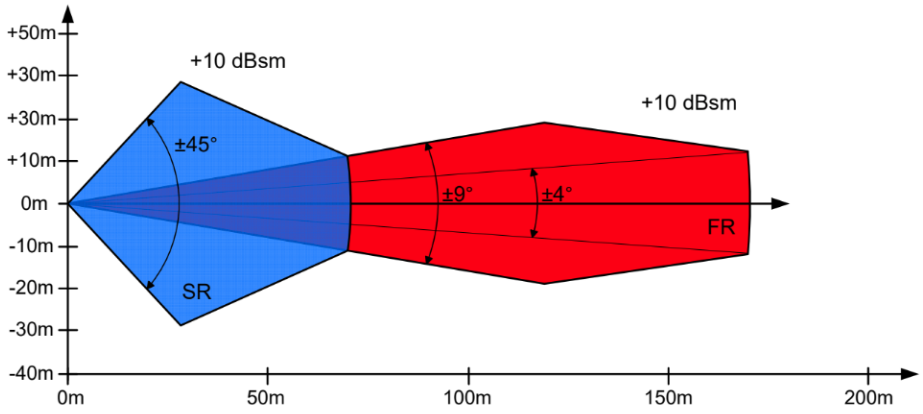
a *keyframe* to mark its importance in calculating performance metrics.

The measurement vehicle is mounted with five radar sensor units as visualized in Figure 3.2a. A radar sensor unit is composed of a short, a medium and a long range radar sensor. The short range sensors have a significantly larger field of view as visualized in Figure 3.2b.

The radar point cloud generated by CFAR is sparse. One can include radar points from previous radar measurements to acquire a denser point cloud. In this work the previous five radar measurements have been translated and rotated to account for the movement of the measurement vehicle and are included in the point cloud. The increase in point density is visualized in Figure 3.3.

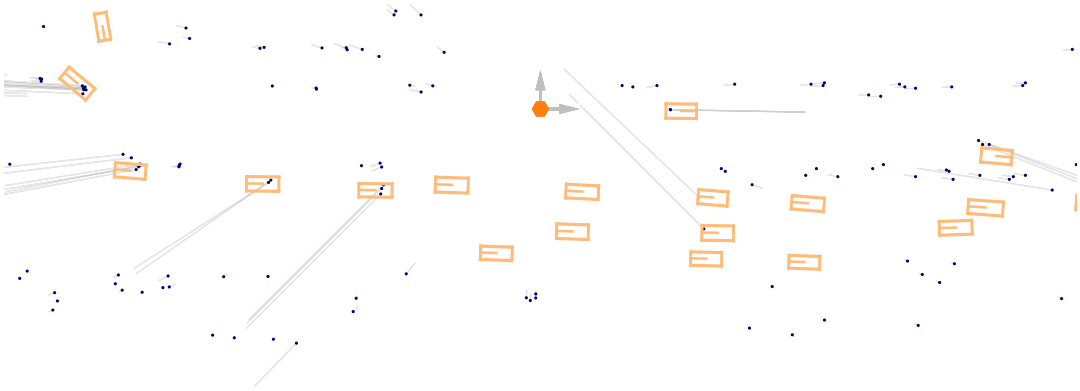


(a) A visualization of the mounted sensors on the measurement vehicle. The vehicle model is a Renault Zoe [4]. The car is mounted with five radar sensor units.

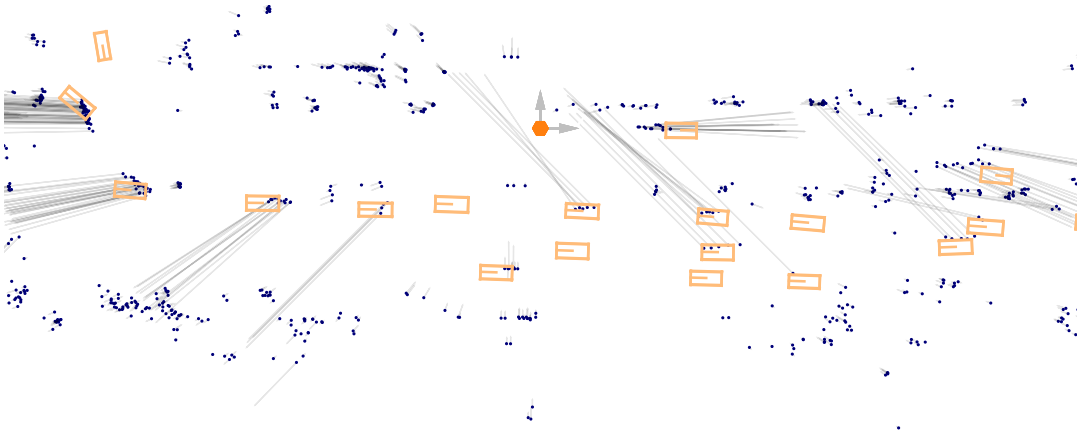


(b) A illustration of the field of view of the short, medium and long range radar sensors which comprises the radar sensor unit.

Figure 3.2: As seen in Figure 3.2b the field of view of the radar sensors is narrow ranges longer than 50 m. The two radar sensors mounted on the back of the car have significantly intersecting intersecting field of views.



(a) A visualization of one radar measurement from the radar sensor suite. Note the sparsity of the point cloud.



(b) Six consecutive measurements visualized in one point cloud. The complete set of measurements have been taken within an interval of 0.5 s. Note the increase in point density in comparison to 3.3a.

Figure 3.3: The radar points ● visualized with annotated vehicles ▯ and the measurement vehicle ⬣. The measured velocity at the radar points are visualized as a vector — with length proportional to the magnitude of the velocity. The increased point density achieved by including previous sensor measurements makes the data representation more suitable for deep learning.

4
Methods

In this chapter the object detection pipeline is described and visualized. The pre-processing steps of the datasets are described as well as the performance metrics used to evaluate the model.

4.1 Pre-processing

The point cloud used as input at frame time τ consists of the six most recent radar measurements from the full radar sensor suite. A categorical covariate t is added to each radar point indicating the age of the measurement with respect to time τ. The measurements are rotated and translated to account for the movement and velocity of the measurement vehicle.

The measurements from the radar sensor suite are transformed to a unified coordinate system centered on the measurement vehicle. This allows for measurements from different sensors to inform the embedding of a radar point.

If a radar point is located inside an annotated bounding box it is assumed that the annotated object generated the radar point. Therefore, a radar point which is located within a bounding box is annotated with the class label and localization parameters of the annotated bounding box. To account for the inherent noise in radar measurements, the size of the bounding boxes are temporally increased by 20% during this process. Any detection points which are not located inside a bounding box are labeled as *Background*.

The annotated bounding boxes are not axis-aligned. The yaw angle ϕ denotes the rotation along the z axis w.r.t the center of the bounding box. With the aim to construct the angle prediction as a classification task, the yaw angle is discretized into eight equisized bins. The motivation is that when using radar data it may be difficult to distinguish the front and back of a vehicle. As a regression task this ambiguity would lead to large losses for predictions that correctly predicted the orientation in 180° but incorrectly distinguished the front of the vehicle. Formulating the task as a classification problem also avoids the problematic discontinuity at $\phi = 0, 2\pi$.

Given a point cloud $\mathbb{P}$ with points $v_i = (x_i, m_i, a_i) \in \mathbb{P}$, $x \in \mathbb{R}^2$ marked with the covariates

$$m_i = (\sigma_i, u_i^{(r)}, t_i)$$

and annotation

$$a_i = (c_i^{(x)}, c_i^{(y)}, h_i, w_i, u_i, \phi_i, y_i),$$

where σ_i denotes radar cross section, $u_i^{(r)}$ denotes radial velocity and t_i denotes the time covariate. The annotation is composed of the center position c_i, height h_i, width w_i, velocity u_i, orientation bin ϕ_i and class label y_i. A graph $G = (\mathbb{P}, E)$ is constructed with the radar points $v_i \in \mathbb{P}$ as vertices and edges

$$E = \left\{ (p_i, p_j) \mid \|x_i - x_j\|_2 < r \right\},$$

including self loops. In this work the radius r is set to 1 m.

Two additional covariates are constructed for each node. The degree of a node, i.e. the number of edges connected to the node, is appended as a proxy for the local point density. Additionally, the distance d from the measurement vehicle to the radar point is also appended as a covariate. The mark then comprises

$$m_i = (\sigma_i, u_i^{(r)}, t_i, \deg(v_i), d_i).$$

The dataset consists of 850 driving sequences which are approximately 20 seconds long [4]. Holdout validation is defined with 700 driving sequences used to fit the model parameters, 100 sequences are used to evaluate model selection and 50 sequences are used as a test set.

4.2 Data augmentation

With the aim to prevent the model from overfitting to the training dataset, noise is added to the samples in the training set. The velocity covariate $u_i^{(r)}$ is scaled as

$$u_i^{(r)} \leftarrow a_v \, u_i^{(r)}, \quad a_v \sim \text{unif}(0.8, 1.2).$$

The positional coordinates are translated as

$$c_i^{(x)} \leftarrow c_i^{(x)} + a_x, \quad a_x \sim \text{unif}(-0.1, 0.1),$$

$$c_i^{(y)} \leftarrow c_i^{(y)} + a_y, \quad a_y \sim \text{unif}(-0.1, 0.1)$$

and the radar cross section σ is translated as

$$\sigma_i \leftarrow \sigma_i + a_\sigma, \quad a_\sigma \sim \text{unif}(-0.04, 0.04).$$

4.3 Model architecture

The model architecture used in this work consists of an encoder which embeds the radar points based on the local context and a decoder which generates one object proposal from the embedding of every radar point. The model parameters are fit

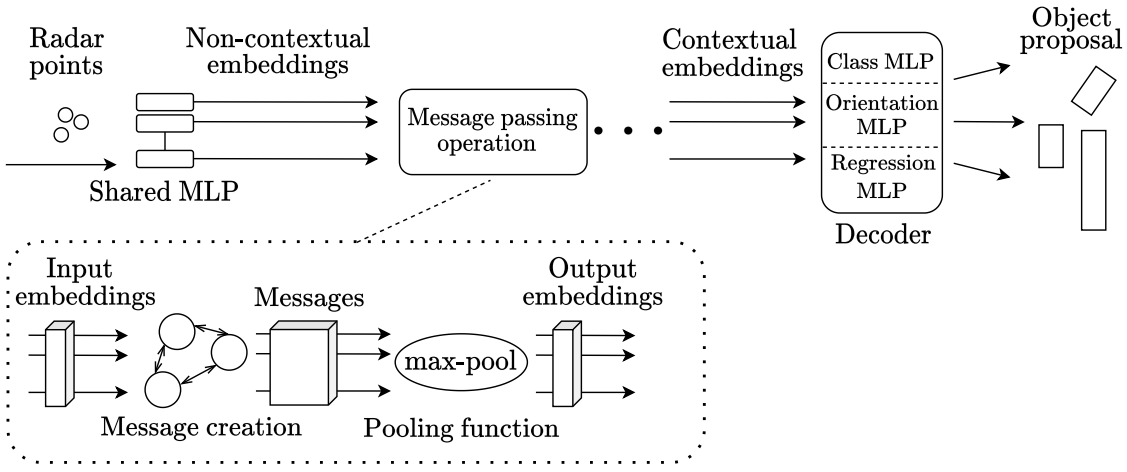


Figure 4.1: A illustration of the object detection network with the graph encoder. Note that $\cdots$ signifies that several message passing operations are performed in sequence. For a network utilizing the attention encoder, the max-pool operation is replaced with a self-attention mechanism.

by comparing the object proposals to object annotations based on a selection of loss functions. At inference, multiple predictions of the same object is not preferable and therefore overlapping proposals are suppressed.

Three encoders are evaluated in this work. The graph encoder and the attention encoder are defined in the geometric setting using message passing operations. The PointNet^{++} encoder is implemented as described in [12]. A visualization of the model architecture is found in Figure 4.1. Note that non-contextual embeddings are generated using a shared MLP. This is the first step for any of the encoders and will not be further mentioned.

4.3.1 Graph encoder

The graph encoder consists of a sequence of message passing operations which uses the max pool function to pool messages. In review of the geometric deep learning theory previously presented, a message function

$$b_{i,j} = f(v_i, v_j),$$

constructs a message $b_{i,j}$ along edge $e_{i,j}$. All messages $b_{i,\cdot}$ directed to a node v_i are pooled by a pooling function $\rho(\cdot)$. A new embedding m_i for the node v_i is generated as

$$m_i \leftarrow g\left(\rho\left(\{b_{i,j} \mid j : e_{i,j} \in E\}\right), v_i\right),$$

where $g(\cdot)$ is some function which further embeds the pooled messages. In this work the functions considered for $f(\cdot)$ and $g(\cdot)$ are MLPs with detailed information found in Appendix A.

With the aim to capture local structures in the dataset rather than overfitting to global features, the absolute coordinates of the radar points are not used as covariates. Instead the relative position $x_j - x_i$ and the input embedding m_j is used to define the operation

$$\begin{aligned} b_{i,j} &\leftarrow f\left(x_j - x_i, m_j\right) \\ m_i &\leftarrow g\left(\rho\left(\{b_{i,j} \mid j : e_{i,j} \in E\}\right), m_i\right), \end{aligned} \tag{4.1}$$

with ρ selected as the max pool function.

As visualized in Figure 4.1, multiple message passing operations are performed in sequence. The number of operations as well as the MLP parameters can be found in Appendix A.

4.3.2 Attention encoder

The attention encoder shares many similarities with the graph encoder. It uses the message operation defined in (4.1). However, a self-attention mechanism is defined to pool the messages $b_{i,\cdot}$.

Given a set of messages directed to node i, $B_i = \{b_{i,j} \mid j : e_{i,j} \in E\}$, an attention operation is defined by generating a key vector k_i , query vectors $q_\cdot$ and value vectors $v_\cdot$. These are constructed by passing the messages $b_{i,\cdot}$ through the multi-layered perceptrons: MLP_q, MLP_k and MLP_v.

Attention scores

$$s_{i,j} = k_i \, q_j, \quad q_j \in \{\text{MLP}_q(b_{i,j}) \mid b_{i,j} \in B_i\}, \quad k_i = \text{MLP}_k(b_{i,i}),$$

are calculated and passed through a softmax function to generate the attention weights

$$w_{i,j} = \frac{e^{s_{i,j}}}{\sum_{k=1}^{n} e^{s_{i,k}}}.$$

The pooled message z_i is then constructed as a weighted scalar product of the value vectors

$$z_i = \sum_j w_{i,j} v_j, \qquad v_j \in \{\text{MLP}_v(b_{i,j}) \mid b_{i,j} \in B_i\}.$$

The output embedding is then calculated analogously to the graph encoder architecture

$$m_i \leftarrow g\left(z_i, m_i\right).$$

Note that the inclusion of the m_i as a covariate to the MLP $g(\cdot)$ may be interpreted as a skip connection.

4.3.3 Mixed encoder

The mixed encoder consists of a graph encoder followed by a attention encoder. The motivation is that the max pooling operation has shown to be robust to noise [40] and could therefore extract robust features. However, the attention pooling operations used later in the network provides the opportunity to learn the size of the local receptive field. The model parameters used for the mixed encoder can be found in Appendix A.

4.3.4 PointNet^{++} encoder

In this work, the PointNet^{++} encoder serves as a baseline architecture. The architecture embeds radar points by pooling the input features of points in small spatial regions. One can find a detailed description of the architecture in Section 2.7.1 or in [12]. In this work the subsampling processes used in PointNet^{++} was removed to account for the sparsity of radar point cloud. Features were extracted from spherical spatial regions with radius $r = \{0.2\,\text{m}, 1\,\text{m}\}$ in addition to the global features. The model parameters used in this work can be found in Appendix A.

4.3.5 Decoder

The decoder outputs one object proposal for every point in the point cloud. The architecture consists of three multi-layered perceptions (MLPs) which in parallel takes an embedded point as input, see Figure 4.1. The classification head consists of an MLP which outputs a probability distribution over the object classes as well as a binary prediction if the proposal is an object or not, here named the objectness score. The orientation head is an MLP which outputs a probability distribution over the discrete orientation bins.

The MLP regression head outputs five scalar values. The center position of the proposed bounding box (x, y) is regressed as

$$x = x_{point} + \delta_x,$$
$$y = y_{point} + \delta_x,$$

where (x_{point}, y_{point}) is the coordinates of the input radar point and (δ_x, δ_y) are the regressed scalars. The width and the height is regressed in the parameterization

$$h = \bar{h}_{class} + \delta_h,$$
$$w = \bar{w}_{class} + \delta_w,$$

where (δ_h, δ_w) are the regressed values and $(\bar{h}_{class}, \bar{w}_{class})$ denotes the median height and width of the class. The absolute velocity u is regressed as a positive scalar and it is assumed that direction of travel is the same as the orientation angle.

4.4 Loss functions

The loss functions used in this work compare how dissimilar an object proposal generated from radar point v is to the annotation of v. The loss function

$$\mathcal{L} = c_1\mathcal{L}_{classification} + c_2\mathcal{L}_{localization}, \quad c_1,\, c_2 \in \mathbb{R}^+ \tag{4.2}$$

is composed of a linear combination of the classification loss $\mathcal{L}_{classification}$ and the localization loss $\mathcal{L}_{localization}$.

For a point cloud $\mathbb{P}$

$$v_i \in \mathbb{P}, \quad v_i = (x_i, m_i, a_i),$$

with point-wise annotation

$$a_i = (c_i^{(x)}, c_i^{(y)}, h_i, w_i, u_i, \phi_i, y_i),$$

the model outputs a predicted class distribution p_i and a objectness prediction $p_i^{(o)}$ for each radar point $v_i \in \mathbb{P}$. If the objectness prediction is smaller than threshold τ_{object} then the proposal is classified as background. The model also outputs the localization parameters $l_i = (\hat{c}_i^{(x)}, \hat{c}_i^{(y)}, \hat{h}_i, \hat{w}_i, \hat{\phi}_i, \hat{u}_i)$ which define the center coordinates, the height, width, orientation and the absolute velocity of the predicted bounding box.

The classification loss is defined as

$$\mathcal{L}_{classification} = c_3\mathcal{L}_{DICE}(B) + c_4 \sum_{(p_i^{(o)}, y_i^{(o)}) \in A} \mathcal{L}_{ce}(p_i^{(o)}, y_i^{(o)}) + c_5 \sum_{(p_i, y_i) \in B} \mathcal{L}_{ce}(p_i, y_i), \tag{4.3}$$

where

$$A = \{(p_i^{(o)}, y_i^{(o)}) \mid v_i \in \mathbb{P}\}, \quad y_i^{(o)} = \begin{cases} 1, \text{if } y_i = \text{Background} \\ 0, \text{else} \end{cases}$$

is the set of objectness predictions with corresponding annotation and

$$B = \{(p_i, y_i) \mid y_i \neq \text{Background},\, v_i \in \mathbb{P}\}$$

is the set of predictions and annotations for non-background annotations.

The localization loss is defined for proposals which are correctly classified

$$\mathcal{L}_{localization} = \sum_{v_i \in C} \sum_{(\hat{q}, q) \in Q_i} c_q\mathcal{L}_{\text{Huber}}(\hat{q}, q) + c_\phi\mathcal{L}_{ce}(\hat{\phi}_i, \phi_i),$$

$$Q_i = \{(\hat{c}_i^{(x)}, c_i^{(x)}),\, (\hat{c}_i^{(y)}, c_i^{(y)}),\, (\hat{h}_i, h_i),\, (\hat{w}_i, w_i),\, (\hat{u}_i, u_i)\},$$

$$C = \{v_i \mid \operatorname*{argmax}_i(p_i) = y_i,\, p_i^{(o)} > \tau_{object}\},$$

where C is the set of correctly classified proposals and Q are the localization parameters output by the decoder with the corresponding annotation.

To avoid large losses from training examples with many radar points and annotations, the classification loss and the localization loss are averaged as

$$\mathcal{L}_{classification} = b_1\mathcal{L}_{DICE}(B) + \frac{1}{|A|}b_2\sum_{(p_i^{(o)},y_i^{(o)})}\mathcal{L}_{ce}(p_i^{(o)}, y_i^{(o)}) + \frac{1}{|B|}b_3\sum_{(p_i,y_i)\in B}\mathcal{L}_{ce}(p_i, y_i), \quad (4.4)$$

$$\mathcal{L}_{localization} = \frac{1}{|C|}\sum_{v_i\in C}\sum_{(\hat{q},q)\in Q_i} c_q\mathcal{L}_{\text{Huber}}(\hat{q}, q) + c_\phi\mathcal{L}_{ce}(\hat{\phi}_i, \phi_i). \quad (4.5)$$

The constants $c_\cdot$ are used to scale the losses.

4.4.1 Optimizer

To optimize the object detection model with regards to the loss function described, this work utilizes the Adam optimizer. The learning rate is changed according to the learning rate schedule displayed in Figure 4.2. The optimization parameters used in this work can be found tabulated in Appendix A.

The learning rate schedule starts with a warmup segment with a low learning rate. The motivation is that the Adam optimzer needs a large set of previous updates to correctly estimate the moments of the gradient. The practice of using a warmup segment has been motivated by previous empirical studies [41]. The learning rate is decrease towards the end of the training which has show to help convergence of the optimization algorithm [42].

4.5 Non-maximum suppression

At inference, non-maximum suppression as described in Algorithm 1 is used to suppress spatially overlapping predictions. The intersection-over-union (IoU) calculation at step 7 in Algorithm 1 is calculated in the x–y plane. The overlap threshold T, found in Appendix A, is selected as a small value with the motivation that objects in the dataset are seldom overlapping.

It is less computationally expensive to calculate the intersection-over-union (IoU) of rectangles which are axis-aligned than of those which may be oriented in any direction. Therefore, the IoU$(\cdot)$ implementation used in this work calculates the IoU metric of axis-aligned rectangles which inscribe the predicted bounding boxes. A GPU accelerated implementation of the IoU calculation [43] was tested and showed to be orders of magnitude slower than the axis-aligned CPU implementation using the python library numpy.

The scoring value used to select the most confident object proposal is the sum of the predicted objectness $p_i^{(o)}$ and the selected orientation probability $\max(\phi_i)$.

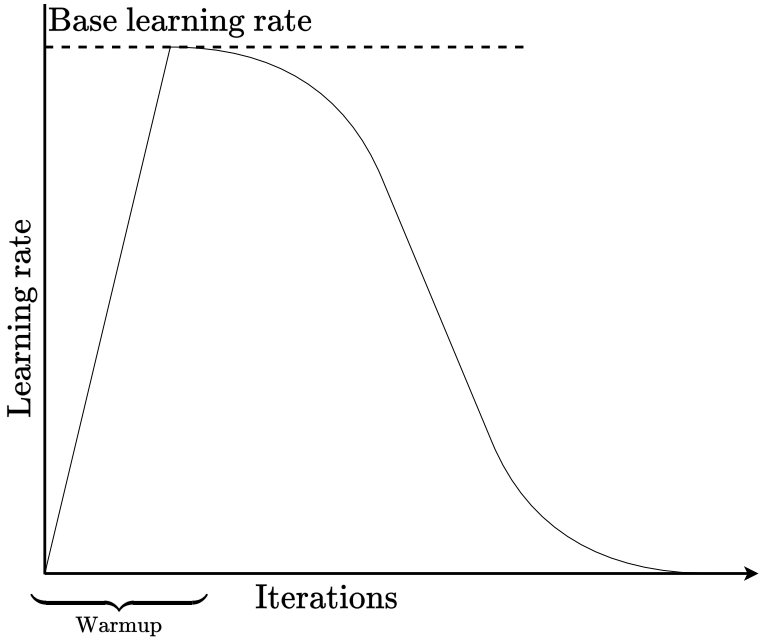


Figure 4.2: A visualization of the learning rate schedule used in this work. The schedule consists of a linear function splined with a half wave cosine function. The learning rate determines the step size used in the parameter update.

The motivation being that a confident object proposal should be confident of the proposed class and the spatial orientation.

4.6 Performance metrics

The purpose of the performance metrics presented here is to measure the similarity between the set of predicted objects output by the model and the set of annotated objects in the dataset. In this work the average precision (AP) metric is used to evaluate the detection and classification performance for a class. In addition a set of localization metrics are defined to evaluate the performance of the position, size, orientation and velocity predictions.

The annotated objects are matched with the closest predicted object within 3 meters by distance measure in the ground plane between the object centers. An annotated object is at most matched with one predicted object and any predicted object which is not matched with an annotation is considered a false positive. Any annotation which do not contain a radar point is removed from the dataset.

4.6.1 Average precision

In this work, classification is considered in a binary setting for each class. In binary classification the quantities true positives (TP), false positives (FP), true negatives

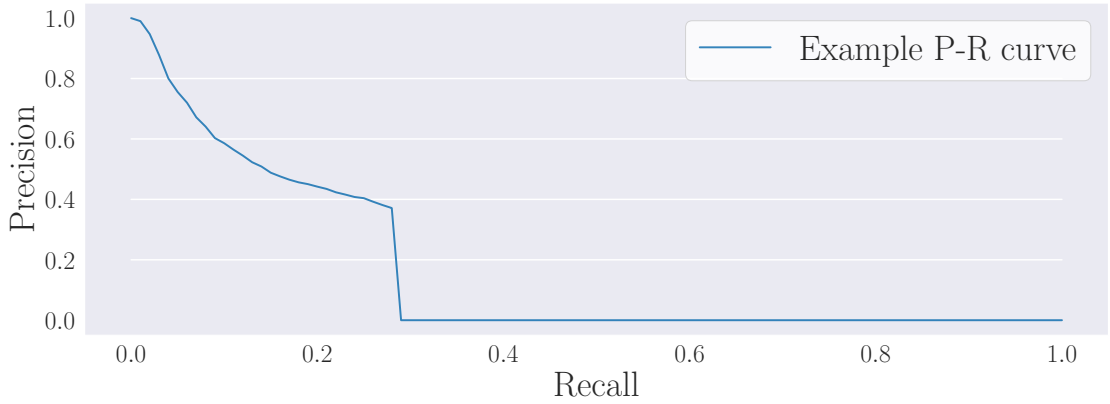


Figure 4.3: A example of a Precision - Recall curve. The sharp drop in precision stems from that undetected objects are considered false negatives at all thresholds τ.

(TN) and false negatives (FN) are used to define metrics such as

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} = \frac{\text{TP}}{\text{all predictions}},$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{\text{TP}}{\text{all annotations}}.$$

A prediction is considered positive if the generated probability is larger than some threshold τ. A prediction is considered true if the class-prediction is consistent with the class of the matched annotation.

One can calculate several recall R_τ and precision P_τ values by varying the threshold τ. To evaluate a classifier independent of the threshold τ one can interpolate an R, P curve from the R_τ, P_τ values, visualized in Figure 4.3. To summarize the information in the R, P curve in one scalar one can calculate the area under the curve, this metric is named the average precision (AP).

4.6.2 Localization metrics

For any correctly classified prediction it is of interest to measure how well the model predicted the physical extend, the heading and the velocity of the object. For a predicted object $b_{pred} = (\mathbf{x}_{pred}, \phi_{pred}, \mathbf{v}_{pred})$ with center position $\mathbf{x}$ and orientation ϕ and an annotated object b_{ann}, the translating error

$$e_\delta = \|\mathbf{x}_{pred} - \mathbf{x}_{ann}\|_2$$

and the orientation error

$$e_\Phi = \Delta\phi = |\phi_{pred} - \phi_{ann}|$$

measure the models ability to predict the heading and the position of an object. A visualization of these metrics is found in Figure 4.4.

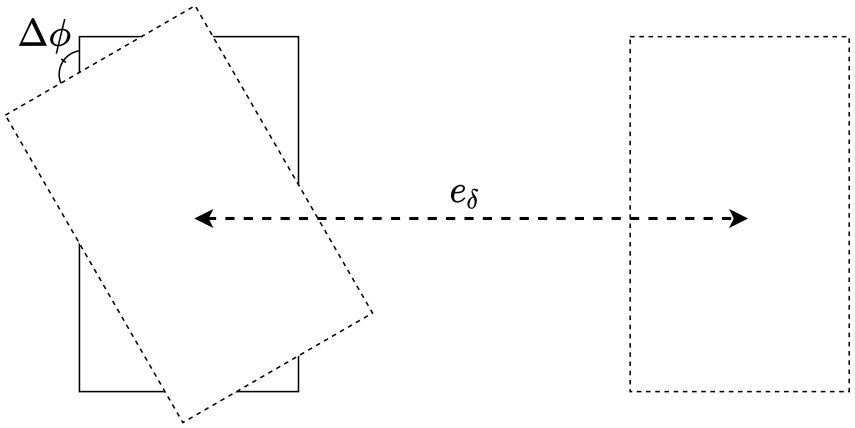


Figure 4.4: Visualization of the translation error e_δ and the orientation error ϕ for the predicted bounding box □ the annotated bounding box ⌜⌝.

The fidelity of the predicted size of the object is measured by first aligning the predicted and the annotated object as visualized in figure 4.5. The intersection-over-union (IoU) of the predicted object bounding box and the annotated bounding box after alignment measure how well the width and height prediction reflects the annotation.

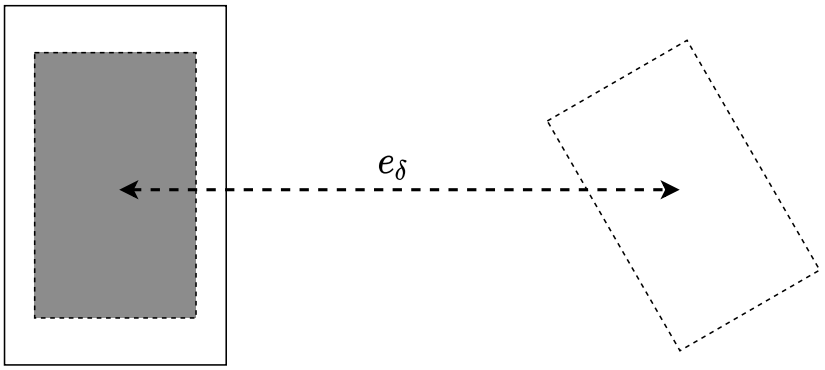


Figure 4.5: Visualization of the intersection area for the predicted bounding box □ and the annotated bounding box ⌜⌝ after center and orientation alignment.

Lastly, the velocity error $e_v = \|\mathbf{v}_{\text{pred}} - \mathbf{v}_{\text{ann}}\|_2$ is used to measure the velocity prediction. Note that the predicted velocity vector $\mathbf{v}_{\text{pred}}$ has length u_{pred} and is parallel to the predicted heading of the object. In summary, the localization metrics are e_δ, e_ϕ, e_v and IoU which measure how correct the object prediction is beyond the class classification.

5
Results

In this chapter one finds a comparison of the effectiveness of the different encoders. The performance of the best performing encoder is evaluated in detail. An ablation study verifies various aspects of the methodology such as the engineered features.

5.1 Quantitative results

The quantitative results comprise statistics on how well the different encoders performed in the object detection task. This comparison focuses on the performance on the two most prevailing object classes, *Cars* and *Pedestrians*. No encoder detected any class other than the two most prevailing classes on the test set. The object threshold $\tau = 0.5$ for the comparison.

Table 5.1: A selection of the performance metrics as evaluated on the test set. The Car and Pedestrian classes are the most common classes in the dataset. The mixed encoder achieves the highest AP while the graph encoder performs well on the localization metrics.

		Performance metrics			
	AP	e_δ	e_ϕ	IoU	e_u
Graph encoder					
Car	0.20	0.68	0.22	0.65	1.06
Pedestrian	0.13	0.39	0.33	0.60	0.46
Attention encoder					
Car	0.18	0.83	0.32	0.64	0.99
Pedestrian	0.15	0.42	0.38	0.60	0.59
Mixed encoder					
Car	0.21	0.77	0.27	0.64	0.93
Pedestrian	0.15	0.43	0.51	0.60	0.62
PointNet encoder					
Car	0.16	1.05	0.24	0.63	1.46
Pedestrian	0.10	0.47	0.38	0.58	0.60

5.1.1 Graph encoder

The detection model using a graph encoder performed well on the localization tasks. In particular, the model achieved low translation and orientation errors compared to the other models as seen in Table 5.1. The precision - recall curve displayed in Figure 5.1 shows that model achieves low recall values. The low recall is a consequence of the model not detecting objects in the scene which is reflected in a large number of false negative classifications.

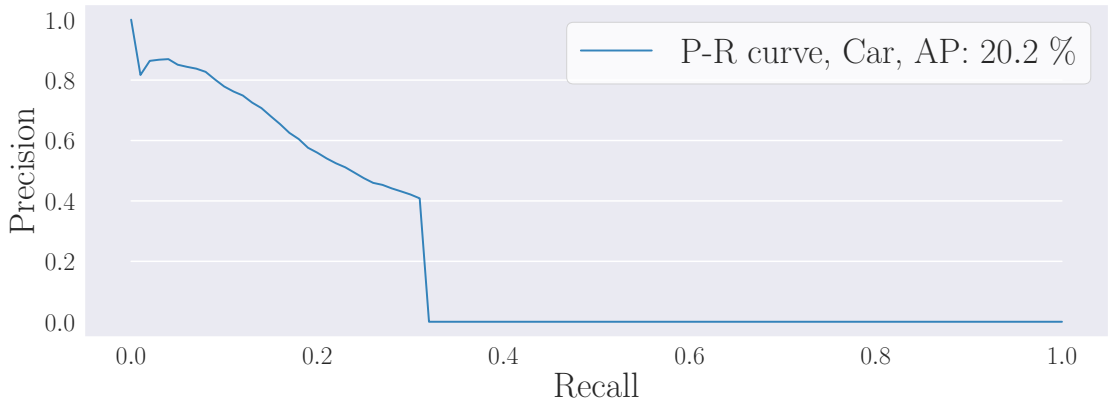


Figure 5.1: The precision - recall curve for the *Car* class. The detection model uses the graph encoder with an objectness threshold of 0.5.

5.1.2 Attention encoder

A detection model with the attention encoder achieved similar performance to the graph encoder. The attention encoder performed well on the Pedestrian class as can be seen in Table 5.1. In comparison to the graph encoder, the P-R curve for the attention encoder displayed in Figure 5.2 indicates larger recall and lower precision for the *Car* class.

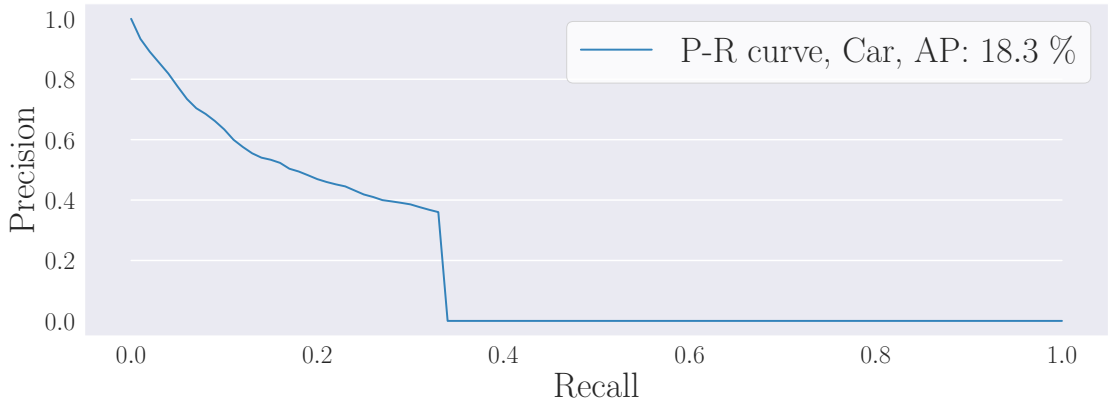


Figure 5.2: A plot of a Precision - Recall curve for the attention encoder. The sharp drop in precision stems from that undetected objects are considered false negatives at all thresholds τ.

5.1.3 Mixed encoder

The mixed encoder achieved the highest AP metrics as can be seen in Table 5.1 and the highest recall as seen in Figure 5.3. The encoder performed worse in terms of localization metrics. However, since localization is only evaluated for correctly classified objects the localization metrics are not directly comparable.

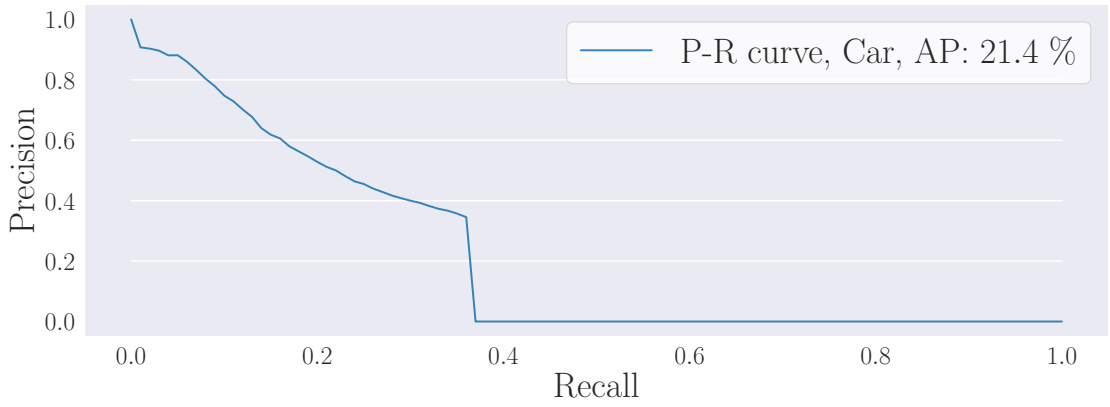


Figure 5.3: The Precision - Recall curve generated by the mixed encoder. The encoder achieved the highest AP among the encoders considered.

5.1.4 PointNet^{++}

The PointNet^{++} encoder acheived the lowest AP for both classes. The AP and other metrics can be viewed in Table 5.1. The P-R curve for the *Car* class is displayed in Figure 5.4.

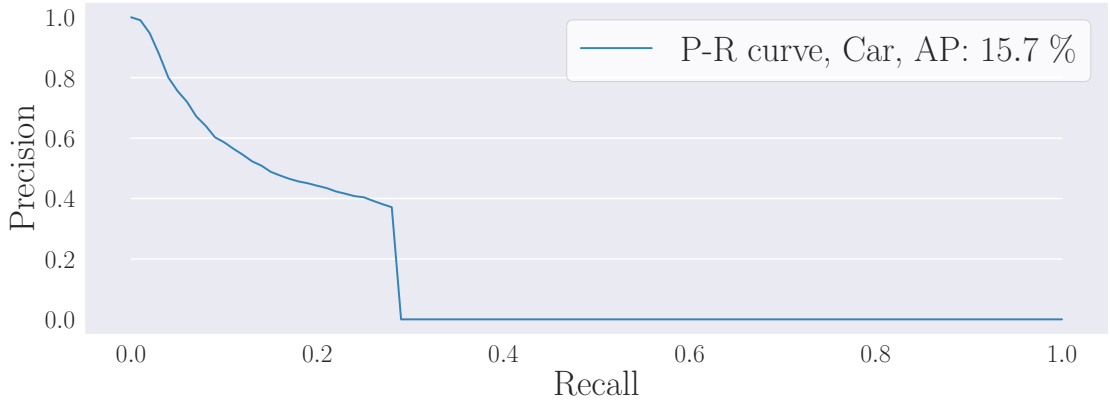


Figure 5.4: The Precision - Recall curve generated by the PointNet^{++} encoder for the *Car* class. The PointNet^{++} encoder underperformed in comparison to the other encoders achieving the lowest AP.

5.2 Qualitative results

The performance of the object detection model is here visualized in bird's eye view (BEV) supplemented by camera images. The examples displayed are curated to illustrate model performance under specific circumstances. The model does not predict the height of an object and therefore the height of closest annotated object is used when visualizing the predicted objects in the camera images.

The detection model underperforms in examples with stationary objects such as the parking lot visualized in Figure 5.5. A majority of the stationary objects are not detected and this limitation is found consistently when evaluating the model. Furthermore, the heading direction of the predicted vehicle in the sample is incorrect.

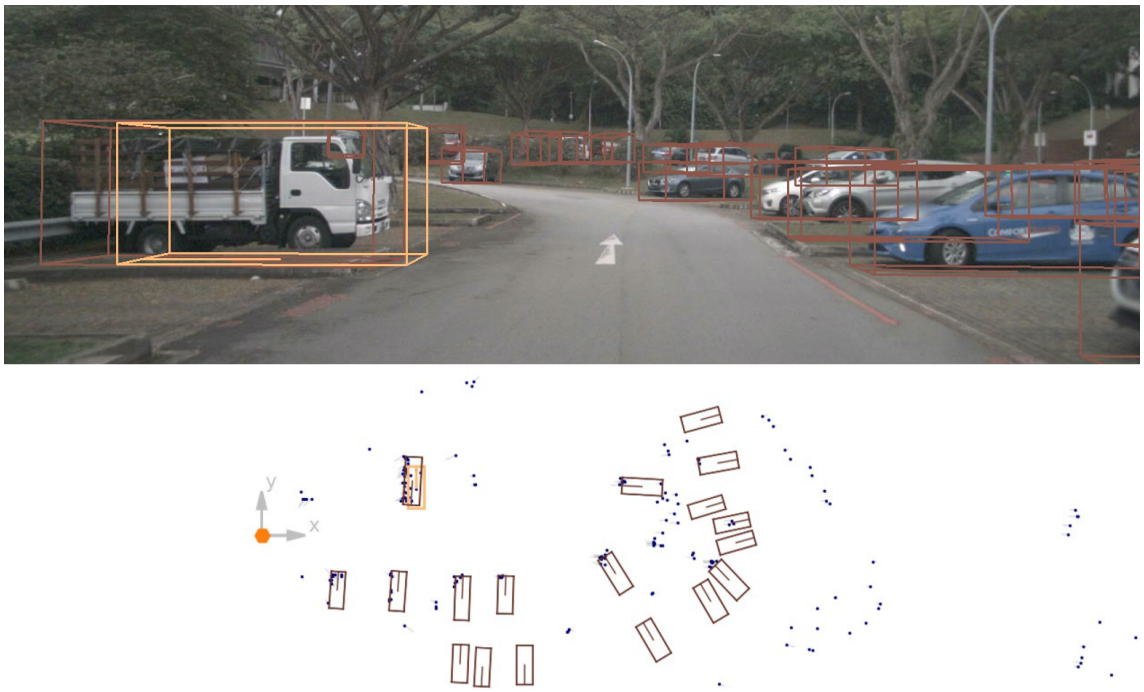


Figure 5.5: The model underperforms on examples with stationary objects such as in a parking lot. Note that only one vehicle is detected and the model incorrectly identified the heading direction. Detected objects ▭ are visualized with annotated objects ▭, radar points ● and the measurement vehicle ⬢. The heading direction of an object is indicated with —.

The detection model performs well in detecting objects with a large measured velocity as visualized in Figure 5.6. The model's ability to predict the localization metrics reflects the results in Table 5.1 with the largest translation error being approximately 2 m. The objects in the scene are detected consistently throughout the driving sequence.

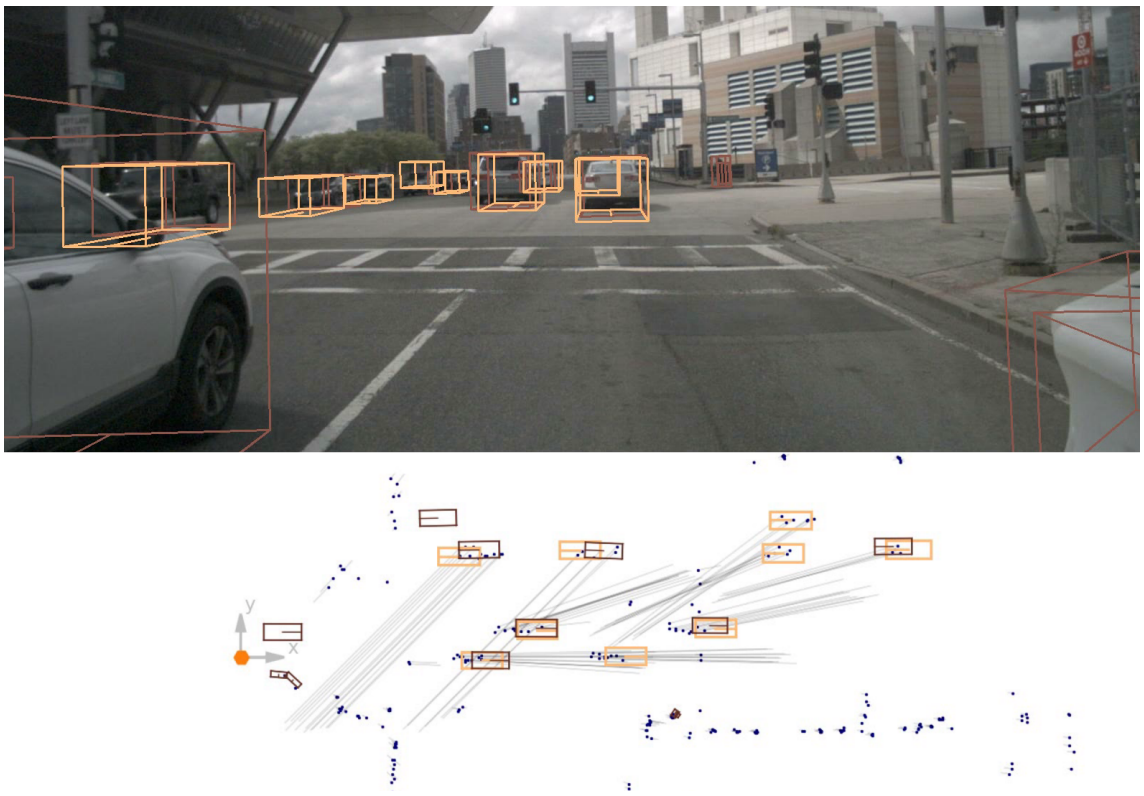


Figure 5.6: A visualized sample which includes several non-stationary objects. Detected objects ▯ are visualized with annotated objects ▯, radar points ● and the measurement vehicle ⬢. The heading direction of an object is indicated with — and the measured velocity of a radar point is visualized with the vector — with length proportional to the magnitude of the velocity.

The visualized samples in Figure 5.5, 5.6 indicate that the model has a stronger performance on moving objects than on stationary objects. However, the radar sensors only measure velocity in the radial direction w.r.t. the sensors. Therefore objects which move in the tangential direction w.r.t the sensor have a low measured velocity and the object detection model underperforms on these cases as well. The number of these cases is significant in common traffic scenarios such as intersections.

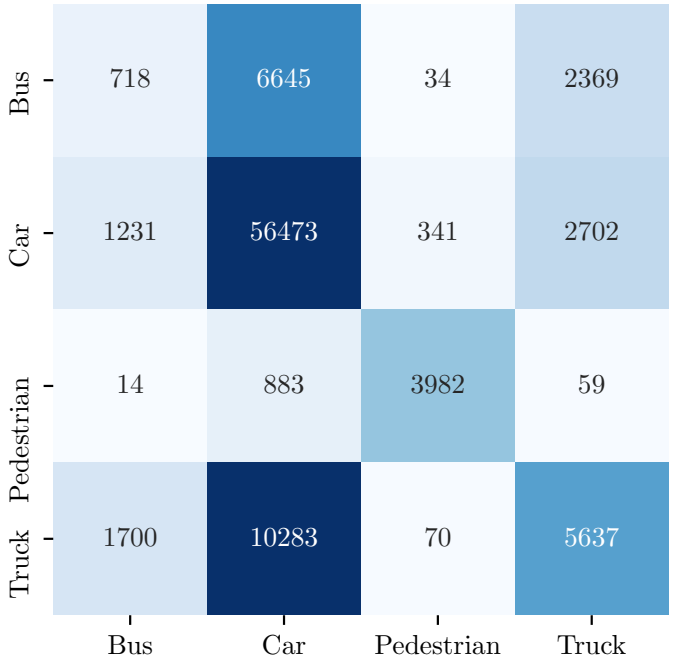


Figure 5.7: A confusion matrix for a selection of the classes in an object proposal classification task. The result is generated from the test set and proposals which have a predicted objectness score lower than 0.5 are not included.

5.3 Detailed results

The mixed encoder had the highest performance with respect to AP and its results are reviewed here in detail. Only objects in the *Pedestrian* and *Car* classes were detected in the test set. It is therefore of interest to investigate the object proposals' classification metrics to quantify the models ability to detect the remaining classes before non-maximum suppression. The model's performance in classifying the object proposals is visualized as a confusion matrix in Figure 5.7. It is apparent that the model correctly classifies some proposals as the remaining classes. However, these are then suppressed by NMS.

The qualitative results indicated that the model had better performance on moving objects. To quantify this difference a model was trained to detect only moving objects. As a pre-processing step all radar points with zero-valued measured radial velocity were removed. The model was then evaluated on annotations with non-zero valued velocity. The performance metrics for non-stationary objects can be found in Table 5.2 and the P-R curve is shown in Figure 5.8. Notably, the model's performance increased on the classification and localization tasks.

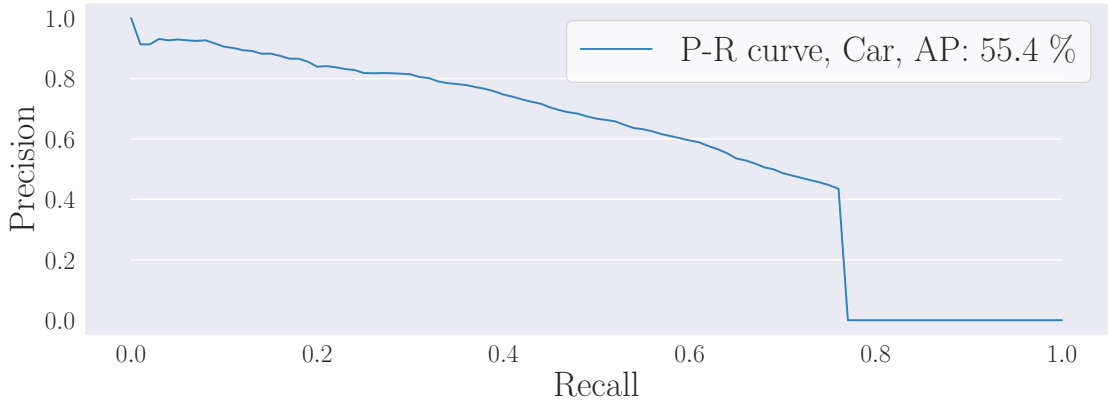


Figure 5.8: The Precision - Recall curve for non-stationary cars using an object detection model with the mixed encoder. Note the increase in maximum recall in comparison to Figure 5.3.

Table 5.2: A selection of the performance metrics of the object detection model on non-stationary objects using the mixed encoder. Note the decrease in e_ϕ in comparison to the results in Table 5.1.

Class	AP	e_δ	e_ϕ	IoU	e_u
Car	0.55	0.70	0.12	0.64	1.27
Pedestrian	0.21	0.35	0.25	0.60	0.40

5.4 Ablation study

The aim of the ablation study is to evaluate if specific model design choices or other elements of the methodology presented are beneficial. The design choices here explored are the selection of coordinate system, the addition of engineered covariates such as the point density and the inclusion of the orientation confidence in the non-maximum suppression score.

5.4.1 Choice of Covariates

It is of interest to evaluate the importance of the covariates used in this work. With this aim, models were trained with one of the covariates masked and evaluated on the test set. The results displayed in Table 5.3, show that removing the time covariate increases the translation error e_δ for potentially fast moving objects such as cars. The change in performance when masking the density or range covariate is in contrast negligible. The addition of the absolute coordinate as a covariate in the generation of the non-contextual embeddings increases the performance for the *Car* class and decreases the performance on the *Pedestrian* class.

Table 5.3: Performance metrics for a detection model using the mixed encoder with a selection of covariates removed or added. The **time covariate** indicates the time when a radar point was generated. The **density covariate** indicates the number of edges connected to a radar point. The **range covariate** denotes the distance from the radar point to the measurement vehicle. The addition of the **absolute coordinate** of a radar point as a covariate is also explored.

		Performance metrics			
Removed covariate	AP	e_δ	e_ϕ	IoU	e_u
Original model					
Car	0.21	0.77	0.27	0.64	0.93
Pedestrian	0.15	0.43	0.51	0.60	0.62
Time covariate					
Car	0.19	0.94	0.29	0.64	1.05
Pedestrian	0.14	0.42	0.44	0.60	0.57
Density covariate					
Car	0.20	0.74	0.27	0.64	0.91
Pedestrian	0.15	0.41	0.49	0.62	0.64
Range covariate					
Car	0.19	0.70	0.26	0.62	1.01
Pedestrian	0.15	0.31	0.41	0.61	0.62
Added covariate	AP	e_δ	e_ϕ	IoU	e_u
Absolute coordinate					
Car	0.23	0.55	0.21	0.64	0.92
Pedestrian	0.11	0.34	0.23	0.58	0.44

5.4.2 Choice of Coordinate system

In this work the radar points from the five sensors have been transformed to a coordinate system centered on the measurement vehicle. Since the radar sensors are only able to measure velocity in the radial direction it is investigated whether keeping the radar points in the sensor coordinate system is beneficial for the model's performance. A model is trained on radar points in their respective sensor coordinate system with no edges constructed between measurements from different sensors. The results are found in Table 5.4. The model performance decreases when trained and tested on radar points in the sensors' coordinate system.

Table 5.4: Performance metrics for a detection model using the mixed encoder. **Sensor coordinates** refers to keeping the radar points in the respective sensor's coordinate system. The **objectness NMS** referrers to using a NMS scoring function which disregards the orientation confidence.

		Performance metrics			
Modification	AP	e_δ	e_ϕ	IoU	e_u
Original model					
Car	0.21	0.77	0.27	0.64	0.93
Pedestrian	0.15	0.43	0.51	0.60	0.62
Sensor coordinates					
Car	0.16	0.91	0.23	0.64	1.30
Pedestrian	0.11	0.84	0.49	0.61	0.64
Objectness NMS					
Car	0.21	0.78	0.28	0.64	1.01
Pedestrian	0.16	0.41	0.33	0.61	0.50

5.4.3 Choice of NMS scoring function

The non-maximum suppression algorithm defined in Algorithm 1 requires a scoring function to act as a proxy for the confidence of object proposal. In this work the scoring function has been defined as the sum of the predicted objectness and the predicted probability of the selected orientation. The model was evaluated with only the objectness probability included in the scoring function and the results can be found in Table 5.4. The inclusion of the orientation probability in the scoring function did not affect the model's performance.

6
Discussion

In this chapter one finds a discussion regarding the performance and limitations of the presented work. In addition, areas of future work are explored.

6.1 Conclusion

The work presented has shown that end-to-end deep learning methods for object detection in the radar modality is a viable approach, in particular for objects in motion. However, the performance is hindered by two limitations. It is difficult for the model to distinguish stationary objects from the surrounding environment and proposals from uncommon classes are consistently supressed by the non-maximum supression algorithm.

Any object detected by the model is generally covered by several object proposals. Some of the proposals may be incorrectly classified as indicated by the confusion matrix in Figure 5.7. The NMS algorithm selects the most confident proposal as measured by some scoring function. In this work the NMS algorithm rarely selected the uncommon classes which indicates that the objectness probability used in this work is a poor proxy for proposal confidence and is systematically overestimated for the common classes like *Car* and *Pedestrian*.

Simply training a more effective classifier would lower the number of incorrectly classified proposals and therefore mitigate the suppression of uncommon classes. Another approach would be to cluster the embedded radar points in some way and assume that the points in a cluster is generated by one unique object - circumventing the need for NMS. Finding a more suitable scoring function or using a heuristic such as majority vote for classification could also be beneficial.

It is challenging for an object detector operating on the radar point cloud to distinguish a parked vehicle from the surrounding environment. The materials such as metal and plastic which comprise a car are also common in the environment. Furthermore, the geometry of a car might not be descriptive as it is roughly a rectangle in the x–y plane. Therefore it is likely necessary to provide more information to the model in order to achieve the performance needed for application in the automotive industry. For example, it is possible that the cross-section of the car in the range-azimuth plane could distinguish the car from background elements such as a chain

link fence.

6.2 Future work

Radar data in any representation can be difficult to annotate and often requires additional sensor modalities in support of the effort. Given the difficulty of annotating radar data it would be interesting to explore the construction of self-supervised tasks as a pre-training stage. The intention is that the model will learn to embed the input data in a representation useful for other tasks such as object detection. A simple self-supervised task might be to predict the next range-azimuth heatmap in a sequence.

Recent work [44] have explored instance segmentation of point cloud representations using loss functions which explicitly separates the embeddings of different instances and constricts embeddings of points which belong to the same instance. At inference, the embeddings are clustered to produce a set of objects, in this context named an instance segmentation. It would be interesting to use this methodology for object detection in the radar domain with the addition of classifying the generated cluster.

Bibliography

[1] Jacques Hadamard. Sur les problèmes aux dérivées partielles et leur signification physique. *Princeton university bulletin*, pages 49–52, 1902.

[2] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11621–11631, 2020.

[3] Sujeet Milind Patole, Murat Torlak, Dan Wang, and Murtaza Ali. Automotive radars: A review of signal processing techniques. *IEEE Signal Processing Magazine*, 34(2):22–35, 2017.

[4] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11621–11631, 2020.

[5] Andras Palffy, Jiaao Dong, Julian FP Kooij, and Dariu M Gavrila. Cnn based road user detection using the 3d radar cube. *IEEE Robotics and Automation Letters*, 5(2):1263–1270, 2020.

[6] Nicolas Scheiner, Nils Appenrodt, Jürgen Dickmann, and Bernhard Sick. A multi-stage clustering framework for automotive radar data. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 2060–2067. IEEE, 2019.

[7] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361. IEEE, 2012.

[8] Kunihiko Fukushima and Sei Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer, 1982.

[9] Yulan Guo, Hanyun Wang, Qingyong Hu, Hao Liu, Li Liu, and Mohammed Bennamoun. Deep learning for 3d point clouds: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[10] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[11] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su,

et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

[12] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in neural information processing systems*, pages 5099–5108, 2017.

[13] Weijing Shi and Raj Rajkumar. Point-gnn: Graph neural network for 3d object detection in a point cloud. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1711–1719, 2020.

[14] Yan Yan, Yuxing Mao, and Bo Li. Second: Sparsely embedded convolutional detection. *Sensors*, 18(10):3337, 2018.

[15] Alex H Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 12697–12705, 2019.

[16] Bence Major, Daniel Fontijne, Amin Ansari, Ravi Teja Sukhavasi, Radhika Gowaikar, Michael Hamilton, Sean Lee, Slawomir Grzechnik, and Sundar Subramanian. Vehicle detection with automotive radar using deep learning on range-azimuth-doppler tensors. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 0–0, 2019.

[17] Wai Kai Chen. *The electrical engineering handbook*. Elsevier, 2004.

[18] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[19] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[20] Qi Wang, Yue Ma, Kun Zhao, and Yingjie Tian. A comprehensive survey of loss functions in machine learning. *Annals of Data Science*, pages 1–26, 2020.

[21] Carole H Sudre, Wenqi Li, Tom Vercauteren, Sebastien Ourselin, and M Jorge Cardoso. Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations. In *Deep learning in medical image analysis and multimodal learning for clinical decision support*, pages 240–248. Springer, 2017.

[22] Lee R Dice. Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302, 1945.

[23] Peter J Huber. Robust estimation of a location parameter. In *Breakthroughs in statistics*, pages 492–518. Springer, 1992.

[24] Zhong-Qiu Zhao, Peng Zheng, Shou-tao Xu, and Xindong Wu. Object detection with deep learning: A review. *IEEE transactions on neural networks and learning systems*, 30(11):3212–3232, 2019.

[25] Wenming Yang, Xuechen Zhang, Yapeng Tian, Wei Wang, Jing-Hao Xue, and Qingmin Liao. Deep learning for single image super-resolution: A brief review. *IEEE Transactions on Multimedia*, 21(12):3106–3121, 2019.

[26] Clément Godard, Oisin Mac Aodha, and Gabriel J Brostow. Unsupervised monocular depth estimation with left-right consistency. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 270–279, 2017.

[27] Alice Lucas, Michael Iliadis, Rafael Molina, and Aggelos K Katsaggelos. Using deep neural networks for inverse problems in imaging: beyond analytical methods. *IEEE Signal Processing Magazine*, 35(1):20–36, 2018.

[28] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[29] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[30] Wenming Cao, Zhiyue Yan, Zhiquan He, and Zhihai He. A comprehensive survey on geometric deep learning. *IEEE Access*, 8:35929–35949, 2020.

[31] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018.

[32] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[33] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[34] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

[35] Niki Parmar, Prajit Ramachandran, Ashish Vaswani, Irwan Bello, Anselm Levskaya, and Jon Shlens. Stand-alone self-attention in vision models. In *Advances in Neural Information Processing Systems*, pages 68–80, 2019.

[36] Andrea Galassi, Marco Lippi, and Paolo Torroni. Attention, please! a critical review of neural attention models in natural language processing. *arXiv preprint arXiv:1902.02181*, 2019.

[37] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[38] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[39] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[40] Dominik Scherer, Andreas Müller, and Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In *International conference on artificial neural networks*, pages 92–101. Springer, 2010.

[41] Akhilesh Gotmare, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. A closer look at deep learning heuristics: Learning rate restarts, warmup and distillation. *arXiv preprint arXiv:1810.13243*, 2018.

[42] Samuel L Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V Le. Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*, 2017.

[43] Lia Corrales. retinanet-examples, May 2020.

[44] Guangnan Wu, Zhiyi Pan, Peng Jiang, and Changhe Tu. Bi-directional attention for joint instance and semantic segmentation in point clouds. *arXiv preprint arXiv:2003.05420*, 2020.

A

Architecture parameters

Here disclosed are the architecture and optimization parameters used in this work. One can find the optimization parameters in Table A.1 and the network architectures in Table A.2.

Table A.1: The parameters used in this work related to the optimization of the network.

		Parameters	
Optimizer	Algorithm	Base learning rate	L2 regularization
	Adam	2×10^{-5}	0.01
Scheduler	Schedule	Warmup iterations	Epochs
	Half-cosine	1000	20

Table A.2: A listing of the MLP architectures used in this work consisting of the components linear layer (Lin), batch normalization (BN) and rectified linear unit (ReLU). The operations field specifies how many message passing operations are used in the architecture.

	Multilayer perceptron pipeline →						
Graph encoder							
MLP_f	BN	Lin(512)	BN	ReLU	Lin(512)	BN	ReLU
MLP_g	BN	Lin(512)	BN	ReLU	Lin(512)	BN	ReLU
Operations	8						
Attention encoder							
MLP_f	BN	Lin(448)	BN	ReLU	Lin(512)	BN	ReLU
MLP_g	BN	Lin(448)	BN	ReLU	Lin(512)	BN	ReLU
MLP_k		Lin(448)					
MLP_q		Lin(448)					
MLP_v		Lin(448)					
Operations	8						
Mixed encoder							
MLP_f	BN	Lin(512)	BN	ReLU	Lin(512)	BN	ReLU
MLP_g	BN	Lin(512)	BN	ReLU	Lin(512)	BN	ReLU
Operations	6						
MLP_f	BN	Lin(512)	BN	ReLU	Lin(512)	BN	ReLU
MLP_g	BN	Lin(512)	BN	ReLU	Lin(512)	BN	ReLU
MLP_k		Lin(512)					
MLP_q		Lin(512)					
MLP_v		Lin(512)					
Operations	2						

	Multilayer perceptron pipeline →						
PointNet encoder							
0.2 m	Lin(64)	BN	ReLU	Lin(64)	BN	RelU	Lin(64)
	ReLU	BN					
1 m	Lin(128)	BN	ReLU	Lin(128)	BN	RelU	Lin(256)
	ReLU	BN					
Global	Lin(256)	BN	ReLU	Lin(512)	BN	RelU	Lin(512)
	ReLU	BN					
Decoder							
Class MLP	Lin(512)	BN	ReLU	Lin(512)	BN	RelU	Lin(10)
	ReLU	BN					
Orientation MLP	Lin(512)	BN	ReLU	Lin(512)	BN	RelU	Lin(8)
	ReLU	BN					
Regression MLP	Lin(512)	BN	ReLU	Lin(512)	BN	RelU	Lin(5)
	ReLU	BN					