TÉCNICO LISBOA

Autonomous vehicle perception using a driving simulator: the capabilities of artificial LiDAR data

João Miguel Falcão Espadinha

Thesis to obtain the Master of Science Degree in

Aerospace Engineering

Supervisors: M.S. Ivan Lebedev
Prof. Alexandre José Malheiro Bernardino

Examination Committee

Chairperson: Prof. Paulo Jorge Coelho Ramalho Oliveira
Supervisor: Prof. Alexandre José Malheiro Bernardino
Member of the Committee: Prof. Cristiano Premebida

January 2021

Acknowledgments

First, I want to thank my family, namely to Avó Jesuína, Avô Manuel, Irmã Ana and Mãe Teresa who have always provided support, guidance and have given me the opportunity to live an enriching and happy life.

Secondly, I want to thank M.S. Ivan Lebedev and Dr. Luka Lukic, who crucially helped me during my work, with valuable feedback and guidance and professor Alexandre Bernardino for his input and availability.

Finally, I want to thank Instituto Superior Técnico and Mobis Parts Europe N.V for allowing this opportunity to come true. Adding to that, I want to thank the later one for providing all necessary software and supervising conditions.

Abstract

In autonomous driving, object detection and semantic segmentation are critical tasks since it is required to not only detect and track objects on the road, but also to have a deep understanding of the surroundings. Since annotating 3D point clouds is a high resource and time-consuming task, artificial data is seen as an upcoming resource that could solve these issues. In this work we propose a study of the capabilities and potential of artificial LiDAR data for object detection and semantic segmentation, using deep learning methods. For 3D vehicle perception challenges, training deep neural networks in big, sparse and unordered point clouds has shown to be a hard task, in part due to the lack of publicly available data. We used simulator CARLA to generate our data, and studied in depth ways of mitigating the differences between artificial and real world data. We modelled both the noise from real world point clouds, and the missed reflections (that we called point dropout) that occur in real world data collection. We also explored potential benefits of using pre-trained models on artificial data when fine tuning with all or a fraction of the available real world data. We found clear benefits when using artificial data to pre-train a network, which allowed us to use a reduced amount of real world data, and boosting the performance of our models.

Keywords: Object detection, semantic segmentation, artificial LiDAR data, deep learning.

Contents

List of Tables

List of Figures

Acronyms

AOS	Average Orientation Similarity.
AP	Average Precision.
BB	Bounding Boxes.
BEV	Birds Eye View.
BS	Batch Size.
CNN	Convolutional Neural Network.
DNN	Deep Neural Network.
FCN	Fully Convolutional Network.
FN	False Negative.
FP	False Positive.
GUI	Graphical User Interface.
IoU	Intersection over Union.
LR	Learning Rate.
mIoU	Mean Intersection over Union.
NMS	Non Maximum Suppression.
RPN	Region Proposal Network.
TP	True Positive.

Chapter 1

Introduction

In this work, we use data from simulator CARLA [1] to learn the tasks of object detection and semantic segmentation for autonomous driving purposes. We employed state-of-the-art Deep Neural Networks (DNNs), which receive point clouds as input and provide, in one case, the detection boxes and, in the other, the semantic class of each point.

Since we want to apply the obtained knowledge from the DNNs in real world scenarios, an expected problem that we face is the difference between data generated artificially and gathered from real world scenarios. To mitigate this issue we study point cloud properties present in real world scenarios, model and apply them to the artificial point cloud. In addition, we use strategies to provide as much variability in background and foreground agents as the simulator allows us, during artificial scene generation and selection.

After training the network with artificial data and validating on real world data, we assess the potential use of artificial data to pre-train a network that is, afterwards, fine tuned with real world data. We explore some transfer learning techniques to study how the performance can be affected and how much we can reduce the amount of real world data used.

1.1 Motivation

The overall goal of autonomous driving is allowing the vehicle to make the most informed decision it can, based on the circumstances it faces. Thus having a deep understanding of its surroundings and the behaviour of the dynamic agents is of the utmost importance. Object detection and semantic segmentation are some of the key tasks used to help solving those challenges, and the ones we will address in this work.

Significant progress has been made on 2D object detection and semantic segmentation but, for 3D vehicle perception challenges, training DNNs in big, sparse and unordered point clouds has shown to be a more challenging task. As a result, there is a gap between performances for 2D and 3D perception, which could be narrowed with additional publicly available data. This lack of data can be explained by the intense process of 3D point cloud annotation, which requires time and resources. Besides that, the available data usually has big class imbalances, and is prone to human error when labeling.

As a result, artificial data can be an important asset to tackle the previous problems. Simulators usually provide highly precise annotations and, once we set up the data capturing process, annotated data is generated with less time, cost and human resources. In addition, during the data creation process we can adjust class statistics to our needs and create scenes which are hard to replicate in real world. In short, enabling the use of artificial data for the described tasks could lead to a leap in performance of autonomous driving systems.

1.2 Objectives and contributions

The main objective of this work is to study the capabilities and potential of artificial LiDAR data for object detection and semantic segmentation. We want to assess the utility of artificial data using deep learning methods, and apply the gathered knowledge to real world scenarios. The main challenges that we have to overcome in order to achieve our goal are: (1) reducing the dissimilarity between artificial and real data, and (2) discovering how we can benefit from using a pre-trained model with artificial data in comparison with training the DNN from scratch.

The performance of a DNN depends greatly on the data that we use to train it. Using artificial data to learn a certain task, and test it on a different domain (real world) can be a difficult exercise, due to the differences between both sets. This way, our contribution to solve challenge (1) can be summed up as:

- We created a noise model and a point dropout model to simulate real world effects. These models were applied on our artificial data, and we empirically studied their contributions on the performance of object detection and semantic segmentation.

With regards to challenge (2), we decided to explore other potential uses of artificial data. We performed multiple experiments to determine how we could benefit from using a pre-trained model on artificial data, from which we brought the following contributions:

- We evaluated two different transfer learning strategies, by breaking down the DNNs into two different sections: encoding and decoding. We decided whether we should freeze the weights from the encoding section or not when fine-tuning our DNNs and, as a result, we assessed the similarity of encoded features with artificial and real data.
- After determining the best transfer learning strategy to use, we performed a set of experiments to find out how the use of a pre-trained model on artificial data could affect performance and amount of real data required to fine tune our DNNs.

1.3 Outline

This thesis is organized as follows. In Chapter 2 we start with a topic overview, and then review some important object detection and semantic segmentation works, as well as some related works that focus on the same issues that we tackle. In Chapter 3, we define the methodology that was used during the development of this work, give detailed information regarding the data creation and selection process and the pre-processing models that were created. In the end, we describe the DNNs and training strategies used. In Chapter 4, we describe our experimental setup, presenting the simulator that was chosen and describing the implementation details and evaluation metrics used to compare our models. In Chapter 5, we explain the hypotheses that we want to investigate, the experiments designed to test those hypotheses, as well as the corresponding results. Finally, in Chapter 6, we present the most important conclusions from this thesis and propose the most promising directions for future work.

Chapter 2

State-of-the-art

In this chapter, we will present the state-of-the-art methods that were important for our work. We will start by giving a brief topic overview (Section 2.1) where we frame our work in the context of autonomous driving. In Section 2.2, we will explain some of the important work that was developed for object detection and semantic segmentation, using images and LiDAR data. Finally, in Section 2.3, we will present some works that have similar objectives to ours: create simulated data and assess its performance on autonomous driving tasks.

2.1 Topic overview

LiDAR (LIght Detection And Ranging) is an active remote sensing system which provides precise depth measurements of the surroundings. It fires pulses of laser lights and measures the amount of time they take to return. With the return time we can calculate the distance between the LiDAR and the target and, by repeating this process in quick succession, a 3D representation of the environment is created.

LiDAR data points have a total of four elements: the 3D coordinates (x,y,z) and an intensity value, which represents the energy that each pulse returns with. In addition, the most common representation of this type of data is the point cloud, because of its simplicity. LiDAR has several applications, for example in spaceflight, robotics and, in our case, autonomous driving. Unlike cameras, it provides precise depth information, which makes it a good choice for environmental sensing.

Initial works used point clouds in computer vision pipelines but, with the rise in popularity of deep learning, they were soon replaced by DNNs. Currently, DNNs are the state-of-the-art methods on 3D autonomous vehicle perception challenges and have proven to be a powerfull tool when quality data is provided. The performance of these methods highly rely on the data that is used and, for this reason, publicly available datasets started to emerge. Some of the available datasets with annotated LiDAR data are H3D [2], nuScenes [3] and the one used in this work, KITTI [4].

As mentioned before, the creation of these types of datasets requires a lot of time and resources, reasons why recent studies have tried to create and employ artificial data for autonomous driving purposes. This data has been generated, in some cases, in video games like *Grand Theft Auto V* and, in other cases, using simulators specifically designed for this purpose, like SYNTHIA and CARLA [1]. With our work, we want to use artificial data to provide insight on how we can benefit from it, using deep learning methods.

2.2 Background work

In this section, we will give a brief explanation of the main objectives of object detection and semantic segmentation, as well as an overview of some important works for each task. We will briefly go trough the evolution of deep learning methods for each task, starting with its implementation for images and finishing with some approaches for 3D point clouds.

2.2.1 Object detection

Object detection has the purpose of identifying and detecting objects of a certain class. This identification can be performed in images or, in our case, in a 3D representation like a point cloud.

In early works, it was established that CNNs were the preferred architectures when performing object detection on images. These networks are composed of three main types of layers: Convolution Layer, Pooling Layer and Fully Connected Layer. The convolution layers are responsible for transforming the input in several feature maps, one for each kernel that is applied. The pooling layers are usually used between convolutions and are responsible for reducing the spacial size of the representations. This way, computational cost is reduced, since less parameters have to be trained. Finally, the simple fully connected layers are usually the last layer of the network, and they just connect each neuron of one layer to each neuron of the next layer.

Later, some works [5, 6] suggested a two-stage approach. In a first stage, a Region Proposal Network (RPN) is used to extract several regions of interest from a scene, which are regions where the objects may be. Then, in a second stage, these proposals are verified and classified by a second network, like a CNN. Even though this approach requires more inference time, it has shown to outperform the one-stage approaches, which do not use a RPN.

For object detection in point clouds, it is logical to follow the image detection line of thought and apply 3D convolutional networks, as it is performed in [7]. However, this approach requires a lot of computational resources and time. To tackle this problem, newer methods transform the point cloud into 2D representations and then apply 2D convolutions. In some cases the point cloud is projected into the ground plane, in others it is projected into the image plane and, for the network used in this work, PointPillars [8], a pseudo-image is created. An explanation of the structure of this network is presented in Section 3.4.1.

2.2.2 Semantic Segmentation

Semantic segmentation is the process of linking each element of a certain representation to a class label. If we perform this task on an image, the elements are pixels, if we perform on a point cloud, each element is a three dimensional point.

Regarding semantic segmentation in images, a commonly used architecture is the Fully Convolutional Network (FCN). This approach, proposed by Long et al. [9], follows the architecture of a CNN, with the difference that the fully connected layer is substituted by convolutional layers. This way, an output map with smaller size than the image (due to the previous pooling layers) is obtained, which is then converted to the original image size using backwards convolution. Following studies, such as [10] build upon FCN, proposing an encoder-decoder architecture. The encoder block uses a CNN and, the important modification, the decoder, restores the lower dimension features to the original image resolution. In the end, the feature representation is mapped to the desired number of semantic classes.

As for object detection, CNN based approaches for semantic segmentation in point clouds use a 2D or a 3D approach. As before, projection based methods are some of the leading approaches due to their smaller computational needs and the good results that they provide. In this work, we use RangeNet++, which uses a spherical projection approach to create a 2D representation. The 2D representation is then fed into an encoder-decoder network. This architecture is further explained in Section 3.4.1.

2.3 Related work

Artificial data for autonomous driving has recently started to become a subject of study. As it was mentioned before, finding a solution to the high-cost, time consuming and laborious task of labeling data is a hot topic that has the potential of helping to solve the problem of data availability, in autonomous driving.

Although there are some works that explore the creation and use of artificial images [11, 12] using deep learning methods, both for object detection and semantic segmentation, there is a lack of work regarding artificial 3D scene generation. In other words, LiDAR point clouds have not yet been the most prioritized object of study in regards to artificial data capabilities. There are, however, some works that explore its capabilities for object detection and instance segmentation, and compare the performance obtained when using KITTI [4] data or artificial data, as we do.

Fang et al. [13] proposed a slightly different approach than ours. Instead of creating fully artificially generated scenes, Fang uses scans from real world scenarios and augments them with synthetic obstacles, like cars, pedestrians, etc. In his approach, first, 3D scans of road scenes are taken, executing multiple scans for each scene, in order to obtain dense point clouds. This way, we can, later, obtain sparser point clouds that simulate the ones created by LiDAR sensors. Secondly, they developed a data-driven approach which extracts, from real traffic scenes, the distribution of obstacles' positions, orientations and shapes. This learned distribution is then used to augment the real world scenarios with the computer generated objects. Finally, for each generated scene, the final point cloud is rendered using the Velodyne HDL-64E S3 specifications. Figure 2.1 illustrates the whole proposed approach.

Results showed that retaining the complexity and realism of real world scenarios brings added value in comparison with fully simulated data. However, this approach is a middle-ground between training DNNs with only real or simulated data. Even tough we can generate different scenes from the same

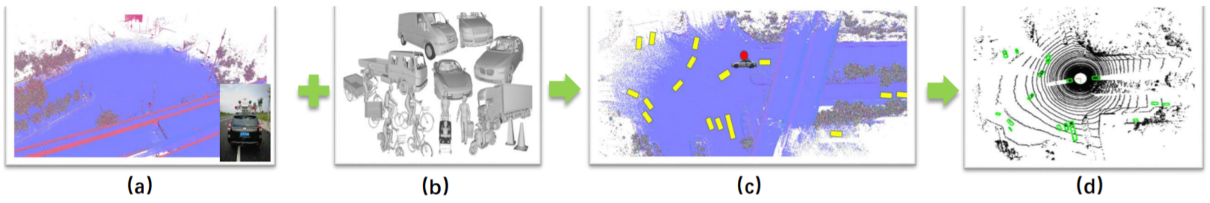


Figure 2.1: Proposed LiDAR point cloud simulation in [13]. (a) represents the static background construction. (b) shows artificially generated obstacles. (c) represents an example of object placement in the real world background. (d) is an example of the final simulated LiDAR point cloud with ground truth 3D bounding boxes.[1]

backgrounds, if we want a large sample size of scenarios, human annotation is still needed.

PreSIL is the most similar work to ours and was developed by Hurl et al. [14]. In that work, fully artificial data is created using *Grand Theft Auto V* (GTA V), and used to detect pedestrians. The capturing process in PreSIL consists in driving autonomously in each scenario, while retrieving object information at certain time stamps. GTA V provides information regarding object position, dimensions, heading, type and vehicle model. The point cloud is created using depth images, as we do in our work and which is explained in Section 3.2.1. In addition, a simplistic noise model was applied to the artificial point cloud, in order to simulate this property from real world data. There are, however, some limitations in this approach, like the fact that GTA V does not provide reflectivity information or the fact that full 360^{o} scans are not provided.

Hurl et al. [14] concluded that artificial data can boost performance when using a pre-trained model on this data. This is especially true when detecting objects with low class representability in real world datasets, like pedestrians. It can be explained by the fact that datasets created using simulators can be more easily adapted to the task's needs, for example including more pedestrians.

[1]Image taken from [13]

Chapter 3

Methodology

The work developed consists in a study of the capabilities and potential of artificial LiDAR data for the tasks of object detection and semantic segmentation, using deep learning in autonomous driving scenarios. In this chapter we will, at first, give an overview of the work developed and, later, explain in more detail each step of its development.

Before entering the chapters' content, it is important to, first, define some important terms that will be used ahead, in order to avoid misunderstandings. First, when we mention *hyperparameters* we are referring to all the parameters of a network's architecture or to the training process that are changed manually or are not updated in the training loop (*e.g.* number of epochs, batch size, learning rate). In the other hand, *parameters* or *weights* are parameters that are adjusted in the training loop, trough backpropagation (*e.g.* filters weights). Finally, we consider a *model* an instance of the network (a set of its weights) that was trained with certain training and validation datasets and a certain set of *hyperparameters*.

3.1 Pipeline Overview

We can divide our work pipeline in four major sections: data creation, data pre-processing, learning process and the evaluation step, as it is shown in Figure 3.1. The main contributions of this thesis are in the pre-processing and evaluation phases.

First, in data generation (Section 3.2), we will explain how the artificial data was created. We will not only mention how the point cloud was generated and how the ground truth was annotated for each task, but also the thought behind the generation and selection of the scenes. Then in the pre-processing section (Section 3.3), we will describe the real world effects we simulated, and how we modelled them. Finally, in the learning section (Section 3.4) we will introduce and explain the networks' architectures that were chosen for object detection and semantic segmentation, as well as explain the datasets used during training. Finally, in the same section, we will mention the training strategies that were adopted in order to fine tune a model pre-trained on a dataset from a different domain.

After all these stages, the evaluation step is initiated, in which all the models we created during

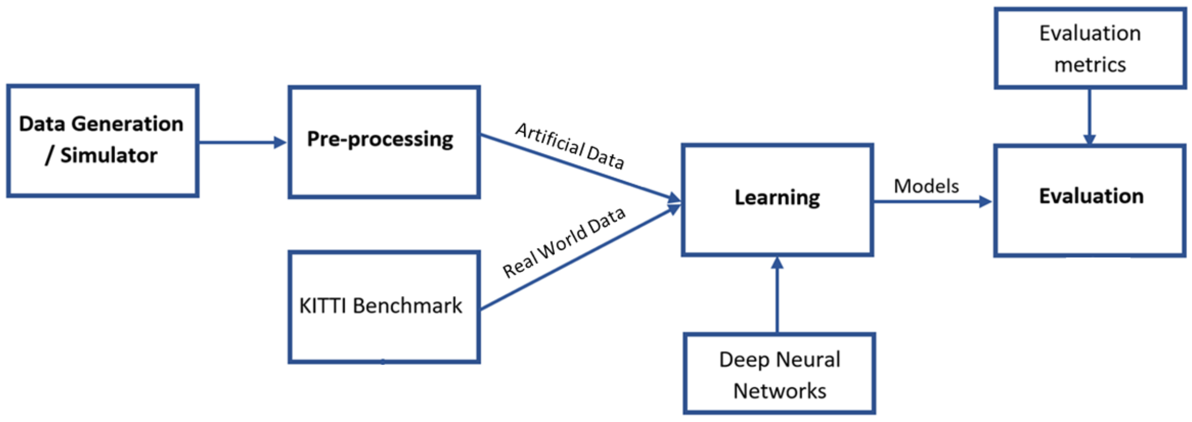


Figure 3.1: Pipeline of the work that was developed.

training were evaluated with the metrics defined in Chapter 4.

3.2 Data Generation

Since artificial data was the main focus of this work, Data Generation had a key influence in the overall outcome of this thesis. Seeing that there was no public dataset with data from the selected simulator that suited our needs, we had to produce the artificial data used in this work. We had to generate a point cloud from the sensors available in the simulator, as well as annotate all the useful ground truth, for each task. We gave special attention to the variability and richness of our data (explained in Subsection 3.2.3) as well as to the careful selection of the data frames that fulfilled the needs of object detection and semantic segmentation (Subsection 3.2.4).

3.2.1 Point Cloud Generation

During the data creation process, we did not have direct access to a LiDAR sensor in our simulator and, because of that, we needed to create one. A typical LiDAR outputs the values of 3D positions (x, y, z) of each observed point, in the LiDAR coordinate frame, as well as a value of the energy that each point returned to the LiDAR (E_{ret}). To create our point cloud, we had to use other resources that were available in the simulator, in this case, a depth camera. Therefore, in our sensor setup (explained in Section 4.1.1), the depth camera occupied the position that was designated for the LiDAR. For the point cloud creation process we used the depth image and the depth camera parameters. In the depth image we have the information about real world depth (d) of each pixel in the image, while the depth camera parameters, represented by its intrinsic matrix, allow us to map the 3D world scene into the image plane.

To create our point could, first, we define all the unit vectors ($r_{\theta\phi}$) with the directions of the simulated shots from the LiDAR. These vectors were created based on the values of the horizontal and vertical field of view and resolutions, from the specifications of the LiDAR that was being simulated (Velodyne HDL-64E, as mentioned in Section 4.1.1). With respect to the vertical and horizontal resolutions, after comparing the projections of real point clouds from Velodyne HDL-64E and simulated point clouds with the original resolution values from that scanner, a mismatch was observed. The real world point clouds were visibly sparser and, as a consequence, we changed the original resolution values from 0.08º for horizontal and 0.4 for vertical to, respectively, 0.125º and 0.485º.

The computation of each 3D shot is represented by the following expressions:

$$(u, v) = P(r_{\theta\phi}) \,, \tag{3.1a}$$

$$d_f = BilinearInterpolation(d_1, d_2, d_3, d_4) \,, \tag{3.1b}$$

$$(x, y, z) = d_f \cdot r_{\theta\phi} \,. \tag{3.1c}$$

As represented in (3.1a), after the unit vectors were created, we projected them in the image, using the intrinsic matrix, and obtained the pixels coordinates (u, v). With this values, we can calculate the depth for each point that is located within the direction of the original vector (d_f). When we had a point with image coordinates located between pixels, we obtained the depth value by bilinear interpolation of the depth values from the four pixels around it ($d_1, ..., d_4$). Finally, we multiplied each unit vector by the corresponding depth of the generated point, in order to get the 3D position of each point of the point cloud.

Regarding the returned energy value, it was calculated by the following equations:

$$E_{ret} = E_{emit} \times R_{rel} \times R_{ia} \times R_{atm} \,, \tag{3.2a}$$

$$R_{ia} = (1 - \cos\alpha)^{0.5} \,, \tag{3.2b}$$

$$R_{atm} = exp(-\sigma_{air} \times D) \,. \tag{3.2c}$$

where E_{emit} is the energy of the original laser pulse, R_{rel} is the reflectivity of the surface material, R_{ia} denotes the reflection rate with respect to the laser incident angle and R_{atm} is the air attenuation rate, since each laser beam is absorbed and reflected when travelling in the air. R_{rel} is obtained from a prediction of the material a certain point has, based on the class that it belong to. In R_{atm}, reprsented by (3.2c), σ_{air} is a constant equal to 0.004, and D is the distance from the LiDAR center to the target. We can observe, in Figure 3.2, an example of a depth image and the corresponding point cloud.

3.2.2 Ground Truth

When generating our ground truth, we had to follow a predefined format that our DNNs required. In this case, the format used for object detection followed the one used in the 3D object detection dataset from KITTI Benchmark [15], while for semantic segmentation we followed the one used in Semantic KITTI [16].

With respect to object detection, all the ground truth was written in two *.txt* files. The first one had the information of the calibration matrices which, in our case, were:

- **Intrinsic matrix** - mapping from camera coordinate system to image coordinate system;

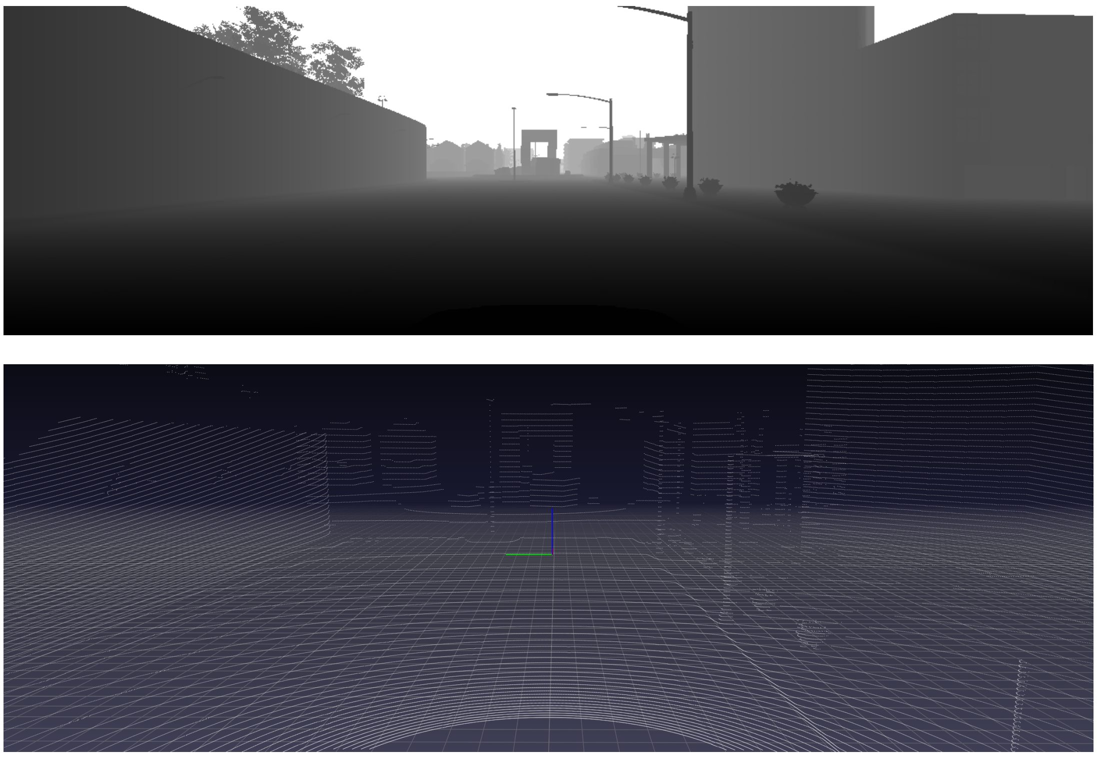


Figure 3.2: Depth image provided by the simulator (top) and the corresponding point cloud (bottom).

- **LiDAR-camera extrinsic matrix** - mapping from LiDAR coordinate system to camera coordinate system.

The second file had information about all the objects in the scene. For each object, we stored information regarding:

- **Object class**;
- **Truncation level** - When the object leaves image boundaries. Its value ranges from 0 (not truncated) to 1 (fully truncated);
- **Occlusion level** - When an object is covered by another object. Assumes the values: 0 (fully visible), 1 (partly occluded), 2 (largely occluded), 3 (unknown);
- **Angle alpha**[1] - Observation angle of the object (allocentric orientation). Its value is in the range [-π, π];
- **Location** - Position in camera coordinates;
- **Dimensions** - Height, length and width of he object;
- **2D bounding box** - Bounding box in the RGB image. Contains the left, right, top and bottom pixels coordinates;
- **Rotation_y**[1] - Angle of rotation around the vertical axis (egocentric orientation). Its value is in the range [-π, π].

[1]For more detailed explanation, see `https://towardsdatascience.comorientation-estimation-in-monocular-3d-object-detection-f850ace91411`

For this task we generated objects from the following classes: “car”, “pedestrian”, “cyclist”, “truck” and “motorcycle”, from which “car” was the class to be detected. In comparison with KITTI dataset, we gave additional relevance to objects that appear in the road, reason why we added the “motorcycle” class. On the other hand, there were other classes in KITTI, such as “person_sitting”, “van” and “tram” that we did not include, since our simulator did not provide them. It is important to mention that, even though it was not used during training, an image for each scan was also created. It allowed us to visualize the ground truth with respect to the camera coordinate frame and, this way, assess the correctness of the generated data. Every described file was named with the number of the frame it belonged to, in order to link all data from the same scene.

Regarding semantic segmentation ground truth, a *.label* file was created for each point cloud, which contained the semantic class that each point belonged to. Each semantic class is represented by an integer and, as it was mentioned in Section 4.1, the simulator provides 25 different semantic classes. However, there were two main problems that we had to tackle before being able to use the semantic segmentation data. First, from the 28 semantic classes from KITTI and the 25 from CARLA, we observed that some classes from different instances were assigned with the same integer value. To solve this issue, we mapped all CARLA classes to the integer value from the closest KITTI class. Secondly, it was observed that, not only some classes from CARLA were more broad than the ones from KITTI, but also that some classes from KITTI were not represented in CARLA. To solve this issue we started by isolating the nineteen semantic classes used in RangeNet++ [17]. After that, the previously detected specific classes were mapped to a broader class, and the ones not represented in CARLA were excluded, as it is represented in Figure 3.3. As we can see, in CARLA there was no distinction between a “person”, a

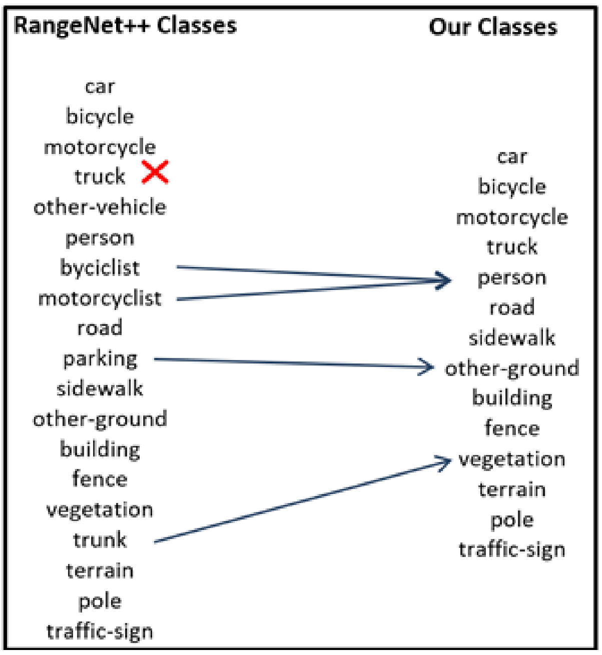


Figure 3.3: Mapping from the initial semantic classes used in RangeNet++ to the ones used in our experiments.

“motorcyclist” and a “bicyclist” and, as a consequence, all of them ended up being considered a “person”.

The same happened with the trunk of a tree that, in the simulator, was not differentiated from the rest of the tree and, therefore, it was also considered as “vegetation”. Regarding the parking space, unlike KITTI, the simulator assigned it to the general class “other-ground”. Finally, the class “other-vehicle” was removed since it referred to vehicles that were not present in our simulator, like bus and tram. In the end, we reduced the initial number of classes used in RangeNet++ from nineteen to fourteen.

3.2.3 Variability in the Datasets

Since we have the objective of studying the capabilities of LiDAR artificial data for real world applications, there is a clear gap between domains (real and artificial) that could undermine the performance of our models. One way to attenuate this gap was explained by Tobin et al. [18], where it was shown that, if the variability in simulation is high enough, models trained in simulation should generalize to the real world with no additional training.

This variability can be accessed in terms variety of backgrounds, agents (vehicles’ models, pedestrians, etc), as well as in terms of agents’ orientations, relative position to our car, and occlusion and truncation levels. However, we were dependent on the resources that were provided by the simulator, in this case the background and the foreground actors, which defined how far we could apply this principle. With respect to the background, we used all the eight different ones available in our simulator, which included scenarios from rural areas as well as cities and highways. Regarding the foreground, we had access to fifteen different pedestrians and twenty-seven different vehicles, which included nineteen different cars, three different bicycles, three different motorcycles and two different trucks.

Our data collection process consisted in driving around a car with our sensor suite, in each town/area, while capturing consecutive frames. The movement of our car and the other dynamic agents was also simulated. During the creation of our scenarios we tried to balance two major elements: fidelity in relation to the real world and diversity of agents and its attributes (orientation, occlusion levels, etc). Since we did not have total control over our agents, the fact that we could not control their movement meant that we frequently captured frames with a small amount of objects. To circumvent this problem, we spawned an additional amount of static objects (cars, motorcycles, etc) in our background, ending up with scenes as the ones in Figure 3.4. These objects were spawned with random orientation, to further contribute to the variability of our dataset. Additionally, we spawned new objects every 100 frames and deleted the existing ones every 500 frames. This was necessary not only to diversify our scenes, but also to ensure that, in some cases, our car would not get blocked by other objects on the road for too long, which would lead to datasets with a big number of repeated frames.

In the end, we ended up with sets of frames from eight different scenarios filled with agents with a variety of orientations, in which the relative position of that same object varies as the car moves. Adding to that, their truncation and occlusion levels also differ as we move through the scenes.

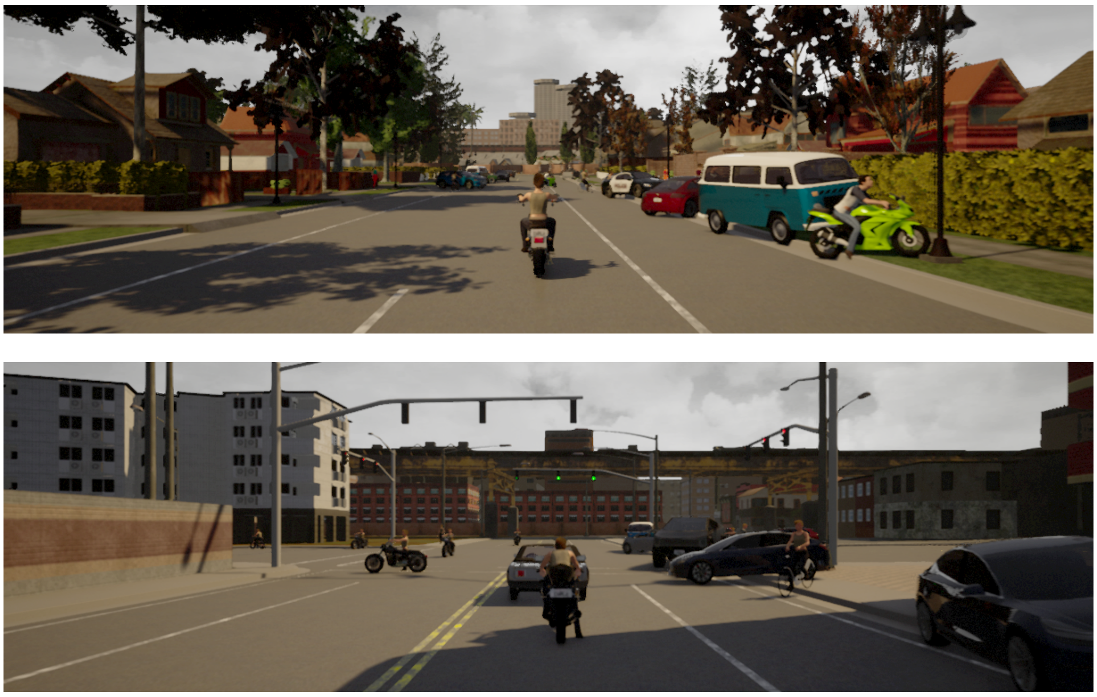


Figure 3.4: Examples of scenes from our simulator where, besides the moving agents, we can see several static agents that provide added variety and richness to our datasets.

3.2.4 Data Selection

When all the simulations were finished, an additional step of data selection was needed in order to ensure that we were training our networks with the highest quality data available. We divided this step into two main stages: a first selection of the most diverse datasets acquired and an additional frame selection performed to balance the diversity and difficulty of the "car" class in the dataset.

The data that we got from the simulator was organized in folders with 4000 sequential frames, and each folder had frames from a single scenario. This way, after creating several data folders with all the different scenarios, a selection of the ones with the highest quality data was required. To fulfill our objective, it was important to, first, define the attributes that characterize high quality data. For us, it was important to assure that our data had a high quantity and variability of agents from all the different classes. To achieve this goal we chose the data folders with a combination of:

1. Highest amount and variety of agents in the scenes;
2. Least amount of repeated frames (repeated frames appeared when the car got blocked by other objects in the scene).

After this first stage, we ended up with a set of folders with sequential frames from our simulator. As it will be mentioned in section 3.4.2, we wanted to keep the same structure between the datasets composed of artificial data and the ones from KITTI [4, 16]. This way, since the semantic segmentation datasets from Semantic KITTI were made of several different sequences of frames, after this first selection, we ended up with our final artificial data for this task.

Regarding object detection, the KITTI dataset was not organized in sequential frames. It was composed of random frames from different scenarios, which means that an additional selection process on the artificial data was needed, after the previous screening. Since in this work we are mostly interested in detecting cars, this additional stage consisted in a selection of each frame based on the amount of cars in it. To each car can also be applied a difficulty score, depending on its level of occlusion, level of truncation and bounding box height. It is important to be considered because the amount of cars in each frame was computed after applying the difficulty filtering that was used in PointPillars [8]. There are three different difficulties (“Easy”, “Moderate” and “Hard”), each one with specific object attributes. In PointPillars, if the attributes for the difficulty mode “Hard” were not met, the ground truth of that object would not be used for training. This way, by performing the same filtering technique we were able to better control the true statistics of the car class. The selection of the frames to use was situational and, this way, it will be explained in the Experiment and Results section, where we describe in more detail the training sets that were created for the different experiments.

3.3 Pre-processing

Before starting our experiments with the generated data, an additional pre-processing stage was added. Our main goal in this step was to study real world effects, in order to model them and apply them on the generated artificial data. The real world effects that we studied were the noise in the point clouds and the missing points due to missed reflections, which we call point dropout. This way, we tried to make artificial data look more similar to real data and, as a result, close the gap between artificial and real domains.

For this stage, a Graphical User Interface (GUI) was created in order to allow us to take the points from certain regions of the point cloud. For each frame, it showed the image and, on top of it, all the projected points from the point cloud. This way, to obtain the points from a certain region, we just needed to select that same area in the image and, as a result, we would get the correspondent points from the point cloud. This GUI was used to support the creation of the noise and point dropout models.

3.3.1 Noise Model

With our noise model, we aim at characterizing the statistics of the noise vector at each point in the point cloud, considering its distance to the origin of the LiDAR reference frame and the angle between its direction and the forward axis (x axis) of the same reference frame. Our model considers that the noise samples follow a normal distribution with a mean value of 0 and a standard deviation that was obtained from the KITTI data (real world data). Since the LiDAR scanner that we incorporated was trying to simulate the one used in KITTI (as it will be explained in Subsection 4.1.1), using real data from this same source allowed us to obtain a noise model that was adjusted to this specific sensor.

In order to compute the standard deviation, we created a function that depends on the characteristics mentioned before. We started by collecting points from different flat surfaces obtained in the KITTI dataset, with the help of the GUI. Then, for each surface, using Least Squares, we fitted a plane to the corresponding points in order to obtain the simulated noiseless surface. After that, for each point, we computed the distance to the origin of the LiDAR reference frame, the angle with the forward axis and the perpendicular distance to the estimated plane. Finally, we made a quadratic regression of the perpendicular distances as a function of distance to origin and angle with x axis. In the end, we obtained the function showed in Figure 3.5.

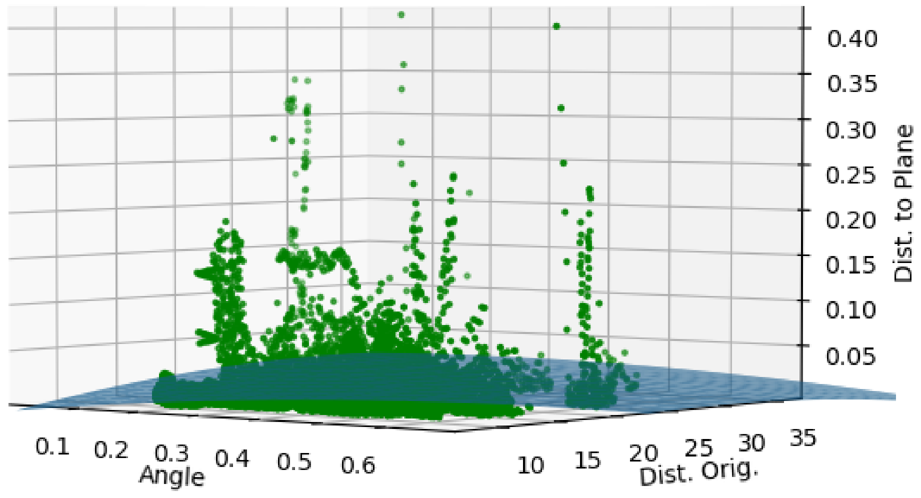


Figure 3.5: Function with the values of the average perpendicular distance to a plane (by approximation, our standard deviation) for each combination of angle with the forward axis and distance to the origin.

This average perpendicular distance to a plane is, by approximation, our standard deviation, and its expression is given by:

$$\sigma = -0.050196\,\alpha^2 - 4.582916 \times 10^{-5}\,d^2 - 0.001986\,d\,\alpha + 0.097530\,\alpha + 0.003070\,d - 0.031166 \tag{3.3}$$

where σ is our standard deviation, α is the angle between the direction of a point and the forward axis and d is the distance to the origin of the LiDAR reference frame.

In order to apply noise to each point, based on its parameters, we extracted the value of the standard deviation from the previous expression. Then, with all the parameters from our normal distribution, we sampled a value from this distribution, and used its absolute value as the magnitude of the noise vector. Finally, we randomly selected the direction of the noise vector and added its coordinates to the selected point. This way, we obtained noisy point clouds, adjusted to the sensor that was used to collect the real data that we trained and validated with.

3.3.2 Point Dropout Model

Regarding our point dropout model, we had the objective of creating a model that would give us a decision of removal or non-removal for each point of the point cloud, considering the same attributes as before. Our model consists in a Bernoulli distribution that gives us the previously mentioned decision, from a value of the probability of removing a certain point, that was obtained from KITTI point clouds.

For this task, a small addition to our GUI was made. In this case, besides projecting the points from the captured point cloud, we also projected the points that we should have if no missed reflections occurred. This addition was necessary, since we wanted to compare the amount of points that our point cloud had with the amount that it should have.

In order to compute the probability of removal, we created a function that depends on the characteristics of each point that were mentioned before. To obtain this function, we started by selecting several small surfaces, from which the theoretical amount of points the surface should have and the amount of points from the actual point cloud were recorded. Then, for each surface, we computed the distance between its mean point and the origin of the LiDAR reference frame, the angle between the mean point and the forward axis and the probability of a point being removed, given by the equation 3.4:

$$p_r = \frac{np_t - np_a}{np_t} \tag{3.4}$$

where p_r is our probability value, np_t is the theoretical number of points and np_a is the actual number of points in a certain surface. Finally, we applied a quadratic regression to the probability values that we computed before. In the end, we obtained a function that can estimate the values of the average probability of removal for each combination of distance to the origin and angle with the x axis (as shown in Figure 3.6). The probability of removal that we got is, by approximation, the probability of a certain point being removed that we will use in our Bernoulli distribution.

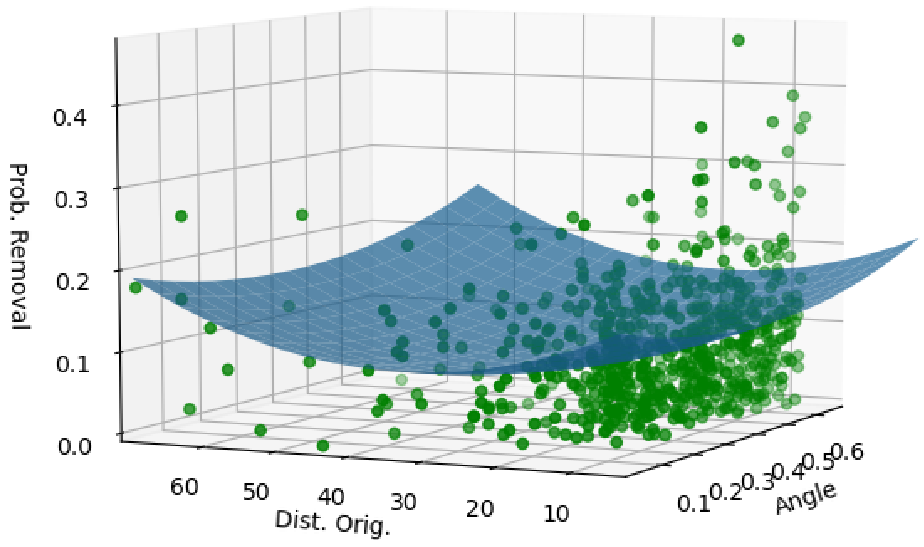


Figure 3.6: Function with the values of the probability of removal for each combination of angle with the forward axis and distance to the origin.

Therefore, the probability of a certain point being removed is given by the equation:

$$p = 0.186136\,\alpha^2 + 6.984331 \times 10^{-5}\,d^2 + 1.648670 \times 10^{-4}\ d\,\alpha - 0.164589\,\alpha + 0.004652\,d - 0.173883 \tag{3.5}$$

In order to simulate the missed reflections from our simulated LiDAR shots, we started by computing the distance of each point to origin and the angle with the forward axis. Then, we used these values and (3.5) to obtain our probability of removal for each point. Finally, we used a Bernoulli distribution with the previous probability value to obtain our final decision of removal or non removal.

3.4 Learning

With all the data ready, we had to define how to use it in order to fulfill the main objective of this thesis: assess the potential of artificial data in real world scenarios, and obtain trustworthy results in both tasks, object detection and semantic segmentation. Thus, in this section, we will present the three main contributing factors: the networks' architectures used for each task (Subsection 3.4.1), the artificial and real datasets used in training and validation (Subsection 3.4.2) and, finally, the transfer learning process used when initializing our networks with pre-trained models (Subsection 3.4.3).

3.4.1 Deep Neural Networks' Architectures

In this section, a brief explanation of the deep neural networks used for our tasks will be performed. As mentioned before, we will describe the architectures of PointPillars [8], used in object detection, and RangeNet++ [17], used in semantic segmentation. We decided to choose these DNNs because they are the state-of-the-art architectures for each respective task. Besides that, these architectures outperformed the former best ones when using the real world data from the same sources that we used.

Object Detection

PointPillars [8] is a network that receives point clouds as input, and outputs oriented 3D boxes. It can be divided in three main components: Pillar Feature Net, Backbone and Detection Head, as illustrated in Figure 3.7.

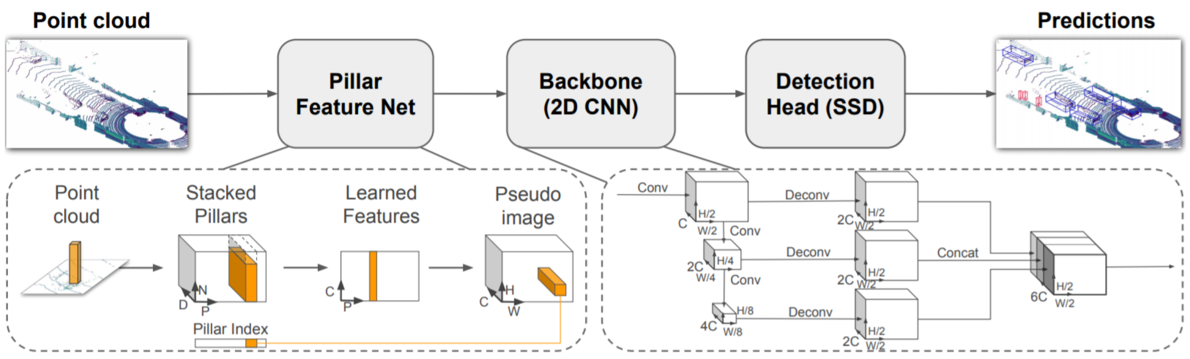


Figure 3.7: PointPillars' architecture overview[2].

The first component, pillar feature net, is the novel encoder that was introduced by the authors of [8]. It was created with the objective of converting the point cloud into a pseudo-image, in order to enable 2D

[2]Image taken from [8]

convolutions, which increases considerably the speed in relation with 3D convolutions. As a first step, the point cloud is represented as a grid map in the xy plane (in LiDAR coordinate frame), and a pillar is created for each grid cell containing points. In order to tackle the sparsity of the point clouds, a limit is imposed on the number of non-empty pillars per sample and on the number of points per pillar, which leads to a compact representation of the raw points. Then, the point in each pillar is used as the input for a simplified version of PointNet [19], to convert the raw point information into point features. The obtained representation goes through a max operation over the feature dimensions. A vector with the maximal feature values is then mapped to the original pillar location, creating our final pseudo-image.

The pseudo-image is then passed to the backbone, which is similar to the one used in VoxelNet[20]. This stage consists in three downsampling and upsampling blocks and, in the end, the upsampled features are concatenated and passed to the next component, the detection head. This stage follows the Single Shot Detector [21] setup, where ground truth and anchor boxes are matched using 2D Intersection over Union (IoU). At the end of this stage, the detection boxes are obtained, each one with a score value that indicates the confidence in each detection.

Semantic Segmentation

RangeNet++ is a projection-base 2D CNN that uses a range image representation of each laser scan to perform semantic inference. It is divided in four steps: a transformation of the point cloud into a range image representation, a 2D fully convolutional semantic segmentation network, a transfer from 2D to 3D which recovers all the points from the original point cloud and, finally, an image based 3D post-processing to clean the point cloud from undesired inference artifacts. The architecture of this network is represented in Figure 3.8.

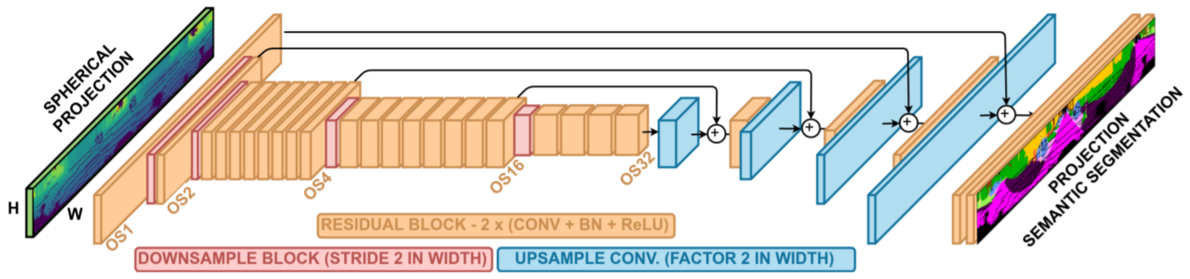

SPHERICAL PROJECTION
H
W
OS1
OS2
OS4
OS16
OS32
RESIDUAL BLOCK - 2 x (CONV + BN + ReLU)
DOWNSAMPLE BLOCK (STRIDE 2 IN WIDTH)
UPSAMPLE CONV. (FACTOR 2 IN WIDTH)
PROJECTION SEMANTIC SEGMENTATION

Figure 3.8: RangeNet++ architecture overview[3].

First, to obtain the spherical range image, each LiDAR point p_i is converted to spherical coordinates and, then, to image coordinates. After that, using the same indexes, information regarding original 3D coordinates, range and reflectance of each p_i is stored in a [5 x *h* x *w*] tensor, where *h* and *w* are the height and width of the desired range image. This new representation is passed to a 2D semantic segmentation CNN, which consists in an encoder-decoder architecture. These architectures are characterized for having an initial encoder section with considerable downsampling, where the features are encoded, followed by a decoder module which upsamples the encoded features to the original image

[3]Image taken from [17]

resolution. After this, we get to a classification head, that is composed by a set of [1x1] convolutions followed by a softmax layer, after which we get the semantic class of each point. With the output from the previous section, a reconstruction of the point cloud is performed. First, each point of the range image is indexed to the corresponding image coordinates. After that, a post-processing stage that consists in a k-Nearest-Neighbor search is applied, from which we get our final semantic label, for each 3D point, from the semantic information of the *k* closest points.

3.4.2 Training and Validation Datasets

For our experiments we used datasets from two different domains: real and artificial. Our artificial data was used purely for training, while the real world data was used for both training and validation purposes. We decided to use only real world data for validation to prevent the model from overfitting to the artificial data, which would jeopardise the main objective of our studies, which is to use our models in real world scenarios. In this section, we will just give an overview of the sources from which we got our data, not mentioning the specific sizes of all the datasets that were used. These details will be given in the Experiments and Results section. In Appendix A we show some of the scenes that make up the datasets that we will describe.

The artificial was created following the methodology described in Section 3.2. We also created specific training sets to assess the significance of our pre-processing models. To achieve this goal, we created a dataset with raw data from the simulator, as well as data with noisy point clouds, point dropout and with both noise and dropout. In our experiments, we also used training sets with different sizes reaching, for object detection, a maximum size of 40 000 frames and, for semantic segmentation, a size of 50 000 frames.

In relation to our real world datasets, we used data from KITTI Benchmark [15] for both object detection and semantic segmentation. Our training/validation distribution choice was based on most common practices from related works, in order to provide a more reliable comparison of results. For the first task, we got our data directly from the 3D object detection dataset, provided by KITTI Benchmark, which is divided into 7481 training instances and 7518 test instances. Since the test set did not provide ground truth annotations, we ended up only using the 7481 training instances that were provided. In the end, we divided the annotated subset into 3712 instances used only for training and 3769 instances used for validation. For semantic segmentation, we used data provided by Semantic KITTI [16]. It consists in 22 different sequences from the odometry dataset, provided by KITTI Benchmark, from which the first eleven (0 to 10) were annotated with 28 different semantic classes. As before, this dataset was divided in two subsets: a training set with annotated sequences (sequences 0 to 10) with a total of 23 201 instances and a test set without annotations (sequences 11 to 21) with a total of 20 351 instances. This way we only worked with the annotated sequences, from which we used sequence 8, with 4071 frames, as validation data and the remaining ones, with 19 130 frames, as training data.

3.4.3 Transfer Learning

During a training process, the main objective is to achieve a good generalization ability. This generalization ability can be obtained with the use of a validation set. However, in a situation where we want to learn a certain task in a domain (source domain), and apply it to a different one (target domain), just validating in a set of data of the target domain may not be enough. In our specific case, we want to use data from the artificial domain to learn how to perform object detection and semantic segmentation on the real domain. This way, we explored an additional strategy to help us tackle this problem: transfer learning. We wanted to use this approach to assess whether transferring the knowledge from a model trained with artificial data to a new model trained with all or a fraction of the real world data could give us an added value in our final performance.

As it was shown by Yosinski et al. [22], there are several training strategies that we could implement when using a pre-trained model. More specifically, we could initialize the whole network with the pre-trained weights, we could initialize randomly some of the weights or we could even freeze some of the network's layers, which means that the corresponding weights would not be updated after each step. The training strategy that we choose depends mainly on the similarity between the source and the target tasks, as well as on the similarity between the datasets that are used. Basically, the more different the tasks and datasets are, the less we benefit from using a pre-train model, *i.e.* we get better performance when training more layers from scratch (with random weights' initialization) or when using the pre-trained weights and fine tuning all layers. When the tasks and datasets are similar, a standard approach that is usually performed consists in transferring the weights from all layers, fine tune the classification layers and freeze the encoding ones, in order to use the previously extracted features.

For our specific case, the source and target tasks are the same and the datasets only differ in domain, reasons why we decided to not initialize layers with random weights. In principle, if the simulator was realistic enough, we would not need to fine tune the encoding layers, because the features from both domains should be, at least, almost the same. Since we do not know for sure how similar the features from both domains are, we decided to investigate two different approaches. In the first one, we followed the standard procedure and transferred all pre-trained weights, from which we froze the encoding layers and trained the remaining ones. This means that, for PointPillars network, we froze the pillar feature net and encoder layers, while for RangeNet++ we only froze the encoder. In the second approach, we also transferred all pre-trained weights but fine tuned the whole network. Performing experiments with both procedures not only allowed us to discover which approach is the best one, but also to assess the similarity between the features from real and artificial domains and, this way, draw conclusions regarding the simulator's realism.

Chapter 4

Experimental Setup

In this chapter, we will describe all the programs used and implementations that were applied in order to perform our experiments and obtain our numerical results.

We will start by presenting the simulator that was used to create our artificial data, as well as some of its capabilities. Then it is described the simulator setup used to record the data. Furthermore, we detail the implementation of the DNNs used for object detection and semantic segmentation, and the test sets used to compare our models. Finally, an explanation of the evaluation metrics used for both tasks will be presented.

4.1 Simulator

In order to create our artificial data, we used simulator CARLA [1] (Car Learning to Act). This is an open-source simulator[1] created to support development, training and validation of autonomous driving systems. Its easy access and customizability are some of the biggest benefits that make this simulator an attractive choice among developers.

CARLA is grounded on Unreal Engine, a state of the art for computer graphics framework, and uses the OpenDRIVE standard to define roads and urban settings. It has a client-server architecture where the client is able to control a chosen agent (in our case, a car) while the server simulates all remaining agents, as well as the world itself.

This simulator provides 25 different semantic classes, such as car, bicycle, road, building, etc, and global bounding box accessibility, meaning that we have access to the bounding boxes of all the elements in the scene. It also provides a variety of digital assets, such as urban layouts, buildings, vehicles and other objects that can be freely used. We can see, in Figure 4.1, some examples of scenarios and agents provided by CARLA. In addition, it provides flexibility in terms of sensor suites specifications and gives some control over static and dynamic actors, environmental conditions and more.

[1]CARLA simulator github repository `https://github.com/carla-simulator/carla`

Figure 4.1: Different scenarios from CARLA simulator.

4.1.1 Sensor Suite

Since our datasets from real world scenarios are from the KITTI Vision Benchmark [15], we decided to use their sensor setup, explained by Geiger et al. [4], as a reference for our own setup. We used points of reference in their setup to define the position of our own sensors, and defined our sensors' reference frames with the same convention. As illustrated in Figure 4.2, their sensor setup is composed of:

- 2 grey scale cameras - Point Grey Flea 2 (FL2-14S3M-C), plus 2 varifocal lenses - Edmund Optics NT59-917;
- 2 RGB cameras - Point Grey Flea 2 (FL2-14S3C-C), plus 2 varifocal lenses - Edmund Optics NT59-917;
- 1 LiDAR scanner - Velodyne HDL-64E;
- 1 inertial navigation system (GPS/IMU) - OXTS RT 3003;

From the previous list, we only used a RGB camera and a LiDAR scanner. In KITTI, a higher number of sensors was used to create datasets for more tasks such as odometry, stereo and optical flow.

The sensor setup we used in CARLA consists in a RGB camera and a LiDAR scanner identical to the ones used in the KITTI setup. We followed some of the the specifications of the original model, which has 120 meters of range and a horizontal and vertical field of view of, respectively, 360^o and 26.9^o. Regarding vertical and horizontal resolutions, we used different values from the original ones (as we explained in Subsection 3.2.1), more specifically 0.125^o for horizontal resolution and 0.485^o for vertical resolution. Regarding the RGB camera, it captures images with a size of 1226x370 pixels.

In our setup, the LiDAR is located 1.95 meters behind the car's front wheel axis, at a height of 1.73

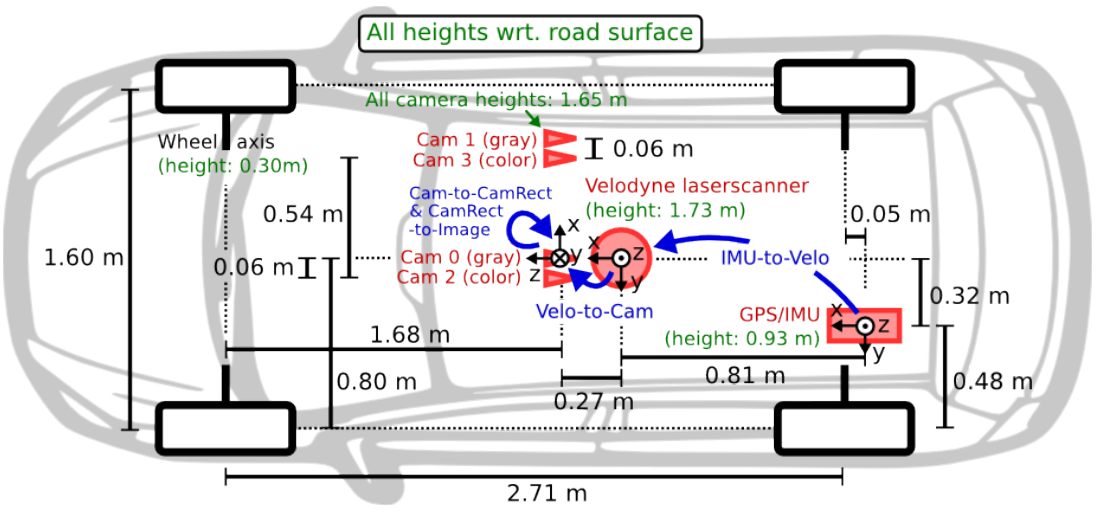


Figure 4.2: Sensor setup used in KITTI Vision Benchmark.[2]

meters and at the center of the car's width. Our RGB camera is located 1.68 meters behind the car's front wheel axis, at a height of 1.65 meters and at the same width as the previous sensor. From Figure 4.2, we can observe that our sensors have the same positions as, respectively, the LiDAR scanner and camera 0, in KITTI setup. Adding to that, we can also observe the camera and LiDAR coordinate systems, whose specifications are given in Table 4.1:

Origin	LiDAR center
x axis: viewing direction	forward: +
	backward: -
y axis: left direction	left: +
	right: -
z axis: up direction	up: +
	down: -

Origin	Camera optical center
x axis: right direction	right: +
	left: -
y axis: down direction	down: +
	up: -
z axis: viewing direction	forward: +
	backward: -

Table 4.1: Specifications of LiDAR coordinate system (left) and camera coordinate system (right).

4.2 Implementation of Deep Neural Networks

With respect to the learning process, we used the implementation of PointPillars [8] from traveller59[3] (for object detection) and of RangeNet++ [17] from Photogrammetry & Robotics Bonn[4] (for semantic segmentation).

We did not perform a through evaluation of hyperparameters, since our experimental studies focus on comparing models trained with different datasets and learning techniques. More precisely, the set of experiments that was performed involved changes in the size (number of frames) and in the domain (simulated vs real) of our training sets. For that reason, we want to keep a constant configuration of hyperparameters during each set of experiments, to ensure the validity of our comparisons. Nonetheless,

[2]Image taken from [4]
[3]Provided in `https://github.com/traveller59/second.pytorch`
[4]Provided in `https://github.com/PRBonn/lidar-bonnetal/tree/master/train/tasks/semantic`

there was one specific hyperparameter that we tuned before executing our set of experiments: learning rate. Adding to that, some additional implementations were also needed, as we will describe in this section.

4.2.1 PointPillars

To run the learning process of the PointPillars network, we used a machine with four NVIDIA Titan V GPUs, each one with 12 GB of memory space. During training, we only used one GPU at a time, since the original network implementation did not support multi GPU training. For that reason, we set our batch size as only 2 because, if we increased this number, our GPU would start lacking memory space.

For the PointPillars implementation, we tuned the learning rate parameters: initial learning rate and decay. We decided to use the idea introduced by Jastrzebski et al. [23] that Learning Rate (LR) to Batch Size (BS) ratio can influence the generalization ability of the network. It is stated that, for SGD, larger LR/BS ratios lead to better generalization. Although ADAM was the chosen optimizer, we used this idea as an initial guide, to know the direction we should follow in our experiments. We decided to, first, decrease the original decay value of 0.8 to 0.5 to have a smaller dip in learning rate values. Then, we diversified the initial learning rate values, following the previously explained thought. In the end, it was observed that ADAM performed the best with a configuration of 0.0001 as the initial learning rate and a decay value of 0.5, which were kept during all of our following set of experiments. We could observe that, unlike what happens with SGD, the ADAM optimizer does not follow the conjecture of Jastrzbeski [23] explained before, since we did not obtain better results with our validation set for higher initial learning rates.

As it was mentioned in Subsection 3.4.2, we are using a validation set with real world data, since we want to prevent overfitting on the the artificial domain. In order to fulfill that objective, an additional implementation had to be added to the original code. At first, the PointPillars [8] implementation did not have any kind of performance assessment on the validation set, *i.e.* no model was being saved based on its performance on an unseen dataset. Instead, the models from the last fourteen epochs were being saved and the last one was being used for evaluation. This was a problem for our goal of preventing overfitting on artificial data, since our validation set with data from the real domain would not be used to determine the best performing model. This way, we implemented an additional functionality that saves a model every time it exceeds the performance of the last best one, in the real domain validation set. This performance assessment was made after each epoch and the metric used to evaluate the model was Average Precision (AP) for overlap between 3D Bounding Boxes (BB), which is explained in Section 4.4. To decide for this metric, we chose to, first, ignore the ones that were evaluating performance on images, which left us with AP for overlap of 3D BB and for overlap in Birds Eye View (BEV). From these two, we chose the first one, since it uses more information from our detections in its calculations, *i.e.* AP for overlap in Birds Eye View (BEV) only uses information from the object position in the ground plane and from the length and width of the 3D bounding box. AP for overlap of 3D BB uses all three positional variables (x, y, z) and all bounding box dimensions in its calculations. For these reasons, we considered this metric the most complete and suitable.

Additionally, a function that allows us to freeze parameters of the network that we don't want to update was developed, in order to assist in our transfer learning studies. For our experiments, we only froze parameters from the pillar feature net and backbone (as will be explained in Chapter 5).

4.2.2 RangeNet++

Regarding RangeNet++, we ran the learning process in a machine with eight NVIDIA Tesla V100 SXM2 GPUs, each one with 32 GB of memory space. For semantic segmentation, the original implementation supported multi GPU training and, because of that, we used two different sets of GPU count and batch size combinations in our experiments (explained in Section 5.1):

- For experiments A and B: GPU count of 8 and batch size of 48;
- For experiment C: GPU count of 4 and batch size of 24.

These differences in configuration are a result of computational resources' availability. We also maintained the same configuration during each set of experiments, to assure the validity of the comparisons that were performed.

For semantic segmentation, and similarly to PointPillars, we also started with the experimental tuning of the learning rate. In this case, SGD was the chosen optimizer, which allowed us to draw direct conclusions between the similarity of our results and the results from [23]. We followed the same procedure and, in this case, we decreased the initial decay from 0.99 to 0.5. Since the original initial learning rate value was already quite large (0.001), we decided to evaluate the performance using smaller values for this hyperparameter. On the one hand, we observed that a larger initial learning rate value led to better performance, which follows the idea in [23]. On the other hand, we observed that the decrease in decay actually dropped the network's performance, which led us to the same configuration as the original one: initial learning rate of 0.001 and decay of 0.99.

Unlike in PointPillars, the original RangeNet++ implementation provided the capability to freeze specific regions of the network, during training, and allowed us to save the best performing model evaluated on the validation set. This way, no additional implementations were required.

4.3 Test Set

Since the main objective of this work is to study the capabilities of artificial data for real world application, it is clear that our models have to be evaluated and compared using data from real world scenarios. Due to this fact, data from the 3D object detection dataset, introduced in KITTI Vision Benchmark [15], and data from the Semantic KITTI [16] dataset will be used, for the corresponding tasks.

As it was mentioned in Subsection 3.4.2, the 3D object detection dataset from KITTI Vision Benchmark is divided into 3 712 training instances and 3 769 testing instances. However, only the training instances have been annotated with ground truth, which stops us from being able to use the test instances to evaluate our models. Due to this fact, we use our validation set to evaluate and compare the trained models, as it was performed in other works such as in PreSIL [14] and in the work developed by Fang et al. [13].

Regarding the Semantic KITTI dataset, the same situation as the one described before happens. In this case 23 201 instances are provided for training and 20 351 instances for testing, in which the test instances don't provide semantic labels. It was observed tough, that in some works such as RangeNet++[17] and even in Semantic KITTI itself, the test instances were being used to evaluate the different models. We realized that Semantic KITTI provided an evaluation server where this dataset was labeled and could be used. Unfortunately, this server cannot be used to train and evaluate multiple models with the purpose of making feature choices and reporting the best results obtained, which is essentially the main goal of our experiments. This way, we decided to follow the same criteria as before and also use the validation set to evaluate and compare the obtained models.

4.4 Evaluation Metrics

With the objective of comparing the different models that were created during the training phase, a set of quantitative metrics were selected. This allows us to do a more in depth analysis of the stronger and weaker points of each model and, this way, get a better understanding about the impact of the changes that were made in the different experiments that were performed.

The explanation of the evaluation metrics that were used will be split in two, regarding each task that was studied: object detection and semantic segmentation. For each one of them, the official KITTI evaluation metrics were used since they are widely chosen for these tasks and, this way, it provides an opportunity for future benchmarking.

4.4.1 Object Detection

Before describing the evaluation metrics used in this task, it is important to define an important term that will be used: Intersection over Union (IoU). In object detection, IoU gives an understanding of how accurate the predicted bounding boxes are, with respect to the ground truth bounding boxes. It is a ratio between the intersection of both boxes by their union, and it is calculated by:

$$IoU = \frac{P_{bb} \cap GT_{bb}}{P_{bb} \cup GT_{bb}} \tag{4.1}$$

where P_{bb} represents the predicted bounding box and GT_{bb} represents the ground truth bounding box. In object detection, it is typically required an IoU of, at least, 70% between a predicted and a ground truth bounding box for that prediction to be considered as a True Positive (TP). Besides that, if multiple detections of a same object are counted, the one with higher IoU is considered a TP and axis aligned Non Maximum Suppression (NMS) is applied with an overlap threshold of 0.5 IoU.

Another characteristic in all the metrics used is the fact that they are divided into three different difficulty modes: “Easy”, “Moderate” and “Hard”. As described before, for each object in the ground truth, a difficulty mode is assigned, depending on the height of the bounding box and its level of truncation and occlusion. According to KITTI evaluation, difficulty levels of boxes are defined as:

- **Easy:** minimum bounding box height of 40 pixels, maximum occlusion level 0 (fully visible) and maximum truncation level of 0.15;
- **Moderate:** minimum bounding box height of 25 pixels, maximum occlusion level 1 (partly occluded), maximum truncation of 0.3;
- **Hard:** minimum bounding box height of 25 pixels, maximum occlusion level 2 (largely occluded), maximum truncation of 0.5.

This way, on each difficulty mode, only objects that are assigned with a difficulty equal or inferior to the one that is being evaluated are considered as ground truth, and the detections regarding objects from harder difficulties are not taken into account.

Finally, the evaluation metrics used for this task are:

1. Average Precision (AP), calculated for:
   (a) Overlap of 3D bounding boxes
   (b) Overlap of bounding boxes in Birds Eye View (BEV)
   (c) Overlap of 2D bounding boxes
2. Average Orientation Similarity (AOS)

Average Precision

To better understand this metric, described by Everingham et al. [24], it is useful to first define what precision and recall are. Precision is a metric that indicates how good a model is at predicting the correct labels. It is the percentage of detections that are correctly assigned, *i.e.* that are TPs, and it is calculated by the equation:

$$precision = \frac{tp}{tp + fp} \tag{4.2}$$

where tp represents the amount of TPs and fp the amount of FPs. On the other hand, recall represents the percentage of the ground truth that was correctly detected. Recall is given by the equation:

$$recall = \frac{tp}{tp + fn} \tag{4.3}$$

where fn represents the amount of False Negatives (FNs).

For each detection, a score that indicates the level of confidence in that detection is assigned. This way, to calculate the Average Precision, several scores thresholds are selected and, for each one, recall and precision are calculated. Then, a version of the measured precision/recall curve with precision monotonically decreasing is computed. We obtain this curve by setting the precision for recall r to the maximum precision obtained for any recall $\tilde{r} \geq r$. Finally, from this previous values, a precision value is chosen for each recall value in the set $r \in \{0.0, 0.1, ..., 0.9, 1.0\}$ and the average of all these values will be our Average Precision. It can be represented by the following equations:

$$AP = \frac{1}{11} \sum_{r \in \{0.0, 0.1, \ldots, 1.0\}} AP_r \tag{4.4}$$

where,

$$AP_r = \max_{\tilde{r} \geq r} p(\tilde{r}) \tag{4.5}$$

Average Orientation Similarity

The AOS metric was first introduced by Geiger et al. [15] and it assesses both 2D detection and 3D orientation. The 3D orientation is represented by the rotation angle around the y axis, in camera coordinates (see Section 4.1.1). This metric follows a similar procedure as in AP but, in this case, instead of calculating the precision for each score's threshold, we calculate the orientation similarity $s \in$[0, 1]. Adding to that, the same procedure of setting s as a monotonically decreasing function is done, as it was performed for precision, in AP. In the end, an average of the orientation similarities for recalls $r \in \{0.0, 0.1, ..., 0.9, 1.0\}$ is calculated. This metric can be calculated by the following equations:

$$AOS = \frac{1}{11} \sum_{r \in \{0.0, 0.1, \ldots, 1.0\}} \max_{\tilde{r} \geq r} s(\tilde{r}) \tag{4.6}$$

where,

$$s(r) = \frac{1}{|D(r)|} \sum_{i \in D(r)} \frac{1 + \cos \Delta_\theta^{(i)}}{2} \delta_i \tag{4.7}$$

and $D(r)$ represents the set of all 2D detections at recall r, $\Delta_\theta^{(i)}$ is the difference between the estimated and ground truth orientations for detection i, and δ_i is a variable that is used to penalize FPs: δ_i=1 if the detection i is assigned to a ground truth bounding box and δ_i=0 if it is not.

4.4.2 Semantic Segmentation

The evaluation metrics used for Semantic Segmentation are:

1. Intersection over Union (IoU)
2. Mean Intersection over Union (mIoU)

Intersection over Union

The IoU metric, also known as Jaccard Index, is calculated for each segmentation class and it indicates how accurately the points' class was predicted, given a certain ground truth. It is, for each class c, the division between the number of points presented in both predicted and ground truth masks by the number of all point presented in either the predicted or ground truth masks, and it can be represented by the equation:

$$IoU = \frac{tp}{tp + fp + fn} \tag{4.8}$$

where tp represents the amount of TPs, fp the amount of FPs and fn the amount of FNs.

Mean Intersection over Union

The mIoU is an average of all the Intersection over Unions, calculated for each semantic class, and it can be represented by the following equation:

$$mIoU = \frac{1}{C} \sum_{c=1}^{C} \frac{tp_c}{tp_c + fp_c + fn_c} \tag{4.9}$$

where C is the number of classes.

Chapter 5

Experiments and Results

In this chapter, we will describe all the experimental studies that were performed and the conclusions that we took from the obtained results. For each study, one or more hypotheses will be defined, as well as the set of experiments performed to verify the validity of these same hypotheses. With these experiments, we have the goal of evaluating, empirically, the effect of changes in certain properties of our training sets and the effect of different learning techniques. Then, with our numerical results, a more in depth analysis of the outcomes of our experiments will be performed. Finally, we will compare the similarities and differences between the conclusions we took for object detection and for semantic segmentation.

As we stated before, we use the architectures of PointPillars[8] and RangeNet++[17] for the tasks of object detection and semantic segmentation. In addition, we use artificially generated training sets from CARLA[1] simulator and real world datasets from KITTI Vision Benchmark[15] and Semantic KITTI[16].

5.1 Studies and Experiments

In this section, a brief explanation of all performed studies will be done. As we can see in table 5.1, for each study, we present the hypotheses that we want to test as well as a small description of the set of experiments that were performed. For this set of experiments, we used different training sets, that will be described ahead. Furthermore, the obtained models were evaluated and compared on the test sets mentioned in Section 4.3 using the evaluation metrics presented in Section 4.4.

During training, a model was saved every time it achieved a better performance on the validation set than the last best one. For object detection, the chosen metric to measure the model's performance was Average Precision for overlap of 3D BB and for semantic segmentation it was Mean Intersection over Union (explained in Section 4.2). This way, in all the mentioned studies, we compared the results from the best performing model, after testing it on the test sets mentioned in Section 4.3.

Study	Description	Tested Hypotheses
A	Train our networks with different pre-processing models applied on the artificial data.	Studying noise and point dropout properties of real data and apply them on artificial data could improve performance.
B	Train our networks with additional amounts of artificial data.	Increasing the size of the training set with data from the artificial domain could improve performance.
C	Divide the networks into 2 blocks: encoding and decoding. Then fine tune the whole network or just the later block, using real data.	Using transfer learning to train a pre-trained model on artificial data, with real data, could give us information on the similarity between encoded features in artificial and real data.
D	After transferring weights from a pre-trained model with artificial data, fine tune the network with successively smaller amounts of real data.	Using transfer learning to train a pre-trained model on artificial data, with all or a fraction of the real data, could give an added value in comparison with training with only real data.

Table 5.1: Description of the each study that was performed and the conclusion we want to find.

5.2 Object Detection

For all the following object detection studies, we evaluated our models on a set composed by unseen real world instances. As mentioned in Section 3.4.2, this set consists in 3769 instances from the 3D Object detection training set of KITTI Vision Benchmark[15]. For these experiments, we decided to choose the car as our object to detect since not only it is the most represented class in our real world dataset, but it is also the class of objects with the most variability in the used simulator.

5.2.1 Study A

As it was mentioned before, we performed Study A with the objective of verifying the impact that our pre-processing models (explained in Section 3.3) have in the performance of the network. For that reason, we created four different datasets, all of them with 4000 frames (similar size to the training set from real world data) and composed of artificial data. All four training sets have the same data frames, being the only difference between them in the pre-pocessing model or models that were applied. This way, our four different training sets have point clouds with either:

1. No pre-processing model applied (raw data);
2. Only noise;
3. Only point dropout;
4. Both noise and point dropout.

It is also important to mention that all 4000 frames were chosen using the frame selection process introduced in Section 3.2.4. To create this dataset, we generated 24 000 frames, from which we selected 4000 based on the amount of cars in each one. For this study, we decided to follow a similar distribution of cars per frame as it was observed in the real world dataset. The amount of cars per frame was counted after the “difficulty index” filtering was performed and, in the end, we ended up with an approximate distribution shown in Figure 5.1.

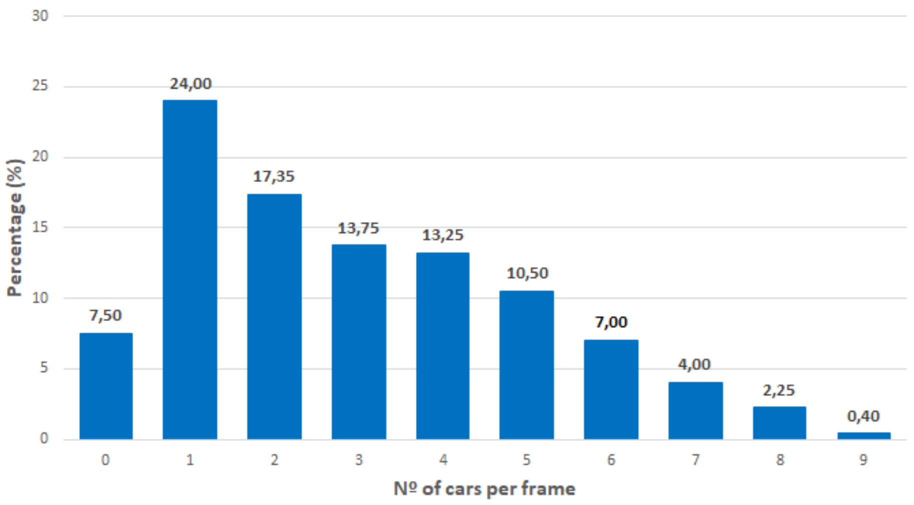


Figure 5.1: Distribution of cars per frame in the datasets used in Study A.

Results

After evaluating the described models, we obtained the results shown in Table 5.2. As we can observe, we get an overall better performance when we apply our pre-processing models on the point cloud. We can also conclude that we get the best performance when we just apply noise in our point cloud. In the end, we can verify that our pre-processing models are helping to close the gap between real and artificial domains, as it was intended to.

Dataset	AP of 3D detections			AP of detections in BEV			AP of 2D detections			AOS		
	Easy	Mod.	Hard	Easy	Mod.	Hard	Easy	Mod.	Hard	Easy	Mod.	Hard
Raw data	13.20	11.14	10.58	45.78	35.90	31.07	56.84	40.91	37.40	55.26	39.39	35.78
Point Drop.	14.26	11.49	11.25	40.84	33.57	30.98	58.98	45.17	**43.87**	54.85	41.46	**39.75**
Noise	**14.78**	**12.01**	**11.47**	**51.95**	**40.44**	**34.71**	**60.48**	**45.47**	42.56	**56.24**	**41.99**	39.24
Both	13.84	11.29	11.00	41.95	31.97	28.87	59.88	44.65	41.78	58.34	42.78	**39.75**

Table 5.2: Results after evaluating the models trained with datasets with different pre-processing implementations (Object Detection).

5.2.2 Study B

For Study B, we performed a set of experiments with the objective of inferring the impact that an increase in dataset size could have in the performance of the obtained models. This way, we created three additional training sets, all of them composed of artificial data and with a size of 10 000, 20 000 and 40 000 frames. Additionally, we decided to apply noise on the point clouds since it proved to be a positive addition that improved our network’s performance, as we concluded in the previous study.

In this case, the datasets were not fully created using the additional data selection process from the previous study. Besides the initial 4000 frames from the previous study that were included in all these datasets, the one with 10 000 frames included 2000 more selected frames, and the remaining ones included 2000 more selected frames than the later one. The remaining data samples were taken directly from the simulator, which means that our datasets would have a higher amount of frames with a smaller amount of cars (between 0 and 4). With the objective of trying to balance the distribution of cars per frame, all of the new selected frames had an amount of cars ranging from 5 to 9.

Results

For this study, we decided to include the results of the model trained with 4 000 frames of noisy data, in order to better compare the evolution of the dataset performances in relation with the best one from the last study. The results are shown in the Table 5.3.

Datas. Size	AP of 3D detections			AP of detections in BEV			AP of 2D detections			AOS		
	Easy	Mod.	Hard	Easy	Mod.	Hard	Easy	Mod.	Hard	Easy	Mod.	Hard
4 000	**14.78**	**12.01**	**11.47**	**51.95**	**40.44**	**34.71**	**60.48**	**45.47**	**42.56**	56.24	41.99	39.24
10 000	13.21	11.34	10.89	41.36	31.77	28.05	57.48	43.74	41.49	**56.34**	**42.63**	**40.22**
20 000	12.87	11.11	10.81	41.45	34.14	30.56	53.31	42.08	39.87	51.90	40.60	38.27
40 000	12.48	10.90	10.79	33.62	28.35	25.95	51.74	39.99	39.55	51.31	39.36	38.64

Table 5.3: Results after evaluating the models trained with datasets with different sizes (Object Detection).

As we can observe, the obtained results are unexpected since, not only we obtained an overall better performance on the model trained with the least amount of data, but also because we can observe a trend of worse performance as we increase the size of the dataset (most noticeable on AP for overlap of 3D and 2D bounding boxes).

These results could be explained by two major reasons. First, when we increased the size of the training sets, not much variability was being added to our data. In other words, our dataset with 4000 frames already had all the available agents as well as all the available scenarios that the simulator provided. This way, the data that we added was quite similar to the one that we already had, making this step not as impactful as we wanted. Another reason that could explain the observed behaviour is the data selection process, since the smaller dataset had fully selected data based on the amount of cars per frame. This extra step that, not only allowed us to choose higher quality data, but also helped us having a bigger control on the dataset statistics, could have been the main reason for good results.

5.2.3 Study C

After performing our previous studies, it was clear that training a neural network with only artificial data did not allow us to reach the peak performances that we could achieve with real world data. This way, we decided to carry out additional studies, where we investigated the potential of using a pre-trained model on artificial data and transfer its knowledge to the real domain. In other words, we performed new trainings with real world data in which we initialized the network with the weights from a pre-trained model on artificial data.

For this third study, we have the goal of verifying whether we are able to get similar performance when just fine tuning the non-encoding blocks of the network in comparison with fine tuning the whole network. This way, we can also get additional information regarding the similarity between the features encoded with real data and artificial data. To achieve our goal, we used the best model from the previous studies as our pre-trained model (trained with 4000 frames of artificial noisy data) and we used a set of real world instances from KITTI Benchmark (described in Section 3.4.2) with 3712 unused frames to train our network. We decided to create two different training sets: one with the whole 3712 and another one with just 2000 frames which we used to fine tune either the whole PointPillars network or just the decoder and single shot detector blocks (Pillar Feature Net and backbone weights were frozen).

Results

As we can observe in table 5.4, we got clearly better results when fine tuning the whole network. The fact that not updating our encoding blocks when training with real data led to worse results allows us to conclude that the encoded features, when training with artificial data, are different from the ones encoded with real world data. This way, there is a need to update the encoding blocks of the network when performing transfer learning, which we will do in the next study.

Model	AP of 3D detections			AP of detections in BEV			AP of 2D detections			AOS		
	Easy	Mod.	Hard	Easy	Mod.	Hard	Easy	Mod.	Hard	Easy	Mod.	Hard
FT 2000	**84.24**	**69.61**	**67.90**	**90.22**	**85.05**	**80.42**	**90.48**	**87.08**	**84.35**	**90.27**	**86.36**	**83.26**
Dec 2000	70.02	54.94	51.71	89.00	77.99	75.31	88.70	77.36	73.98	87.57	75.13	71.15
FT 3712	**85.34**	**75.58**	**69.63**	**90.14**	**86.51**	**81.23**	**90.50**	**88.18**	**86.53**	**90.34**	**87.57**	**85.54**
Dec 3712	71.62	55.85	53.49	89.13	78.15	75.61	88.79	77.73	74.76	87.75	75.71	72.18

Table 5.4: Results after fine tuning the whole network (FT) or just the decoding/detection section (Dec), using real world datasets (Object Detection).

5.2.4 Study D

Finally, in our last study, we wanted not only to find the peak performance of our network but also to investigate the evolution of the evaluation metrics when we successively decrease the size of the training set used for fine tuning. These comparisons were drawn between the cases where we fine tune the network, initialized with the weights from a pre-trained model on artificial data, and when we train the network from scratch. For this purpose, we used the same pre-trained model and the same real world dataset as we used in the previous study but, in this case, we further decreased the size of the real world training sets to 1000 frames and 500 frames.

Model	AP of 3D detections			AP of detections in BEV			AP of 2D detections			AOS		
	Easy	Mod.	Hard	Easy	Mod.	Hard	Easy	Mod.	Hard	Easy	Mod.	Hard
FT 3712	**85.34**	**75.58**	**69.63**	90.14	**86.51**	**81.23**	**90.50**	**88.18**	**86.53**	**90.34**	**87.57**	**85.54**
FT 2000	84.24	69.61	67.90	**90.22**	85.05	80.42	90.48	87.08	84.35	90.27	86.36	83.26
FT 1000	80.22	69.06	66.87	90.14	84.00	80.11	90.21	85.72	80.73	90.09	84.94	79.68
FT 500	78.64	67.39	65.27	90.17	84.77	79.99	90.20	80.96	79.73	90.01	80.15	78.49
R 3712	**85.58**	**75.27**	**69.66**	90.03	**86.99**	**84.17**	**90.71**	**88.66**	**87.11**	**90.61**	**88.13**	**86.08**
R 2000	81.04	70.55	67.25	**90.21**	84.01	80.65	90.51	86.91	82.42	90.25	86.16	81.36
R 1000	79.34	67.57	65.43	90.01	80.43	79.74	90.26	81.78	80.17	90.05	80.95	78.82
R 500	73.12	59.33	56.42	89.06	79.34	77.76	88.66	78.03	76.67	87.58	75.96	73.94

Table 5.5: Results after fine tuning the whole network (FT) or training it from scratch (R), using increasingly smaller real world datasets (Object Detection).

Results

We can observe the results that were obtained in Table 5.5. For this study, there is a clear trend in all the evaluation metrics that can be visualized for the specific case of the AP of 3D detections, in “Hard” difficulty mode (represented in Figure 5.2). As we can see, even though using a pre-trained model with artificial data does not result in a better top performance, it allows us to better stabilize the performance of our networks when we decrease the size of the training set. It is exemplified by the values of AP of 3D detections for “Hard” difficulty, where we achieve 65.27% when training with a dataset with 500 frames, just approximately minus 4% than when we use all 3712 frames of data. This way, we can see a clear contribution of artificial data for the task of object detection, specially when we have limited amount of real data.

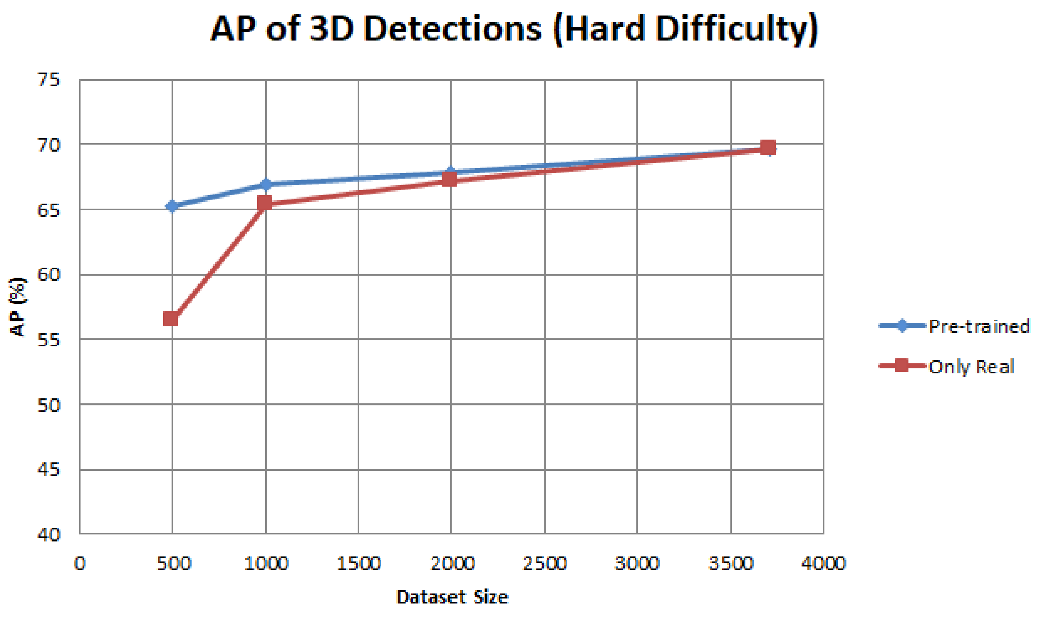


Figure 5.2: Average Precision for overlap of 3D bounding boxes, in “Hard” difficulty mode.

5.3 Semantic Segmentation

Regarding the semantic segmentation studies, we evaluated our models on a set also composed of unseen real world instances. As mentioned in Section 3.4.2, this set consists in 4060 instances from the Semantic KITTI [16] dataset and, for these studies, we classified fourteen different semantic classes (as described in Section 3.2.2).

5.3.1 Study A

As it was performed for the previous task, we also wanted to study the impact of the pre-processing step in semantic segmentation. In this case, we created four different datasets, all of them with 20 000 frames (approximately the same size as in KITTI training set) and data from the artificial domain. As before, our datasets were created with:

1. No pre-processing model applied (raw data);
2. Only noise;
3. Only point dropout;
4. Both noise and point dropout.

Unlike for object detection, instead of shuffling instances from different sequences, we used data directly from the simulator, composed of sequential frames, to simulate the real world dataset structure from Semantic KITTI. This way, the additional frame selection process was not performed in this task (as it was mentioned in Section 3.2.4).

Results

As we can observe in Table 5.6, the application of the pre-processing step in our artificial data increased the performance of the network. Although we cannot observe a uniform trend in all semantic classes, we can see that we get an overall better performance (accessed by the mIoU) when we apply both noise and point dropout in our point clouds. There is also a visibly larger contribution from the point dropout model in comparison with the noise model, as opposed to what is observed for object detection. In short, we can conclude that our pre-processing step helps closing the gap between real and artificial domains, as it was also observed in object detection.

Dataset	car	bicycle	motorcycle	truck	person	road	sidewalk	other-ground	building	fence	vegetation	terrain	pole	traffic-sign	mIoU
No model	23.3	0.3	2.5	**0.5**	9.3	54.6	28	**0.7**	16.3	**2.2**	0.6	5.1	**4.7**	1.2	10.7
Point Drop	**41.3**	**0.7**	3.5	0.4	8.8	**58.1**	27.7	0.6	14.1	1.7	1.1	8.5	3.5	2.2	**12.3**
Noise	27.4	**0.7**	**3.9**	0.3	8.3	55.8	27.4	0.3	**19.7**	1.6	0.4	6.6	4.0	**2.4**	11.3
Both	37.6	0.6	3.4	0.4	**10.3**	55.4	**29.7**	0.4	15.4	2.1	**1.4**	**9.3**	4.6	1.8	12.314

Table 5.6: Results after evaluating the models trained with datasets with different pre-processing implementations (Semantic Segmentation).

5.3.2 Study B

In the Study B we further assessed the impact that the increase in training set size had on the performance of the obtained models. For this purpose, we further increased the size of the training sets from artificial domain to 34 000 and 50 000 frames. As it was done for object detection, we used the conclusions taken from the previous study and applied noise and point dropout to our datasets.

Results

As we can observe in table 5.7, we get an overall better performance as we increase the size of the dataset. We can also observe that, as our dataset gets bigger, the contributions from additional data get smaller, what can be also explained by the small amount of variability that is added, as we increase the dataset size.

Datas. Size	car	bicycle	motorcycle	truck	person	road	sidewalk	other-ground	building	fence	vegetation	terrain	pole	traffic-sign	mIoU
20 000	37.6	0.6	3.4	**0.4**	10.3	55.4	29.7	0.4	15.4	2.1	1.4	**9.3**	4.6	**1.8**	12.314
34 000	**41.0**	0.8	3.4	0.3	**11.3**	**63.0**	31.7	0.7	13.1	1.9	1.1	8.3	**5.1**	1.7	13.1
50 000	38.1	**1.5**	**4.3**	**0.4**	10.6	62.0	**32.3**	**0.9**	**16.2**	**2.2**	**2.1**	8.9	4.8	**1.8**	**13.3**

Table 5.7: Results after evaluating the models trained with datasets with different sizes (Semantic Segmentation).

5.3.3 Study C

With all the studies with only artificial data finished, we performed study C to verify if we were able to get similar performance when just fine tuning the non-encoding blocks of the network in comparison with fine tuning the whole network, as it was done before. We also want to infer if the features that were encoded with real data and with artificial data are similar. For these experiments we used, as our pre-trained model, the one trained with 20 000 frames of artificial data with noise and point dropout applied. Additionally, real world data from Semantic KITTI (described in Section 3.4.2) was used, from which we created two different datasets: one with the whole 19 130 frames and another one with just 10 117 frames. As before, we used two different learning approaches, one where we fine tuned the whole network, and another one where we trained the non encoding blocks, in this case the decoder and segmentation head (weights from the encoder were frozen).

Results

From the results of table 5.8, we can draw the same conclusions that as we took in object detection. For this task, we can also see a decrease in performance when we freeze the encoding blocks, which leads us to conclude that there are differences in the encoded features when we train with real data and when we train with artificial data. This conclusion makes fine tuning the whole network the correct approach when using a pre-trained model with artificial data, as it will be performed in the next study.

Model	car	bicycle	motorcycle	truck	person	road	sidewalk	other-ground	building	fence	vegetation	terrain	pole	traffic-sign	mIoU
FT 10 117	**89.3**	**12.0**	**17.9**	**23.2**	**21.1**	**90.6**	**73.6**	**22.5**	**74.6**	**27.9**	**80.4**	**66.7**	**29.8**	**24.9**	**46.7**
Dec 10 117	84.4	2.7	7.1	9.3	16.8	85.6	63.7	13.8	62.3	12.4	66.1	62.4	20.3	15.5	37.3
FT 19 130	**90.2**	**15.4**	**29.3**	**25.8**	**32.8**	**92.2**	**77.4**	**34.6**	**79.1**	**43.2**	**81.8**	**69.9**	**33.0**	**25.9**	**52.22**
Dec 19 130	84.6	3.7	12.5	7.0	22.5	87.4	67.0	20.5	66.3	14.4	70.8	66.4	18.6	16.1	39.8

Table 5.8: Results after fine tuning the whole network (FT) or just the decoding/segmentation section (Dec), using real world datasets (Semantic Segmentation).

5.3.4 Study D

Finally, in this last study, we tried to find the peak performance of our network as well as its capacity to maintain a stable performance as we decrease the size of the real world training set. We compared the cases where we fine tuned the whole network, initialized with the weights from a pre-trained model, and where we trained the network from scratch. In these experiments we further decreased the amount of data of our training sets to 5 305 frames and 2 173 frames and used the same pre-trained model that was used in Study C.

Results

We can observe all the results that were obtained in Table 5.9 and, for an easier comprehension, we created a graph with the values of the mIoU, as we can see in Figure 5.3. Unlike what was observed in object detection, the use of a pre-trained model is not stabilizing the performance of our network when we decrease the size of the dataset. In the other hand, we can observe a general improvement in performance when using a pre-trained model, which makes artificial data a good addition when training a network for the task of semantic segmentation.

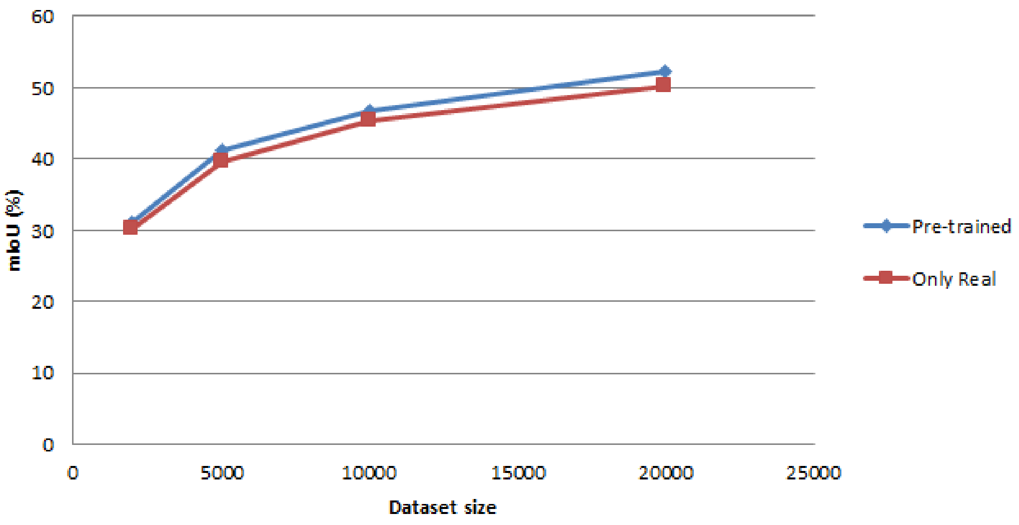


Figure 5.3: mIoU when training after initializing a pre-trained weights and when training from scratch, with an increasingly smaller dataset.

Model	car	bicycle	motorcycle	truck	person	road	sidewalk	other-ground	building	fence	vegetation	terrain	pole	traffic-sign	mIoU
FT 19 130	**90.2**	**15.4**	**29.3**	25.8	**32.8**	**92.2**	**77.4**	**34.6**	**79.1**	**43.2**	**81.8**	**69.9**	**33.0**	**25.9**	**52.22**
FT 10 117	89.3	12.0	17.9	23.2	21.1	90.6	73.6	22.5	74.6	27.9	80.4	66.7	29.8	24.9	46.7
FT 5 305	86.3	5.3	6.0	**26.8**	16.9	87.3	67.4	8.6	69.7	19.6	74.2	61.4	26.0	22.8	41.3
FT 2 173	75.0	1.3	1.2	5.4	7.1	82.3	52.4	2.2	51.5	7.7	63.9	49.7	19.0	17.6	31.2
R 19 130	**89.3**	**16.0**	**17.9**	**32.9**	**26.6**	**92.1**	**76.9**	**34.2**	**76.6**	**39.5**	**80.6**	**68.8**	**28.3**	**24.4**	**50.03**
R 10 117	87.1	17.5	10.8	29.8	16.0	90.4	73.0	21.9	72.4	24.2	78.4	65.9	23.9	23.3	45.3
R 5 305	84.9	9.6	2.7	21.1	9.8	87.2	67.4	9.7	67.6	17.4	73.1	60.2	22.5	23.0	39.7
R 2 173	73.7	0.7	0.1	11.9	1.9	78.9	49.9	2.1	52.1	9.7	65.0	47.7	14.2	15.6	30.2

Table 5.9: Results after fine tuning the whole network (FT) or training it from scratch (R), using increasingly smaller real world datasets (Semantic Segmentation).

5.4 Object Detection and Semantic Segmentation Comparison

Since the same studies were performed for both tasks, we can draw some conclusions about the differences and similarities between the results. We also have to take into consideration some factors that may influence the final results, such as the DNNs that were used in each case, as well as the different nature of each task.

In study A, although we concluded that the pre-processing step helped closing the gap between domains, we found a clear difference in the contribution of each model. For object detection, we got a much smaller contribution from the point dropout model, in comparison with what we got for semantic segmentation. One reason that may explain this behaviour is the difference between the DNNs' architecture, more specifically in the pillar feature net section from object detection. As we explained in Subsection 3.4.1, in this block of the network the point cloud is divided into a set of pillars and each one can have a maximum of 100 points. When this amount of points is exceeded, the ones in excess are removed randomly. This way, not only points can be removed even without using our point dropout model, but there could also exist some cases where, even after using our model, the amount of points could still exceed the maximum allowed. This way, we believe that this first block of PointPillars could influence the effectiveness of our point dropout model.

For study B, clear differences were observed in the results. While for object detection we found that the best model was the one trained with the least amount of data, for semantic segmentation we achieved higher performance as we increased the amount of frames. Some explanations were already given in the corresponding sections but, adding to those ones, we believe that the networks used for these tasks also play a role in these results. As we mentioned before, the number of agent models in CARLA is limited and, as a consequence, the variability in the object detection annotations becomes very limited, since weight, height and length are fixed values to each car model. In the semantic segmentation case, the RangeNet++ pre-processing step, in which all points are projected into a spherical image, provides added variability, since the projections of the point-wise annotations will differ significantly more as we move.

Regarding study C, as it was expected, no differences in results were observed. Since we used data from the same simulator, if there was a difference in the encoded features when using artificial or real world data, the same conclusions should be drawn in both tasks. We just concluded what we already expected: the simulator doesn't provide high enough realism to allow us to use the same encoded features when training with artificial and real world data.

Finally, in study D, we drew different conclusion about the potential use of a pre-trained model in artificial data. While for object detection we achieved a higher stability in performance when we decreased the size of the real world dataset, for semantic segmentation we only observed a slightl boost in performance when using a pre-trained model. It was previously expected that the same stability in performance would be achieved for semantic segmentation but, for this case, we do not have a clear explanation for the difference in behaviour.

Chapter 6

Conclusions

With this work we studied the capabilities and potential of artificial LiDAR data for object detection and semantic segmentation on real world scenarios. To do that we setup a data generation process on CARLA [1] that allowed us to create data with the highest quantity and variability of agents and scenarios possible. We also created a noise and point dropout models to help reducing the gap between real and artificial domains. Furthermore, we used the generated artificial data and real world data from KITTI [4, 16] to train and validate our models. In this work, we performed experiments to assess the highest performance we can achieve when training with artificial data, and to study the potential of using pre-trained models on artificial data to fine tune our DNNs.

In the end, we observed that simulators still cannot achieve a high enough degree of realism to replicate the complexity of the real world. However, we showed how we still can benefit from using artificial data when applying the right training strategies.

6.1 Achievements

This work not only provides an insight on the main challenges we have to face when using artificial data, but also a solid understanding of how we can tackle them and, in the end, benefit from this type of data. As we worked on these challenges, there were some main achievements that we accomplished and should be highlighted:

- the creation of a noise and point dropout models that proved to help closing the gap between real and artificial domains;
- the results obtained when training with only artificial data showed to be competitive when comparing with other related works [13, 14];
- the success achieved by proving how we can benefit from using a pre-trained model on artificial data (Sections 5.5 and 5.9). For object detection, we showed that this data allows us to use a fraction of the real data while maintaining competitive performance and, for semantic segmentation, it boosts the performance of our models.

6.2 Future Work

The most obvious path for the future work is to expand the capabilities of the simulator. While CARLA provides constant updates, there is a possibility for the user to create additional agents and, this way, increase variability of the scenes.

Secondly, there were some justifications for our results that could be further analyzed and fully confirmed. The first one regards the results from study B of object detection (Section 5.2.2) showing better performance when the network was trained with a smaller amount of data. As we explained, we believe that the additional frame selection process plays an important role on why we got better results with the smaller dataset. This way, we could perform further experiments but, this time, creating all the larger datasets with same criteria used for the first one. We would use the same data selection stages and the same overall car per frame statistics used for the smaller dataset. The other one regards the small impact of the point dropout model on object detection (Section 5.2.1) in relation with semantic segmentation (Section 5.3.1). For this one we would have to closely observe the pillar feature net block of PointPillars [8]. We could check how many points from raw point clouds (without pre-processing) this stage usually drops and compare it to the final amount of dropped points when using our point drop model. This way, we could get a better idea of how impactfull this model is for object detection.

Finally, an interesting study that could be performed consists in extending our study C to train our networks with more combinations of frozen layers. As it was explained in Section 3.4.3 we divided our networks into an encoding and decoding blocks, and performed a set of experiments with the weights from the encoding blocks frozen. We could perform additional trainings where we just freeze more initial layers from the encoding section, which extract more general features from the dataset. As a result, we could have a deeper understanding of where the encoded features in artificial and real world data start to differ, and potentially be able to train our DNNs with some frozen layers. This would allow us to reduce the computational cost while retaining the performance obtained when fine-tuning the whole network.

Bibliography

[1] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. CARLA: An Open Urban Driving Simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[2] A. Patil, S. Malla, H. Gang, and Y.-T. Chen. The h3d dataset for full-surround 3d multi-object detection and tracking in crowded urban scenes. IEEE International Conference on Robotics and Automation (ICRA), 2019.

[3] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11618–11628, 2020. doi: 10.1109/CVPR42600.2020.01164.

[4] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets Robotics: The KITTI Dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[5] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014. doi: 10.1109/CVPR.2014.81.

[6] R. Girshick. Fast r-cnn. In *International Conference on Computer Vision (ICCV)*, 2015.

[7] B. Li. 3d fully convolutional network for vehicle detection in point cloud. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1513–1518, 2017. doi: 10.1109/IROS.2017.8205955.

[8] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom. PointPillars: Fast Encoders for Object Detection From Point Clouds. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2019. doi: 10.1109/cvpr.2019.01298. URL `http://dx.doi.org/10.1109/CVPR.2019.01298`.

[9] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440, 2015. doi: 10.1109/CVPR.2015.7298965.

[10] V. Badrinarayanan, A. Kendall, and R. Cipolla. Segnet: A deep convolutional encoder-decoder

architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(12):2481–2495, 2017. doi: 10.1109/TPAMI.2016.2644615.

[11] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, and A. M. Lopez. The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3234–3243, 2016. doi: 10.1109/CVPR.2016.352.

[12] S. R. Richter, Z. Hayder, and V. Koltun. Playing for benchmarks. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2232–2241, 2017. doi: 10.1109/ICCV.2017.243.

[13] J. Fang, D. Zhou, F. Yan, Z. Tongtong, F. Zhang, Y. Ma, L. Wang, and R. Yang. Augmented lidar simulator for autonomous driving. *IEEE Robotics and Automation Letters*, PP:1–1, 01 2020. doi: 10.1109/LRA.2020.2969927.

[14] B. Hurl, K. Czarnecki, and S. Waslander. Precise synthetic image and lidar (presil) dataset for autonomous vehicle perception. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 2522–2529, 2019. doi: 10.1109/IVS.2019.8813809.

[15] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[16] J. Behley, M. Garbade, A. Milioto, J. Quenzel, S. Behnke, C. Stachniss, and J. Gall. SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences. In *Proc. of the IEEE/CVF International Conf. on Computer Vision (ICCV)*, 2019.

[17] A. Milioto, I. Vizzo, J. Behley, and C. Stachniss. RangeNet++: Fast and Accurate LiDAR Semantic Segmentation. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2019.

[18] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. *CoRR*, abs/1703.06907, 2017. URL `http://arxiv.org/abs/1703.06907`.

[19] R. Q. Charles, H. Su, M. Kaichun, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 77–85, 2017. doi: 10.1109/CVPR.2017.16.

[20] Y. Zhou and O. Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).

[21] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. *Lecture Notes in Computer Science*, page 21–37, 2016. ISSN 1611-3349. doi: 10.1007/978-3-319-46448-0_2. URL `http://dx.doi.org/10.1007/978-3-319-46448-0_2`.

[22] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27, pages 3320–3328. Curran Associates, Inc., 2014. URL `https://proceedings.neurips.cc/paper/2014/file/375c71349b295fbe2dcdca9206f20a06-Paper.pdf`.

[23] S. Jastrzebski, Z. Kenton, D. Arpit, N. Ballas, A. Fischer, Y. Bengio, and A. Storkey. *Width of Minima Reached by Stochastic Gradient Descent is Influenced by Learning Rate to Batch Size Ratio: 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4–7, 2018, Proceedings, Part III*, pages 392–402. 10 2018. ISBN 978-3-030-01423-0. doi: 10.1007/978-3-030-01424-7_39.

[24] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2011 (VOC2011) Results. http://www.pascal-network.org/challenges/VOC/voc2011/workshop/index.html.

Appendix A

Datasets' scenes

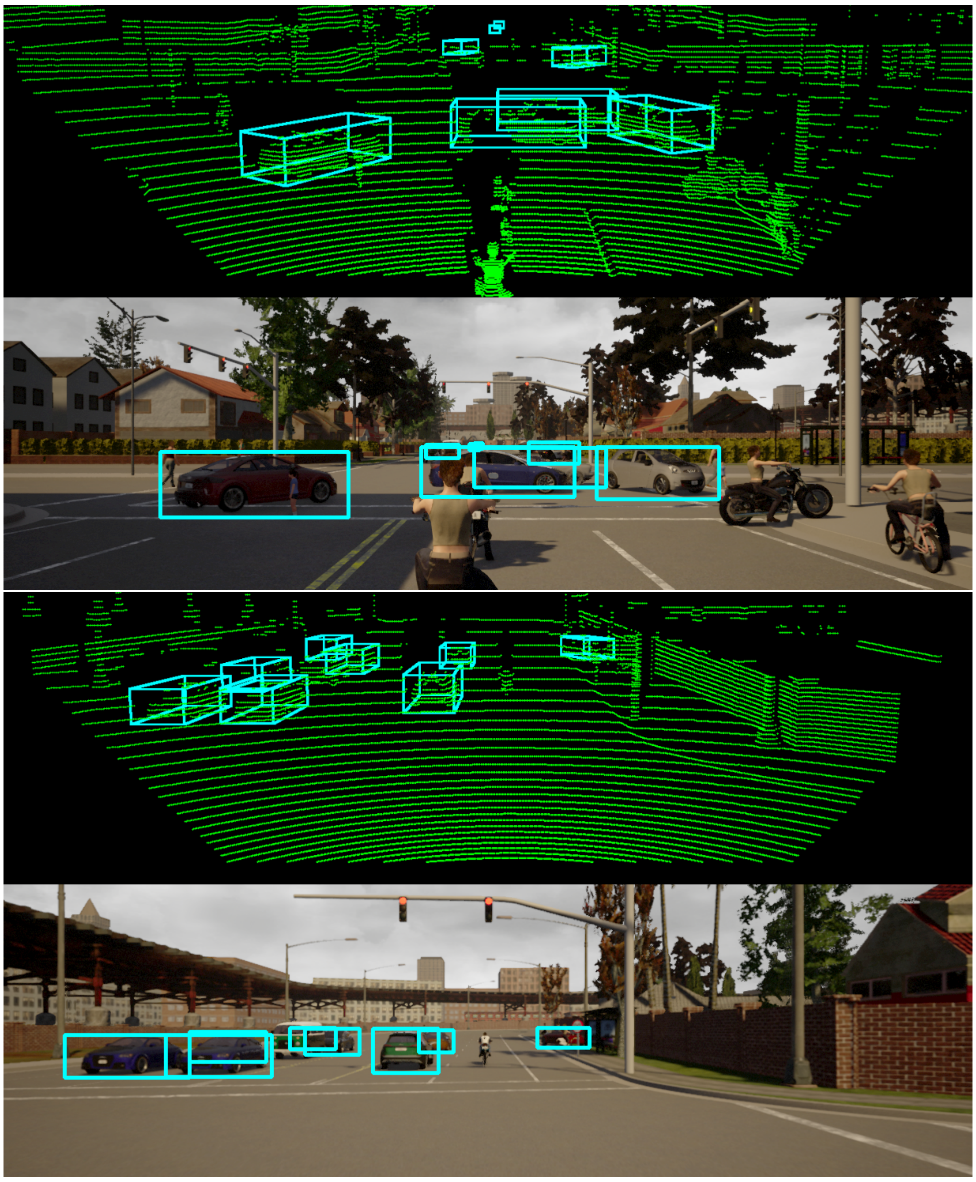


Figure A.1: Examples of scenes from the object detection artificial dataset, and the ground truth boxes (I).

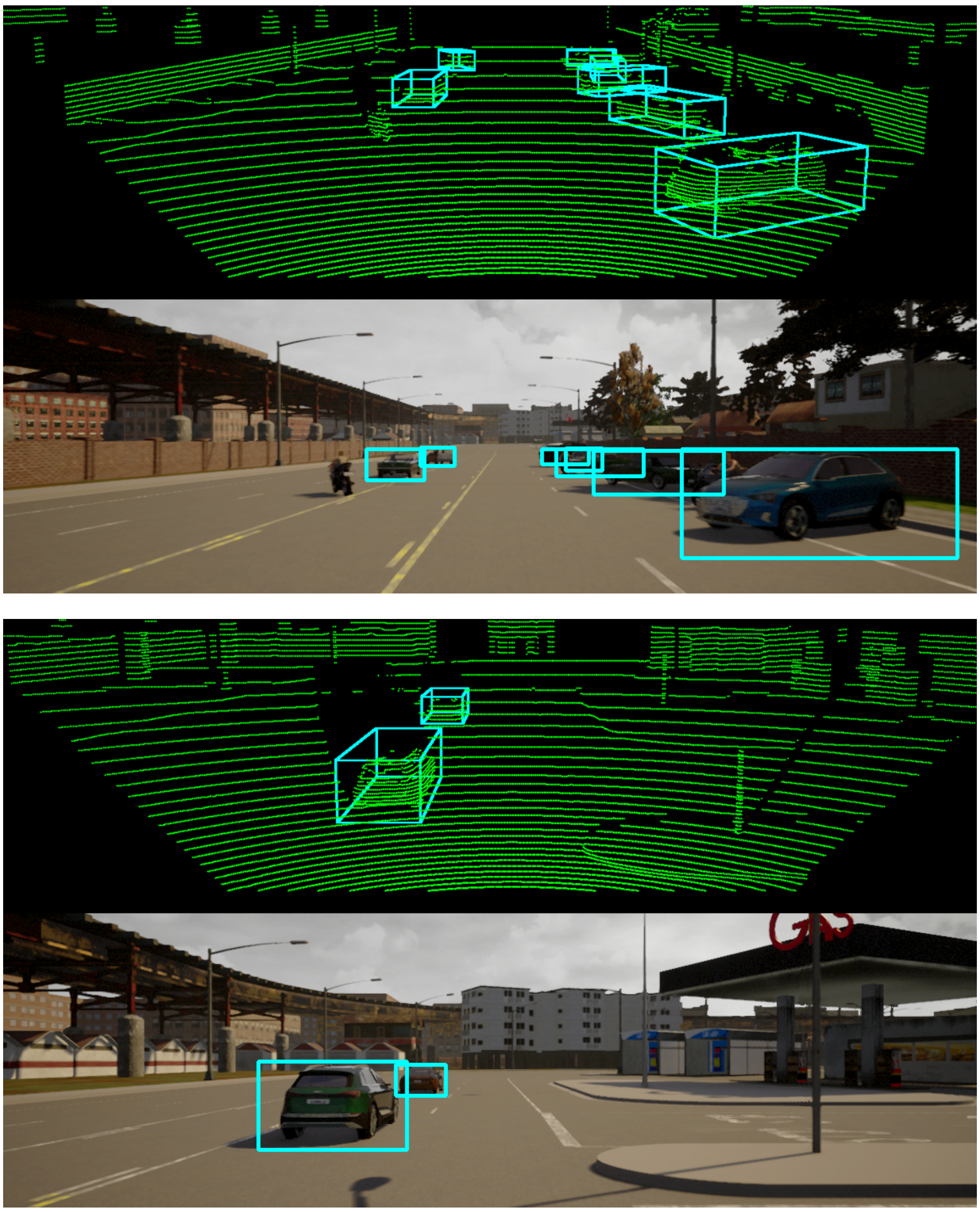


Figure A.2: Examples of scenes from the object detection artificial dataset, and the ground truth boxes (II).

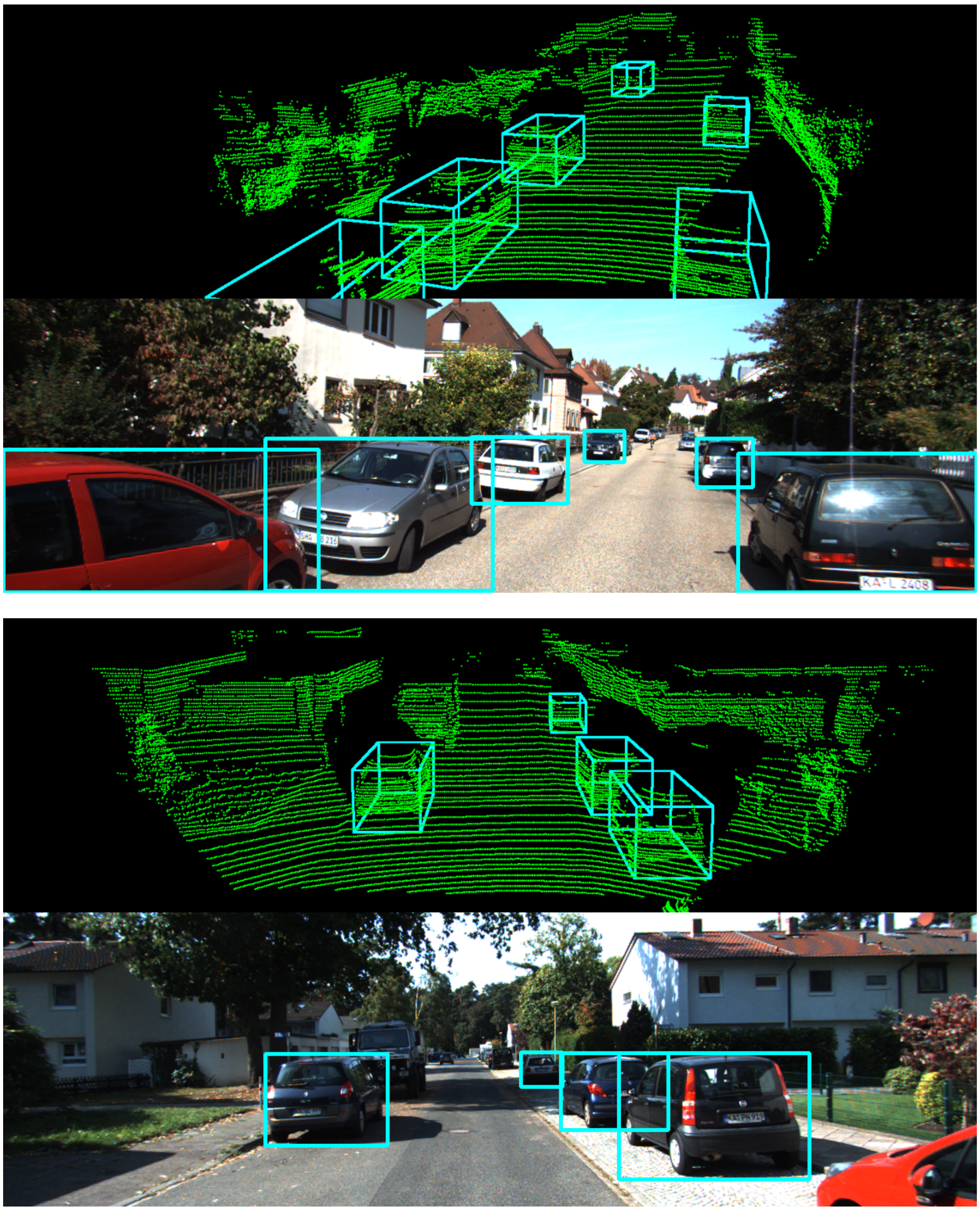


Figure A.3: Examples of scenes from the object detection real dataset, and the ground truth boxes (I).

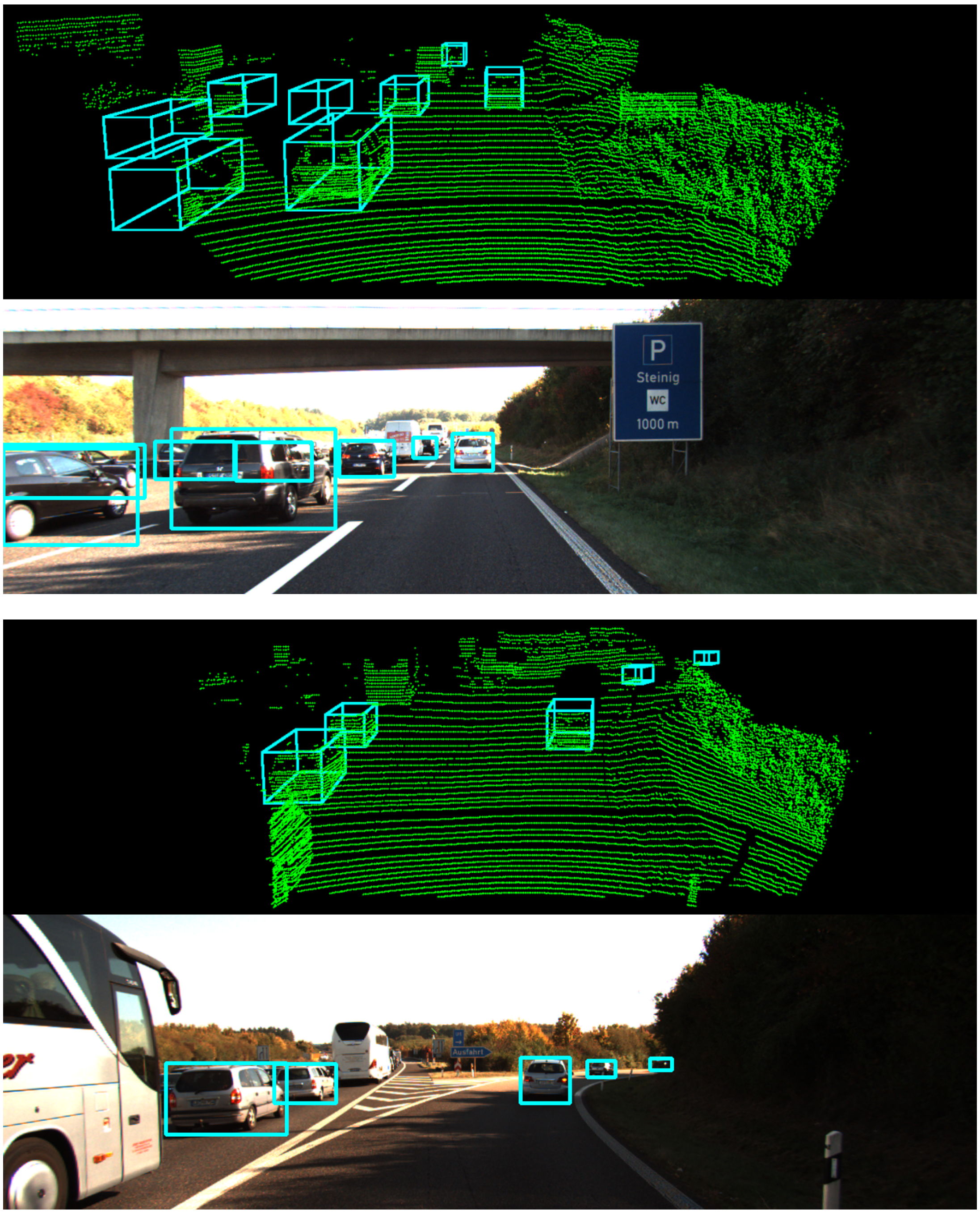
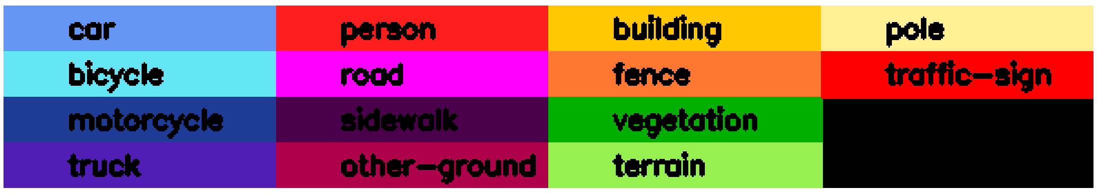


Figure A.4: Examples of scenes from the object detection real dataset, and the ground truth boxes (II).



car	person	building	pole
bicycle	road	fence	traffic–sign
motorcycle	sidewalk	vegetation	
truck	other–ground	terrain	

Figure A.5: Semantic colour code.

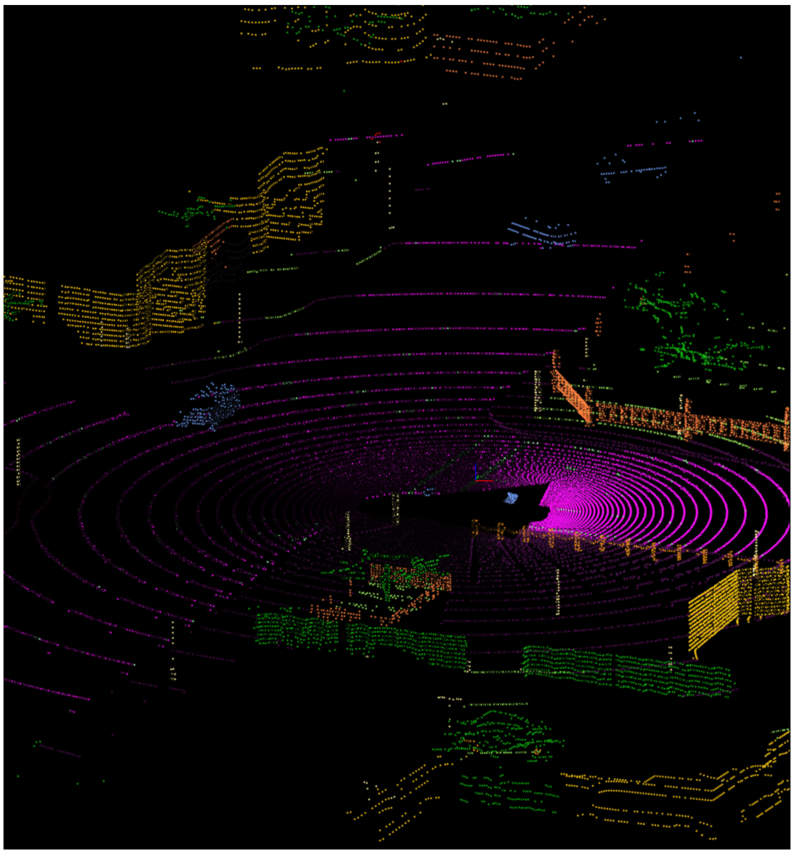
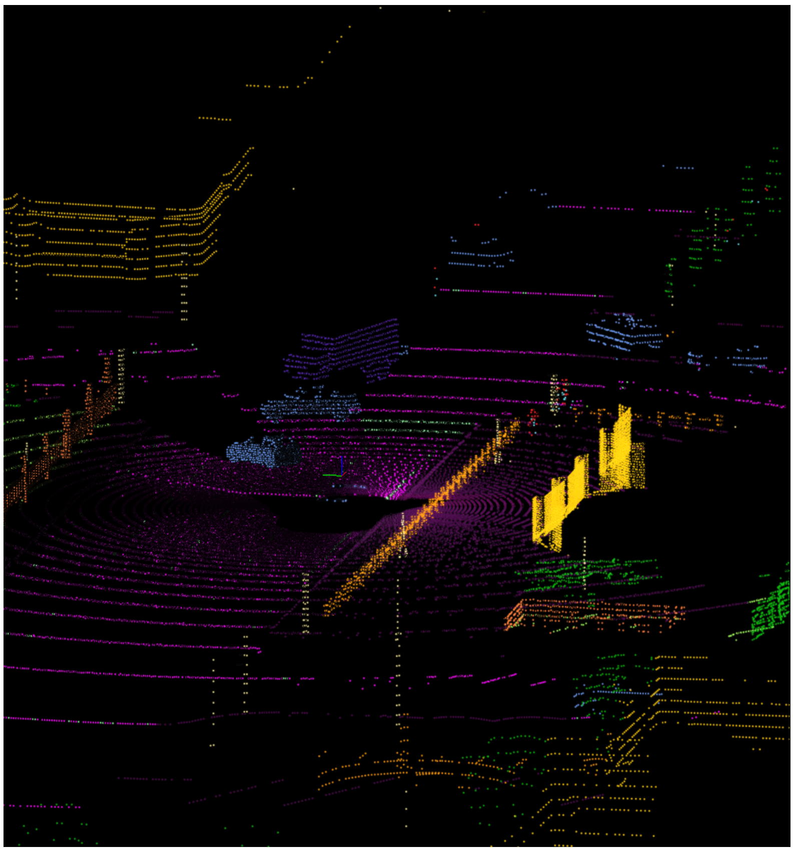
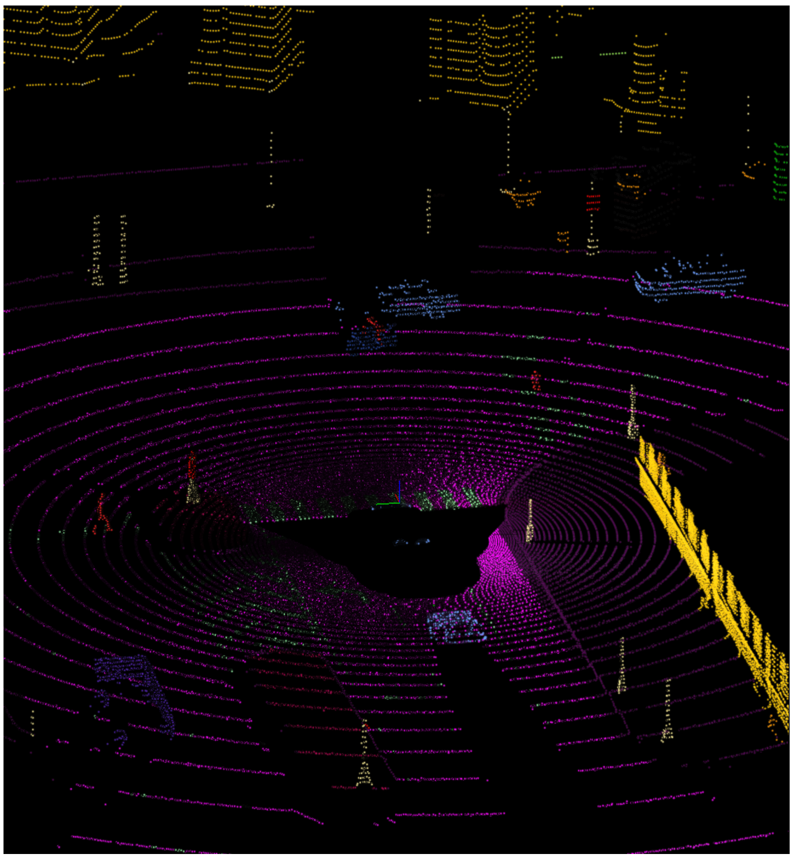



Figure A.6: Examples of scenes from the semantic segmentation artificial dataset with labeled points (I).



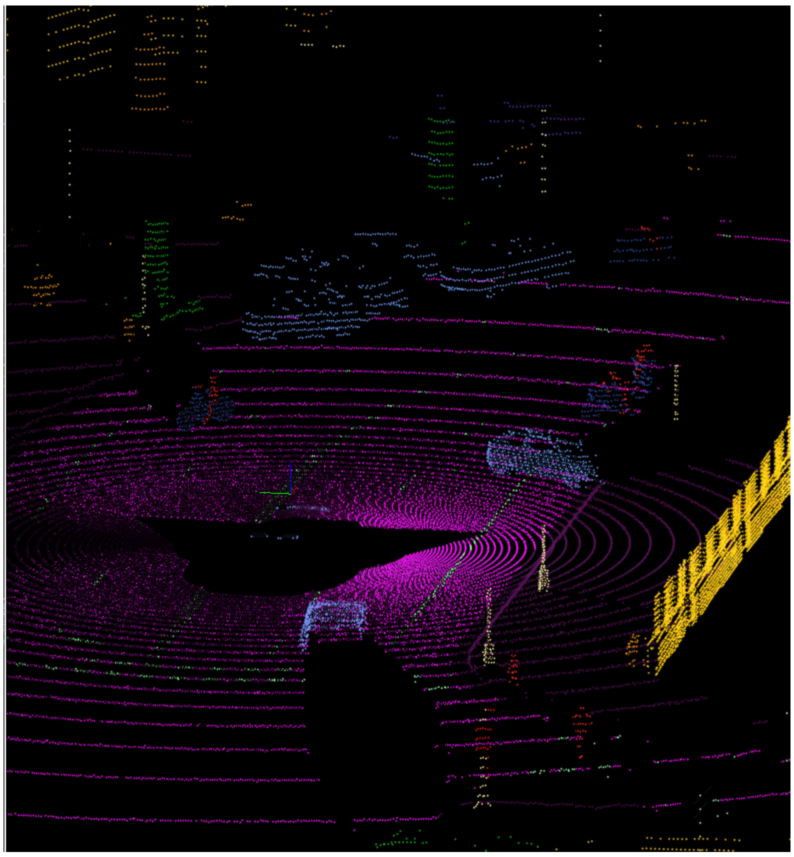


Figure A.7: Examples of scenes from the semantic segmentation artificial dataset with labeled points (II).

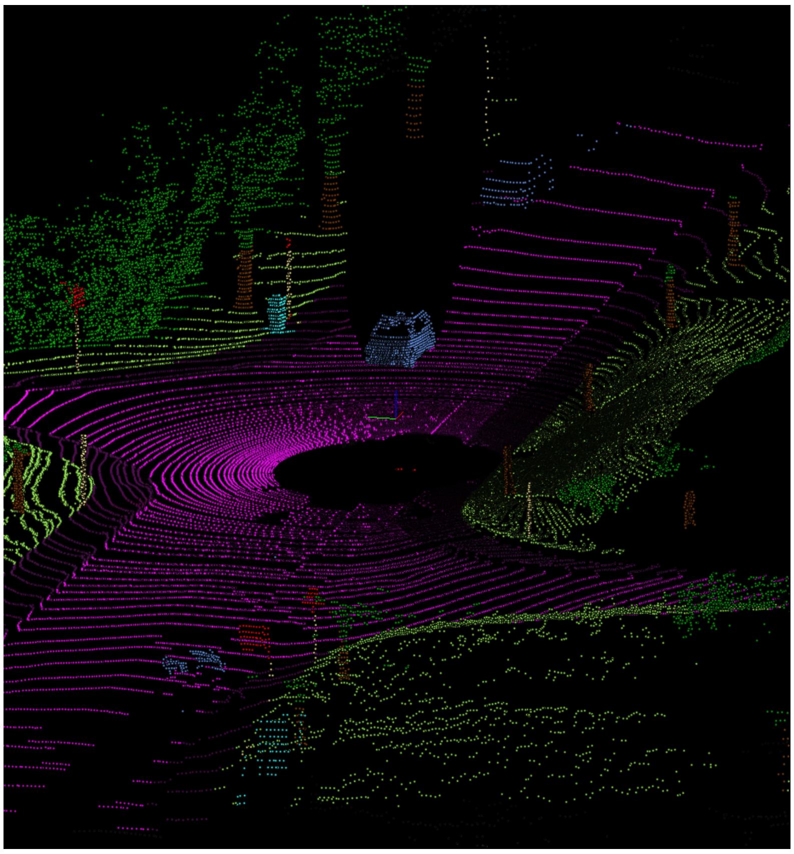
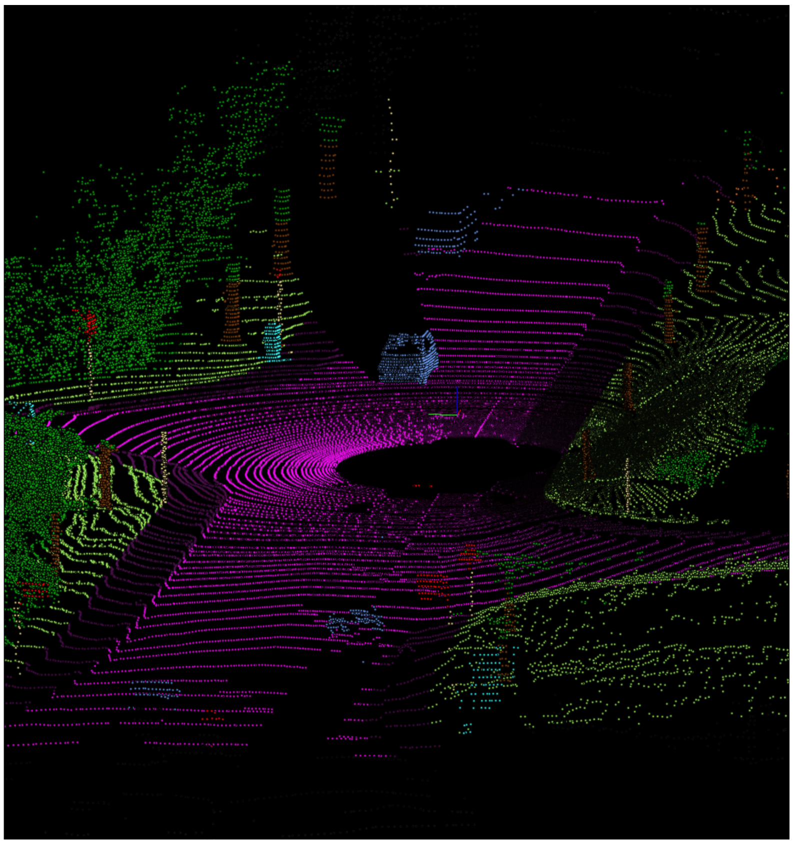



Figure A.8: Examples of scenes from the semantic segmentation real dataset with labeled points (I).

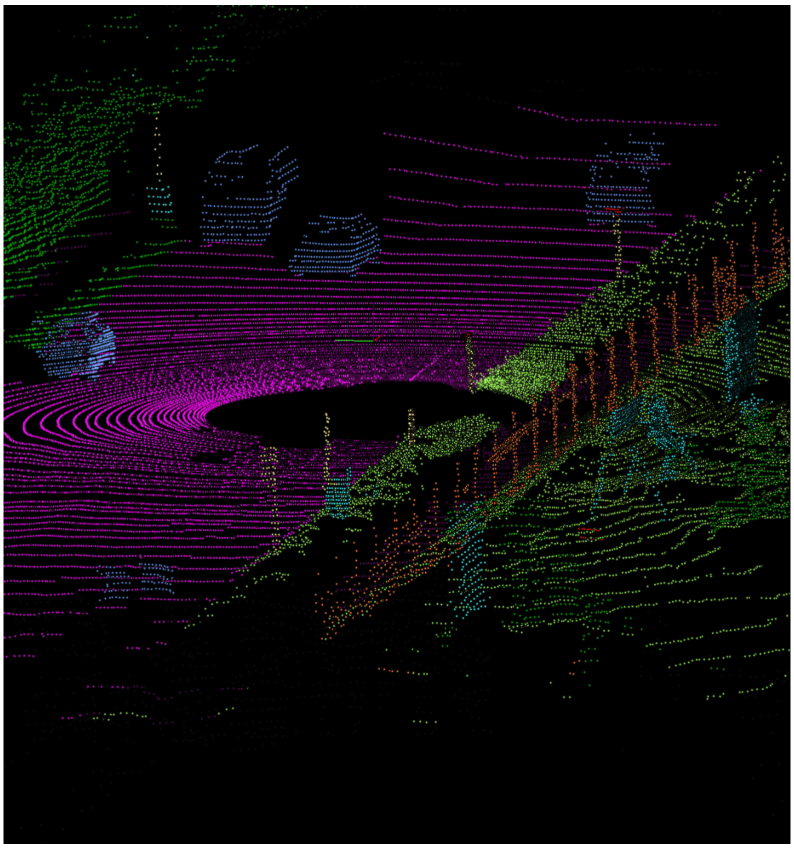
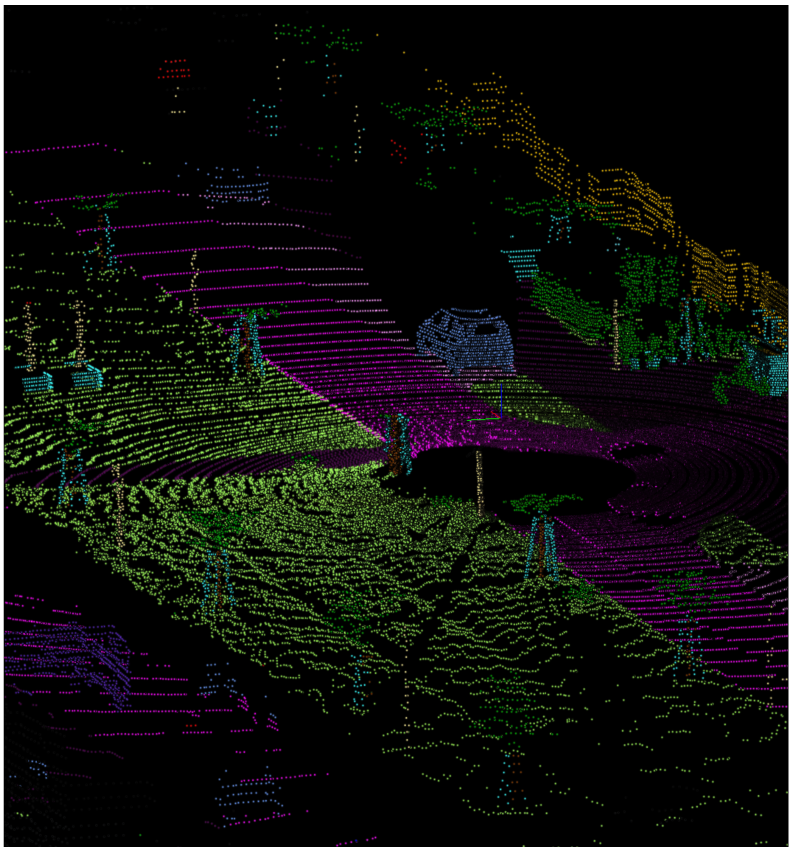



Figure A.9: Examples of scenes from the semantic segmentation real dataset with labeled points (II).