

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Deep Implicit Representations in Autonomous Driving

João Pedro Pinto Mota



Mestrado em Engenharia Informática e Computação

Supervisor: Jaime dos Santos Cardoso

External Supervisor: Ricardo Cerqueira

Second External Supervisor: João Ribeiro Pinto

July 29, 2022

Deep Implicit Representations in Autonomous Driving

João Pedro Pinto Mota

Mestrado em Engenharia Informática e Computação

July 29, 2022

Resumo

Compreender o ambiente é essencial para construir um veículo autónomo de qualquer nível. Uma forma de mapear o ambiente onde o veículo autónomo se encontra é usar uma nuvem de pontos tridimensional gerada por um LiDAR. Esta técnica melhora significativamente a capacidade do sistema de avaliar distâncias, levando a um aumento do interesse em pesquisa nesta aplicação de nuvens de pontos LiDAR.

No entanto, modelos de percepção supervisionada baseados em nuvens de pontos requerem extensos dados anotados, aumentando o seu custo. Uma maneira de melhorar isso é utilizar aprendizagem não supervisionada nessas nuvens de pontos para criar uma representação intermediária do ambiente na qual os algoritmos de percepção são aplicados.

Esta dissertação explora essa ideia e começa por analisar o desempenho de diferentes algoritmos de representação para aplicar num veículo autónomo com resultados diferentes: alguns apenas funcionam à escala de objectos, outros podem representar uma cena à escala do mundo real, mas apenas alguns são aplicáveis a um contexto do mundo real. Em seguida, uma arquitetura recente chamada SISC é reimplementada e usada para experimentar uma nova abordagem de fusão de sensores com dados LiDAR e RGB. Este método de representação visa alcançar bons resultados onde outros métodos encontram dificuldades: a necessidade de representar detalhes numa grande cena do mundo real a partir de entradas esparsas de LiDAR e RGB, ao mesmo tempo em que tenta melhorar a representação e expressar informações de cores, e também melhorar o desempenho do modelo na tarefa de segmentação da cena.

Os resultados mostram que a abordagem de fusão de dados melhorou a qualidade da reconstrução geométrica do modelo com um aumento de 0,3% em IoU. Ainda assim, não conseguiu representar com precisão a cor da cena, sendo capaz apenas de expressar corretamente a luminância dos objetos. O desempenho da segmentação semântica também melhorou, com um aumento de 0,2% de mIoU.

Espera-se que a implementação desta abordagem num veículo autónomo aumente a robustez do sistema, tornando as tarefas de percepção menos suscetíveis a mudanças nas condições ambientais e compensando alguns artefactos do sensor, reduzindo significativamente a necessidade de dados rotulados, uma vez que estas representações são aprendidas sem supervisão. No entanto, espera-se também que isto leve a um aumento de complexidade e tempo de processamento.

Abstract

Understanding the environment is essential to building a self-driving vehicle of any level. One way to map the autonomous vehicle's environment is to use a three-dimensional point cloud of points generated with LiDAR. This technique significantly improves the system's ability to judge distances, leading to increased research interest in this application of LiDAR point clouds.

However, supervised perception models based on point clouds require extensive annotated data, increasing their cost. One way to improve this is to employ unsupervised learning on these point clouds to create a deep intermediate representation of the environment on which perception algorithms are applied.

This dissertation explores that idea and starts by analyzing the performance of different representation algorithms for application in an autonomous vehicle, with different results: some only perform at object-scale, and others can represent a real-world scale scene, but merely a few are applicable to a real-world context. Following this, a recent architecture named SISC is re-implemented and used to experiment on a new sensor fusion approach with LiDAR and RGB data. This representation method aims at achieving good results where other methods encounter difficulties: the need to represent scene-level details in a large real-world scene from sparse LiDAR and RGB input, while also attempting to increase its representation capabilities to express color information and improving the segmentation performance of the model.

Results showed that the data fusion approach increased the geometric reconstruction quality of the model with an increase of 0.3% in IoU. Still, it could not accurately represent the scene's color, being only able to correctly express the luminance of objects. Semantic segmentation performance also improved, with a 0.2% increase in mIoU.

Implementing this approach in an autonomous vehicle is expected to increase the system's reliability by making perception tasks less susceptible to changes in environmental conditions and compensating for some sensor artifacts while significantly reducing the need for labeled data as these representations are learned without supervision. However, this is expected to come at the cost of increasing complexity and processing time.

Keywords: Scene Reconstruction, Scene Representation, Scene Completion, Sensor Fusion, Deep Learning, Self Driving, Intermediary Representation, Point Clouds, LiDAR, Deep Implicit Functions

Acknowledgements

Aos meus pais.

Aos meus orientadores, pelo acompanhamento, disponibilidade e direção.

À Bosch e à equipa, pela oportunidade e por toda a ajuda.

À Bárbara, por todo o apoio.

A todos os que acompanharam este percurso, e me fizeram crescer e aprender.

This work was supported by the European Structural and Investment Funds in the FEDER component, through the Operational Competitiveness and Internationalization Programme (COMPETE 2020) [Project nº 047264; Funding Reference: POCI-01-0247-FEDER-047264].

João Pedro Pinto Mota

“It works on my machine.”

Unknown

Contents

1	Introduction	1
1.1	Context	1
1.2	Motivation	1
1.3	Objectives	2
1.4	Document Structure	2
2	Background and Concepts	4
2.1	Data in Autonomous Vehicles	4
2.1.1	Sensors Used	4
2.1.2	Data Acquisition	4
2.2	Deep Learning	5
2.2.1	Convolutional neural networks	5
2.2.2	Autoencoders	6
2.3	Regularization	7
2.3.1	Dropout	7
2.3.2	Weight Decay	7
2.4	Sensor Fusion	7
2.5	Signed Distance Functions	8
2.6	Marching Cubes	8
3	Implicit Representations from Point-Clouds	13
3.1	Object-scale Representations	13
3.1.1	Occupancy Networks	13
3.1.2	DeepSDF	14
3.2	Scene-scale Representations	14
3.2.1	Convolutional Occupancy Networks	14
3.2.2	DeepLS	15
3.2.3	SIREN	15
3.2.4	Semantic Scene Completion using Local Deep Implicit Functions on Li-DAR Data	15
3.2.5	SISC	17
3.3	Evaluation	17
3.3.1	IoU	17
3.3.2	mIoU	17
3.4	Datasets	18
3.5	Discussion	23

4	Methodology	24
4.1	The Representation	24
4.1.1	How it works	24
4.1.2	Visualizer	27
4.1.3	Issues with the representation	27
4.1.4	Validation loss	28
4.1.5	Dropout	29
4.2	DIR and Sensor Fusion	29
4.2.1	Data preprocessing	30
4.2.2	Architecture changes	31
4.2.3	Loss	31
4.2.4	Data Efficiency	32
5	Results and Discussion	33
5.1	Representation with LiDAR Data	33
5.1.1	Baseline	33
5.1.2	Validation Loss	34
5.1.3	Dropout	34
5.2	Sensor-Fusion with RGB and LiDAR Data	39
5.2.1	Occupancy	39
5.2.2	Color Output	42
5.2.3	Semantic Segmentation	42
5.2.4	Data Efficiency	43
6	Conclusions and Future Work	50
	References	52

List of Figures

2.1	NuScenes' sensor setup for data collection [5]	5
2.2	Multi-layer perceptron and deep learning neural network [49]	6
2.3	a–c An example of convolution operation with a kernel size of 3×3 , no padding, and a stride of 1. [50]	9
2.4	Example of max pooling operation with a filter size of 2×2 , no padding, and a stride of 2, which extracts 2×2 patches from the input tensors and outputs the maximum value in each patch, resulting in downsampling of the in-plane dimension of an input tensor by a factor of 2. [50]	10
2.5	An autoencoder example. The input image is encoded into a compressed representation and then decoded. [2]	10
2.6	Level one represents early fusion, level two represents halfway fusion, and level three represents late fusion. [25]	11
2.7	Originally published 15 combinations [31].	12
3.1	Architecture of occupancy networks [15]	14
3.2	Auto-encoder architecture in a) and auto-decoder in b). [36]	16
3.3	Architecture of convolutional occupancy networks [16]	17
3.4	Qualitative analysis with a similar number of parameters on a synthetic scene [6]	19
3.5	Qualitative analysis with ReLU activation function [46]	19
3.6	Network architecture of [41]	20
3.7	Left to right, top to bottom: Input points, ground truth accumulated points, mesh visualization of continuous output function, derived voxelization at 20 cm edge length. From [41].	20
3.8	Network architecture of [27]	21
3.9	Left: Voxelized input scan with semantic information (for visualization). Right: Target output with completed and labeled scene. [3]	23
4.1	Network structure of Com-Net, from [27]	25
4.2	Top-down view of the model output for frame 545 from sequence 8, trained until epoch 50.	27
4.3	Top-down view of the model output for frame 545 from sequence 8, trained until epoch 200. Overfitting artifacts are visible on both sides of the road: the top and bottom of the image.	28
4.4	Occupied voxels projected into the image as black dots for debugging purposes.	31
5.1	Top-down view of the model output for frame 545 from sequence 8, trained until epoch 100. Overfitting artifacts are visible on both sides of the road: the top and bottom of the image.	36

5.2	Single frame IoU used as model validation, trained until epoch 200. Value in y and number of steps in x	37
5.3	Newly implemented model's validation loss, trained until epoch 195. Value in y and number of steps in x	37
5.4	Validation loss until epoch 50 for models with data augmentation and dropout probabilities of 30% (pink) and 50% (green), and no data augmentation and dropout probabilities of 20% (blue) and 40% (red).	40
5.5	Top-down view of the model output for frame 545 from sequence 8, trained until epoch 200 with dropout probability of 30% (see subsection 4.1.5).	40
5.6	Top-down view of the model output for frame 545 from sequence 8, trained until epoch 50 with RGB input. Overfitting artifacts are visible on both sides of the road: the top and bottom of the image.	43
5.7	Top-down view of the model output for frame 545 from sequence 8, trained until epoch 50 with RGB input and dropout.	44
5.8	Validation loss for RGB component, trained until epoch 50.	44
5.9	Top-down view of the model output for frame 545 from sequence 8, trained until epoch 50 with RGB input and output. Overfitting artifacts are visible on both sides of the road: the top and bottom of the image.	45
5.10	Top-down view of the model output for frame 545 from sequence 8, trained until epoch 50 with RGB input and output, and dropout.	45
5.11	Camera 2's image of the scene for frame 545 in sequence 8.	46
5.12	Top-down view of the model output for frame 545 from sequence 8 with semantic segmentation, trained until epoch 50. Overfitting artifacts are visible on both sides of the road: the top and bottom of the image.	46
5.13	Top-down view of the model output for frame 545 from sequence 8 with semantic segmentation, trained until epoch 50 with RGB input. Overfitting artifacts are visible on both sides of the road: the top and bottom of the image.	47
5.14	Validation loss for data efficiency experiment with sensor fusion (see subsection 4.2.4). Orange is LiDAR only and blue RGB and LiDAR fusion.	48

List of Tables

3.1	Performance comparison on 3D reconstruction from point clouds on the ShapeNet [7] dataset, from [33]	15
3.2	Performance comparison on the real-world ScanNet V2 dataset [11], from [38]	18
3.3	Performance comparison on 3D Scene Dataset [52]	19
3.4	Comparing multiple datasets (adapted from [29])	22
5.1	Parameters used to replicate original paper results.	34
5.2	Replicated results: IoU score for each threshold.	35
5.3	IoU score for each threshold for the dropout experiments and the baseline at epoch 50.	38
5.4	IoU score for each threshold for the chosen dropout experiment and the baseline at epoch 200.	39
5.5	IoU score for each threshold with sensor fusion (see section 4.2) at epoch 50.	41
5.6	IoU score for each threshold with sensor fusion (see section 4.2) and dropout (see subsection 4.1.5) at epoch 50.	41
5.7	mIoU score for each threshold with sensor fusion (see section 4.2) in the semantic scene completion task.	42
5.8	IoU score for each threshold with sensor fusion (see section 4.2) in the semantic scene completion task.	44
5.9	IoU score for each threshold while training with half the dataset (see subsection 4.2.4) at epoch 50.	49

Abbreviations

3D	Three-dimensions
AV	Autonomous Vehicles
CNN	Convolutional neural network
FoV	Field of View
GPU	Graphics processing unit
Hz	Hertz
IMU	Inertial Measurement Unit
IoU	Intersection over Union
LiDAR	Light Detection And Ranging
MLP	Multi-layer perceptron
ms	milliseconds
PLY	Polygon File Format
RGB	The RGB (Red, Green, Blue) color model
SDF	Signed distance function

Chapter 1

Introduction

1.1 Context

One of the great goals of the automotive industry is to create fully autonomous vehicles that allow their passengers to travel without needing to worry about driving. SAE International divided this goal into six steps, referred to as levels, from 0 to 5 [44].

Currently, most new high-end vehicles achieve level 1 automation in certain conditions, namely on the highway, by accelerating and braking automatically, sharing the vehicle control with its driver. In contrast, some vehicles can even reach level 2 autonomy, allowing the driver to let go of the steering wheel and pedals, giving complete control to the car while maintaining attentive to its behavior and surroundings.

To achieve level 3 and above, a new suite of sensors is required to monitor every angle around the vehicle and enable more complex actions. This goal is leading to an increase in the research interest in this application of LiDAR, which many companies and researchers believe is vital to fully autonomous vehicles, as can be seen by the sensor suite present in the only two approved systems that achieve level 3 autonomy currently: the Honda Legend EX with Sensing Elite [20], and the Mercedes S-Class with Drive Pilot [32], both with LiDAR sensors.

This is because LiDARs present a way to extract accurate and geo-referenced data from the environment, using laser pulses to calculate its distance to multiple objects by measuring how long it takes for the reflected light to hit the sensor, allowing it to create three-dimensional point clouds of the scene [37].

1.2 Motivation

Currently, most vehicles that have some degree of autonomy, namely levels 1 and 2, use a front-mounted radar and cameras for the safe operation of their cruise control system. This is, however, not enough for vehicles that aim to achieve higher levels of autonomy. For those systems, LiDAR

sensors have evident applications since the accurate and reliable awareness of the environment is vital for many of the perception problems that need to be tackled for safe self-driving.

Autonomous vehicles can use the point clouds created with LiDAR to perform various tasks with higher accuracy and reliability than only using traditional cameras due to their high density and geo-referencing characteristics [37].

These perception tasks, such as scene segmentation, object detection, localization, and recognition, require the real-time processing of the point cloud, often using deep-learning models trained with real-world data from test vehicles equipped with these sensors.

However, even though the use of LiDAR can significantly increase the safety of autonomous vehicles by increasing the accuracy and reliability of their perception systems, it can also have some drawbacks.

This training of deep learning models on LiDAR's point clouds requires significant amounts of annotated data due to the high complexity of the task, which substantially increases the cost and difficulty of creating an acceptable model. Other drawbacks of using point clouds in autonomous driving are the model's dependency on the version of the sensor in use and its position.

1.3 Objectives

This dissertation intends to study a novel application of deep implicit representations to create a three-dimensional model of the world surrounding the autonomous vehicle that will be used by other deep-learning models.

The representation is expected to reduce the system's sensibility to environmental conditions and the sensor used, lessen the need for labeled data, and simplify many semantic tasks, such as object detection, ego-motion estimation, normal estimation, and sensor fusion.

The scope of this dissertation, however, is to attempt to improve an existing model and experiment with a common task in perception for autonomous vehicles that aims to improve the performance and reliability of the system: sensor fusion with LiDAR and RGB data. This can be used to both improve the geometric reconstruction quality, allow the model to output color information that can be used to enrich the representation, and also to improve the scene segmentation capabilities of the model.

1.4 Document Structure

This dissertation starts with Chapter 1, which describes what motivated this work and what it is trying to achieve, with Chapter 2 giving some extra background and explaining concepts that are relevant in the context of this work. Chapter 3 further builds on this context to provide a more detailed analysis of relevant techniques related to the objectives of this dissertation.

The applied methodology is detailed in Chapter 4, where the procedures for all experiments are outlined, and Chapter 5 exposes the obtained results together with an interpretation of what these mean.

Lastly, Chapter 6 is responsible for drawing final conclusions and summarizing what has been done and what this dissertation achieved. It also leads readers to interesting directions for future research in related topics.

Chapter 2

Background and Concepts

2.1 Data in Autonomous Vehicles

2.1.1 Sensors Used

Autonomous vehicles need reliable and accurate data sources to infer about the surrounding environment. To this end, multiple sensors are used with different goals and strengths, and their signals are processed in real-time by the onboard computer. The most commonly used sensor suite is composed of LiDARs, RADARs, and RGB cameras, as, for example, the level 4 Waymo vehicles currently uses [1].

LiDAR is a sensor that uses laser pulses to accurately measure the distance to several points in the surrounding space and works well in all light conditions. Its higher resolution allows these sensors to paint a clearer 3D picture than radars, and their biggest downside is the high cost and unreliability in adverse weather conditions such as heavy rain, fog, and snow.

RADARs on the other hand use radio waves to measure distance instead of laser pulses, allowing these to work over longer distances. RADARs are also relatively less expensive and less susceptible to adverse weather.

Lastly, vehicles also have multiple RGB cameras positioned around the vehicle to allow for 360° vision of the environment. These are used to capture visual information such as traffic lights, construction zones, and traffic signs, and also to identify objects. Cameras, however, cannot directly capture depth information and struggle with direct sunlight and low light environments.

2.1.2 Data Acquisition

Usually, to capture these data points, the sensors are spread around the car allowing 360° sensing of the environment. Using the data collection platform of the nuScenes dataset [5] seen in Figure 2.1 as an example, there are 5 RADARs around the car, with one in each corner plus one in the front,

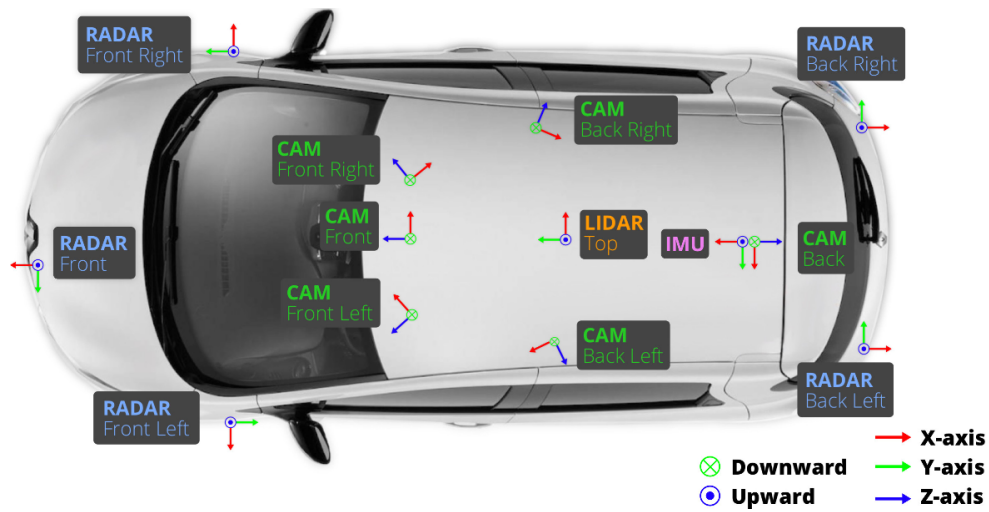


Figure 2.1: NuScenes' sensor setup for data collection [5]

6 cameras facing in all 4 directions with 2 extra in the front, an inertial measurement unit (IMU), and finally, a 360° horizontally spinning LiDAR at the top.

To store data from the aforementioned LiDAR sensor, the most common approach is to save it as a point cloud but it can also be voxelized into a grid of voxels that divides the scene into a group of spatial cubes and saves the occupancy of each one. At the same time, cameras store individual frames as images, and the RADAR data is, in this case, the raw output of the sensor used by the nuScenes data collection platform.

2.2 Deep Learning

Deep learning is a class of machine learning algorithms based on artificial neural networks with multiple hidden layers, as seen in Figure 2.2, and has applications in various fields, such as computer vision and natural language processing [26]. These models use the backpropagation algorithm to update their internal parameters, which are used afterward to compute the output of each layer from the previous one [23].

This type of algorithm is especially suited for situations where the data is complex and large datasets are available.

2.2.1 Convolutional neural networks

One typical architecture of a deep learning model usually used in computer vision with applications in 3D implicit representations is convolutional neural networks (CNN). These models are able to detect important features without human supervision. For this effect, the following building blocks are used:

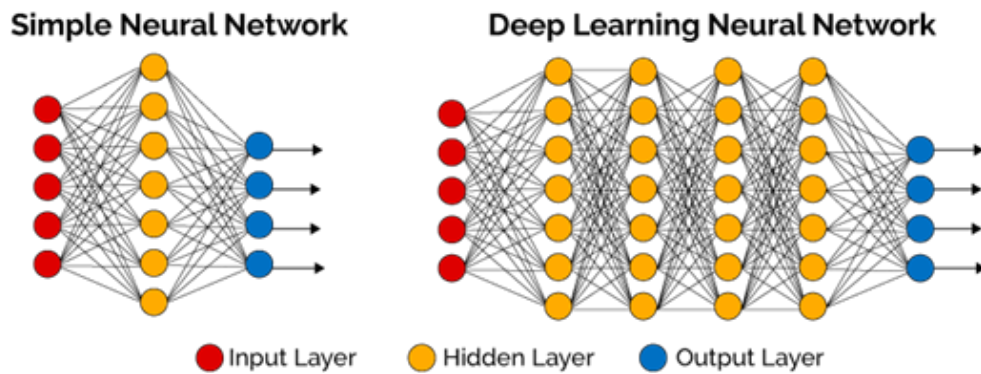


Figure 2.2: Multi-layer perceptron and deep learning neural network [49]

- Convolutional layers are responsible for extracting features. They apply a small array of numbers called kernel across the input by summing their element-wise products, as seen in Figure 2.3. Some hyperparameters that affect convolutions are the size and number of kernels, padding size, which determines whether the outermost elements of the input tensor are overlapped by the kernel center, and the stride distance, which determines the distance between two successive kernel positions. [50]
- Pooling layers' goal is to downsample the dimensionality of the feature maps while introducing translation invariance for small shifts, decreasing the number of learnable parameters. These layers often choose the highest value in a filter, as seen in Figure 2.4 (max-pooling), or compute the average of all elements (average-pooling). Like convolutional layers, stride, padding, and filter size are hyperparameters of pooling operations. [50]
- Global average pooling is an optional layer that can replace fully connected layers in a CNN. It generates a feature map for each category, which will be later averaged and fed to the next layer. Some advantages of this technique are that there is no parameter to optimize at this layer, thus avoiding overfitting and added complexity, and it increases the model's robustness to spatial translations of the input by summing up the spatial information. [30]
- Fully connected layers contain most of the trainable parameters of the network. Also known as dense layers, these are responsible for connecting the final convolution or pooling layer to every output. [50]

2.2.2 Autoencoders

One other interesting architecture is an autoencoder. First introduced in [43], it has the ability to learn efficient representations of the input and model generative processes. This type of network has two main parts, an encoder that maps the input to a latent representation that is later used by a decoder that aims to reconstruct the input from the latent variables, as seen in Figure 2.5. Autoencoder's abilities can be applied to multiple problems, such as image compression, dimensionality reduction, and anomaly detection. [2]

2.3 Regularization

Regularization is a set of techniques that aim to diminish overfitting. Some of the most interesting are presented in this section.

2.3.1 Dropout

Dropout is a regularization method commonly and effectively used against overfitting in neural networks [51]. Its main idea is to randomly drop neurons and connections from the network during training in order to prevent units from co-adapting more than desired. By doing this iteratively, overfitting is somewhat inhibited.

2.3.2 Weight Decay

Also known as L2 penalty, weight decay uses the Euclidean distance formula as a penalty term, leading the network to learn features with smaller weights. This means that the features with less impact on the final prediction will have their weights minimized, limiting their effect. [51]

Mathematically it can be defined with the formula 2.1:

$$\Omega(w) = \sqrt{\sum_i w_i^2}, [51] \quad (2.1)$$

2.4 Sensor Fusion

In deep learning, more specifically in the context of autonomous vehicles (AV), sensor fusion is an important technique that increases the robustness and performance of a model, especially in adverse conditions, by feeding information from multiple sources to the perception model, allowing it to avoid the performance degradation expected from the natural shortcomings of each individual sensor if used alone. [25]

There are three main types of fusion, varying where data is fused in the network, as seen in Figure 2.6. Early fusion (or data fusion) happens at the raw data level: the output from multiple sensors is fused in the first layers of the network. In Halfway fusion (or feature fusion), features are extracted from sensor data and then these are fused, instead of the raw data. The merged features will be fed to decision layers which will use information from the multiple sensors to form a better decision. The last type is called late fusion (or decision fusion), and here, unlike the previous types, multiple classifiers are used to independently generate decisions for each sensor which will be combined to form the final output. [13]

2.5 Signed Distance Functions

A distance function f

$$f(x) : \mathbb{R}^3 \rightarrow \mathbb{R}, \quad (2.2)$$

can represent any object implicitly by mapping the distance of any point in Euclidean space to the closest point on the surface, being $f(x) = 0$ for on-surface points.

Distance functions can be expanded with the use of positive and negative values, allowing a signed distance function (SDF) to represent more information from its return value: points within and outside the surface or object will have a different sign, usually positive values inside and negatives outside, but it is often inverted. [35]

As an example, an SDF f :

$$f(x, y, z) = \sqrt{x^2 + y^2 + z^2} - r, \quad (2.3)$$

represents a sphere centered at the origin and with radius r .

2.6 Marching Cubes

The Marching Cubes algorithm [31] aims to extract a polygonal mesh from a three-dimensional scalar field. It was originally designed to be used with 3D medical data but can be applied in multiple situations, such as extracting a mesh from an implicit representation with signed distance functions.

For this end, the algorithm employs a divide-and-conquer strategy. It locates the surface in every logical cube created from eight pixels and then creates triangles to represent the located surface while also computing their vertex normals. After this step is concluded, the algorithm marches to the next cube.

To determine the surface in a cube, it checks for each of the cube vertexes if they are within an object. In the case of a signed distance function representation of a scene, it would consider vertexes with an SDF value below zero as outside an object and positive values as position within. The surfaces for each of the two-hundred and fifty-six possible combinations ($2^8 = 256$) are pre-computed to make the algorithm more efficient. Figure 2.7 shows fifteen combinations as presented in the original paper.

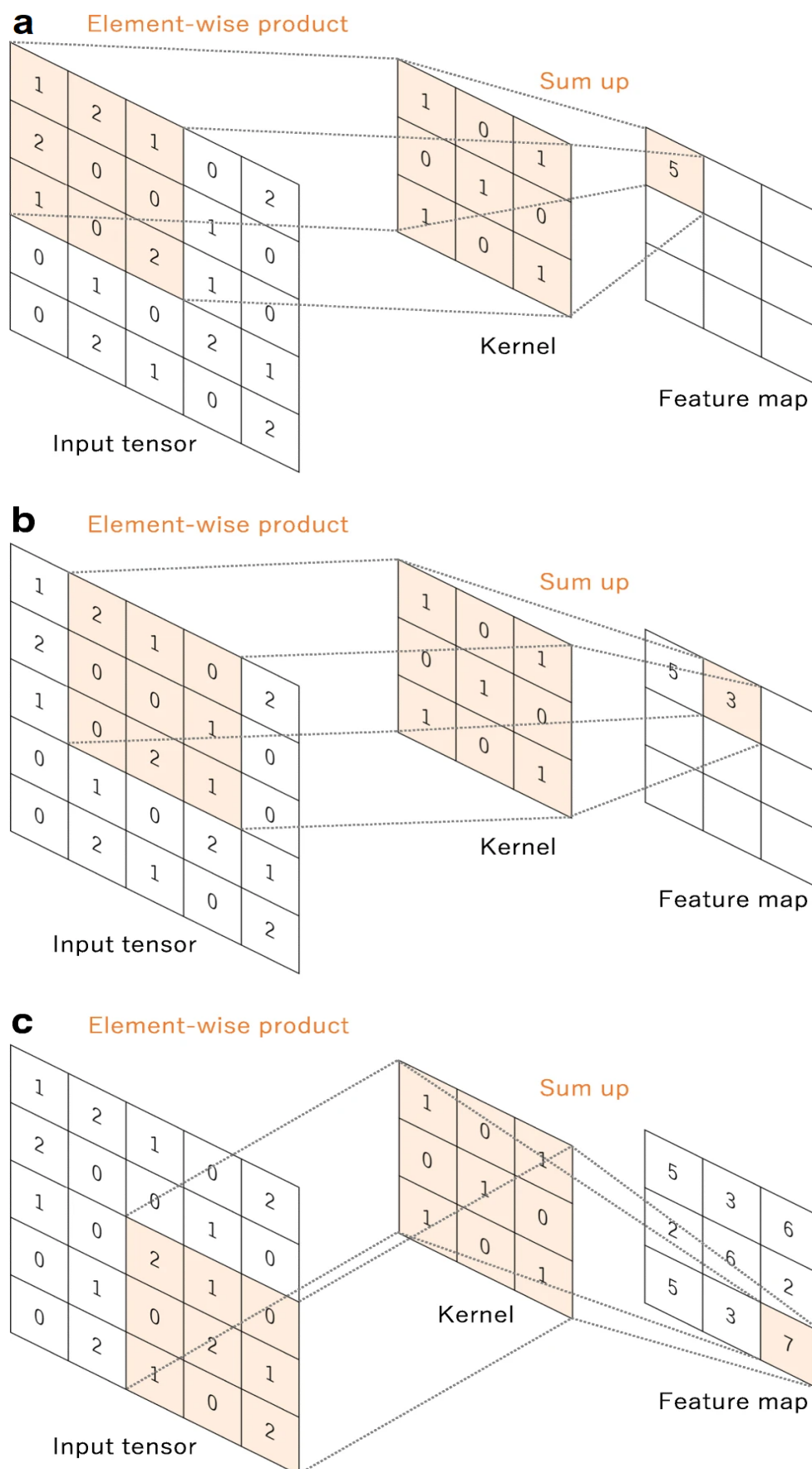


Figure 2.3: a–c An example of convolution operation with a kernel size of 3×3 , no padding, and a stride of 1. [50]

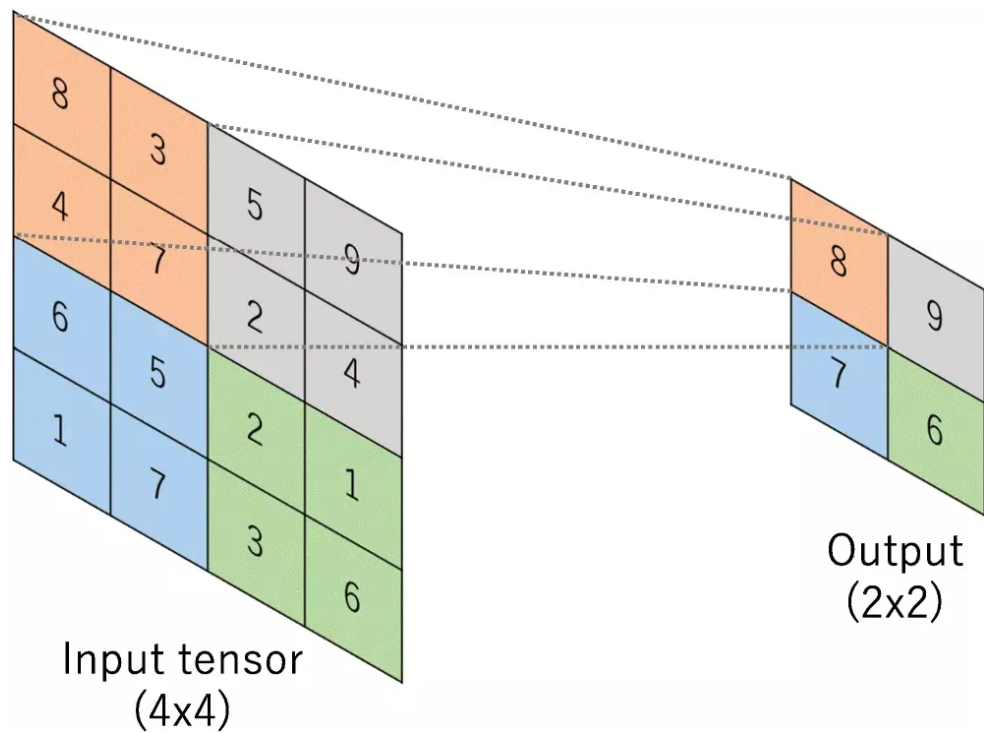


Figure 2.4: Example of max pooling operation with a filter size of 2×2 , no padding, and a stride of 2, which extracts 2×2 patches from the input tensors and outputs the maximum value in each patch, resulting in downsampling of the in-plane dimension of an input tensor by a factor of 2. [50]

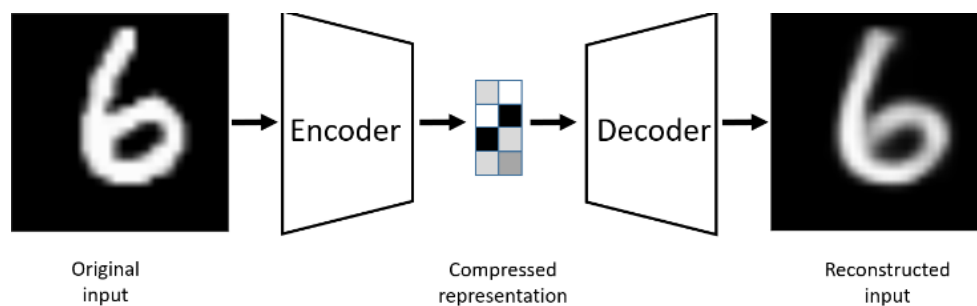


Figure 2.5: An autoencoder example. The input image is encoded into a compressed representation and then decoded. [2]

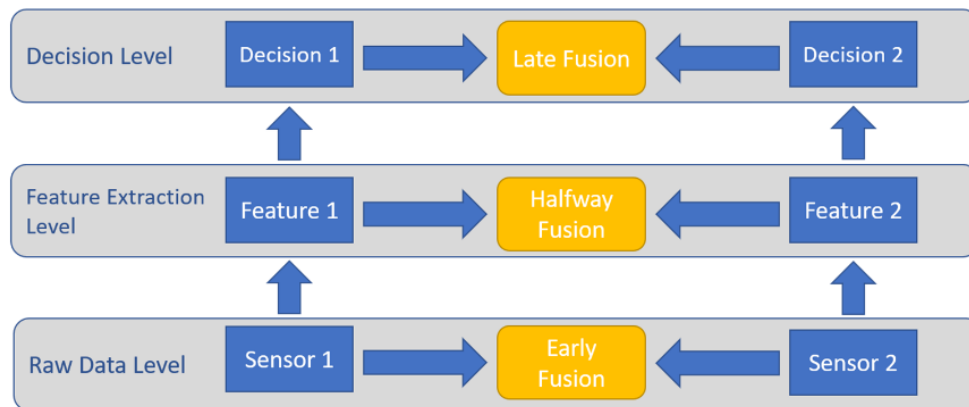


Figure 2.6: Level one represents early fusion, level two represents halfway fusion, and level three represents late fusion. [25]

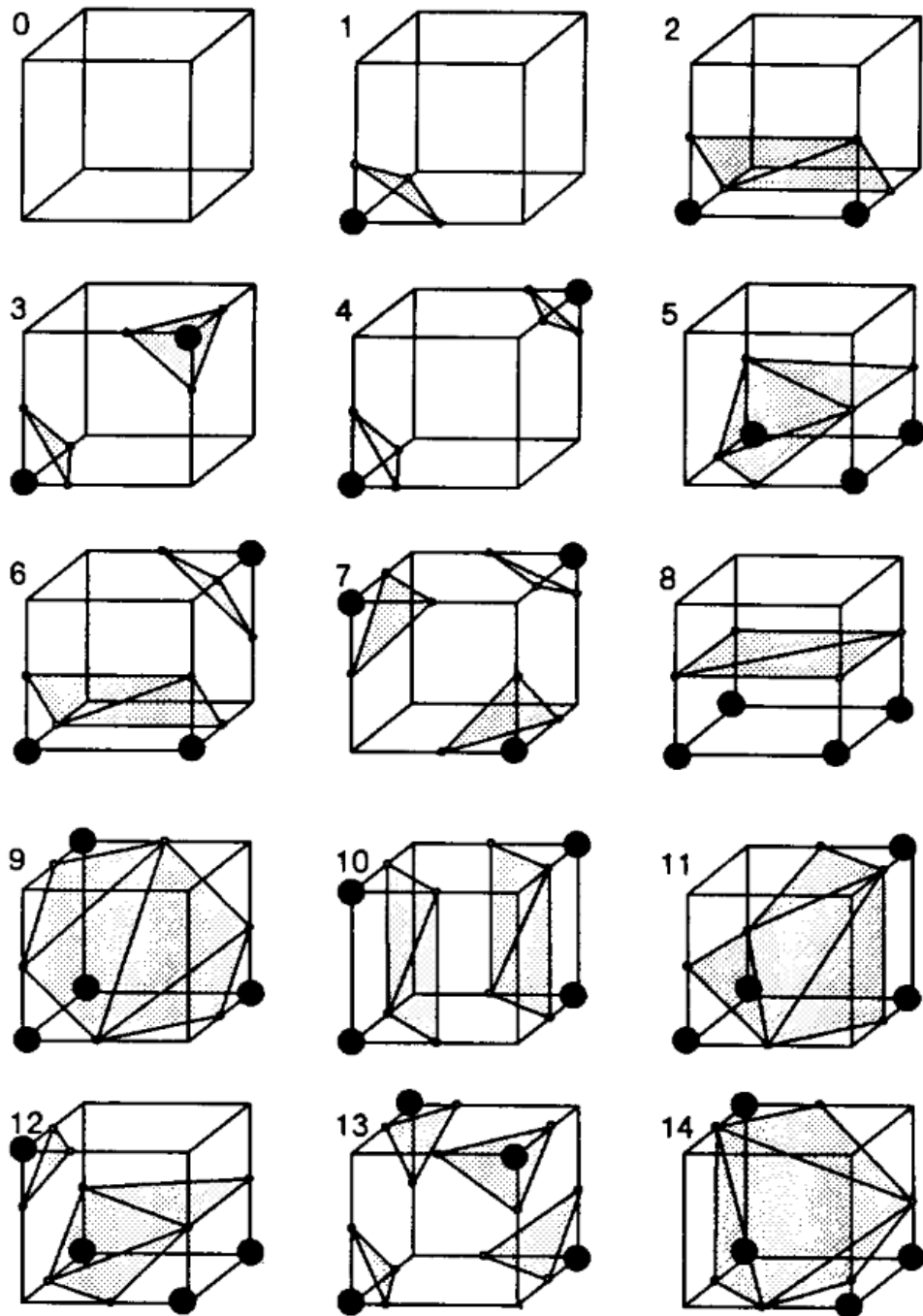


Figure 2.7: Originally published 15 combinations [31].

Chapter 3

Implicit Representations from Point-Clouds

Traditional 3D representations discretely parameterize signals as grids of voxels, point clouds, or meshes, while implicit neural representations continuously map the domain of the signal to a value via a neural network [48].

Multiple techniques allow representing a three-dimensional scene from a point cloud implicitly [46], [36], [6], [45], [33], [38]. This chapter further assesses some of the most promising along with possible evaluation metrics and datasets that can be used.

3.1 Object-scale Representations

Regarding the representation themselves, this section explores some of the most interesting methods that perform well on an object scale but are lacking for a real-world, large-scale scene.

3.1.1 Occupancy Networks

Occupancy networks [33] are a deep learning model that represents surfaces implicitly by the decision boundary of a classifier, with the architecture seen in Figure 3.1.

These models aim to predict the probability of occupancy at a certain Cartesian point in the three-dimensional space, defined by the following mathematical formula:

$$o : \mathbb{R}^3 \rightarrow \{0, 1\}, \quad (3.1)$$

This technique allows for generating the implicit representation either with a single view RGB image or with a point cloud with significantly better results in the latter, by employing different

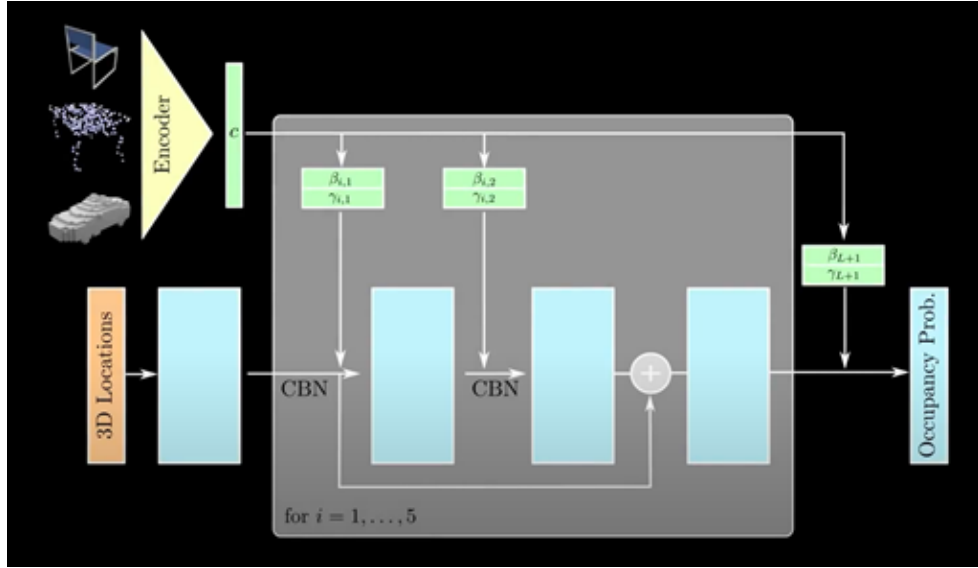


Figure 3.1: Architecture of occupancy networks [15]

encoders for each situation. Mesh generation can be performed using the Marching Cubes algorithm (see section 2.6), defining as a surface the decision boundary for the free/occupied space probability prediction.

3.1.2 DeepSDF

A different way to implicitly represent a 3D surface is through the use of a signed distance function or SDF [35] (see section 2.5). These implicit functions have a negative value within an object and a positive outside it, implying the surface as to where the value is zero.

DeepSDF [36] is a technique that learns shapes and can create an implicit representation of an object from its partial point cloud, completing unseen parts of its shape and with a small memory footprint. To achieve this it uses an auto-decoder architecture as seen in Figure 3.2 to output the predicted SDF value for each queried point.

However, it cannot be applied in this context as it does not have good results at scene scale [36].

3.2 Scene-scale Representations

3.2.1 Convolutional Occupancy Networks

Convolutional occupancy networks [38] are an improvement on the previously presented occupancy networks that utilize an encoder to transform the point cloud into features that are later projected into a volume using average pooling. The convolutional decoder then computes the resulting feature volume to aggregate local and global information as Figure 3.3 shows.

Table 3.1: Performance comparison on 3D reconstruction from point clouds on the ShapeNet [7] dataset, from [33]

	IoU	Chamfer-L1	Normal Consistency
3D-R2N2 [9]	0.565	0.169	0.719
PSGN [12]	-	0.144	-
DMC [28]	0.674	0.117	0.848
ONet [33]	0.778	0.079	0.895

This method also allows the encoder to project the features into one or multiple planes instead of a volume. This allows for higher resolutions but is worse at representing 3D information, and even though these are more computationally efficient and perform better when reconstructing objects, the volume encoder is best at scene-level reconstruction, even at similar GPU memory [38].

This method’s performance is significantly better than the original occupancy networks in the real-world ScanNet V2 dataset [11], with lower Chamfer-L1 and higher F-Score [38], as seen in Table 3.2.

3.2.2 DeepLS

Deep Local Shapes [6] utilizes SDFs locally in order to preserve significantly more detail in its reconstructions by dividing the space into a regular grid of voxels on which the signed distance function will be computed using learned local shape priors.

This method allows using different voxel sizes, altering the number of parameters, while having good results (Figure 3.4), which can allow reducing the inference time to meet the context’s constraints if necessary [6].

As for scene-scale performance on the Synthetic ICL-NUIM Dataset [19], DeepLS performs better than TSDF Fusion [10] in terms of both error and completion [6], as seen in Table 3.3.

3.2.3 SIREN

SIRENs [46] are a relatively simple multi-layer perceptron [14] that uses a periodic activation function, the sine. These networks are ideally suited for representing complex natural signals and their derivatives as Figure 3.5 shows.

To achieve this, however, a 5-layer MLP is trained on a point cloud for approximately six hours on a high-end GPU [46], making it not applicable in this context.

3.2.4 Semantic Scene Completion using Local Deep Implicit Functions on LiDAR Data

A different way to implicitly represent the three-dimensional structure of a scene is through Semantic Scene Completion, where the geometry and semantics of a scene are jointly estimated. The

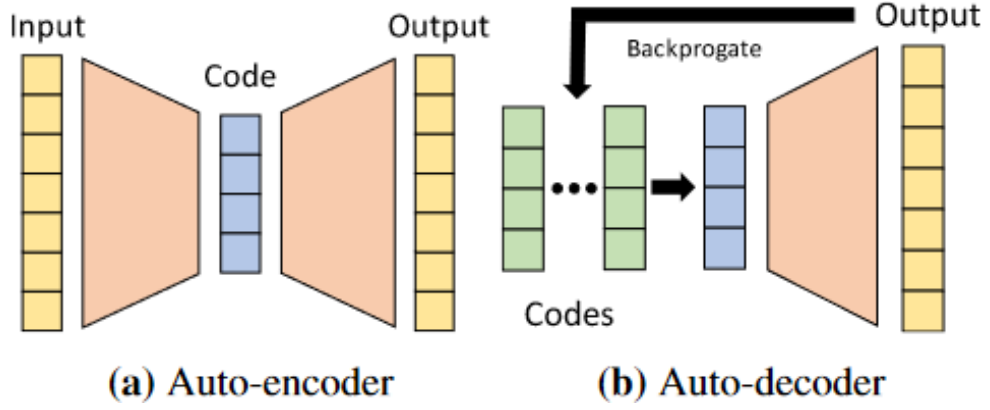


Figure 3.2: Auto-encoder architecture in a) and auto-decoder in b). [36]

paper Semantic Scene Completion using Local Deep Implicit Functions on LiDAR Data [41] explores this idea within this context, using a real-world driving dataset based on LiDAR’s voxelized point clouds called SemanticKITTI [3] to both train and test the model’s capabilities.

To achieve this, the model trains on accumulated voxelized LiDAR scans with semantic information and matches every point in space to the probability of belonging to each one of the $N + 1$ classes, including an additional free space class:

$$\mathbb{R}^3 \rightarrow [0, 1]^{N+1} \quad (3.2)$$

As Figure 3.6 illustrates, the network has various parts, starting with a PointNet [40] module that extracts features from a top-view grid of the input which are later fed to the convolutional encoder. This encoder is responsible for down-sample the features which will be later upsampled to create multiple resolution feature grids. Except for the highest resolution grid, which utilizes a 2x2 square with the 4 nearest cells to bi-linearly interpolate the features, the final module constructs the output from the feature grids in which the queried coordinates are in.

For the total loss function, it uses a weighted sum of the semantic loss, described as the cross-entropy loss between the classification output and the ground truth for all the non-free-space classes, the geometric reconstruction loss, that compares the occupied and free space with the ground truth using a binary cross-entropy loss, and finally, the consistency loss, that is implemented as the Jensen-Shannon divergence between the multiple probability distributions predicted by the local segmentation functions on the support region of a consistency point.

As for the network’s performance, it achieves state-of-the-art results in regards to occupancy with an IoU of 57.7%, and mIoU of 23.6% in the semantic completion task, and can be qualitatively evaluated with Figure 3.7, that compares the predicted continuous output with the input and ground truth points.

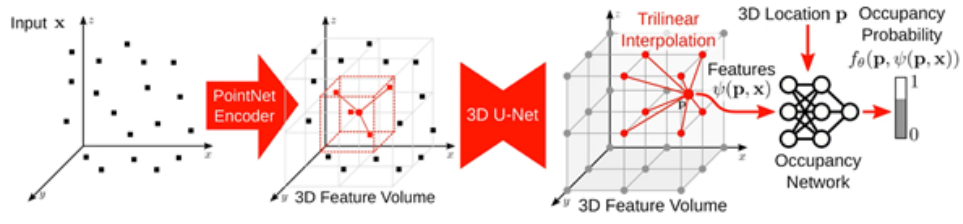


Figure 3.3: Architecture of convolutional occupancy networks [16]

3.2.5 SISC

Similarly to the project presented in subsection 3.2.4, Semi-supervised Implicit Scene Completion from Sparse LiDAR (SISC) [27] aims to predict both the 3D model of the scene and its semantic information through the use of a deep implicit representation. This paper also uses the real-world driving SemanticKITTI dataset [3], testing its performance on a large-scale scene in the self-driving context.

To achieve this, a network with the structure shown in Figure 3.8 is fed a voxelized lidar scan and extracts its voxel-wise shape embeddings, which after being trilinearly sampled to match the position of the queried points, are fed to a sine MLP that predicts a signed distance function for every queried position. To predict the semantic information of the scene there are two available options, referred to as SSC Option A and B, which work differently: while Option A is applied to the input occupancy volume, Option B is a sine MLP that receives the pointwise shape embeddings as input.

Performance-wise, this network has good results with an IoU of 51% for scene completion and a mIoU of 23.4% for the semantic task.

The paper having a publicly available code-base was a decisive factor for being used as the baseline for this dissertation, due to the very limited development time available.

3.3 Evaluation

3.3.1 IoU

Intersection over Union (IoU) is a metric used to evaluate the overlap of two areas or volumes. It is particularly interesting because it only considers the geometric reconstruction quality by using a binary free/occupied scene representation. [42]

3.3.2 mIoU

Also known as the Jaccard Index, mean intersection over union (mIoU) considers IoU of all semantic classes, without free space. It is defined by the formula:

Table 3.2: Performance comparison on the real-world ScanNet V2 dataset [11], from [38]

	Chamfer-L1	F-Score
ONet [33]	0.398	0.390
Point Conv [38]	0.316	0.439
SPSR [22]	0.293	0.731
SPSR (trimmed) [22]	0.086	0.847
CONet-2D ($3 * 128^2$)	0.142	0.776
CONet-3D (64^3)	0.077	0.886

$$mIoU = \frac{1}{C} \sum_{c=1}^C \frac{TP_c}{TP_c + FP_c + FN_c}, [42] \quad (3.3)$$

where TP_c are the true positives, FP_c the false positives and FN_c the false negatives for the predictions for class c .

3.4 Datasets

As for the dataset, it is required to contain point clouds in a real-world environment captured by a test vehicle, so it mimics the experience of an autonomous car.

Multiple options are available (Table 3.4), such as the nuScenes Dataset [5], the Argoverse Dataset [8], the Waymo Perception Dataset ([47]), Level 5 Open Data [24], KITTI-360 [29], and others, but SemanticKITTI [3] was chosen due to being the dataset used in the two papers that most closely relate to the goals of this thesis, which allows for direct comparisons of performance in the multiple experiments conducted.

This dataset is based on the odometry benchmark of the KITTI dataset and provides dense annotations for each scan of all sequences in the original dataset. It shows inner-city traffic, residential areas, and also highway and countryside areas. For the dataset generation, future laser scans were superimposed in a predefined region in front of the car for each frame, allowing the creation of pairs of inputs and targets for the semantic scene completion task, as seen in Figure 3.9. These scans are represented as a voxelized grid with a resolution of 0.2m and a volume of 256 x 256 x 32 voxels. [3]

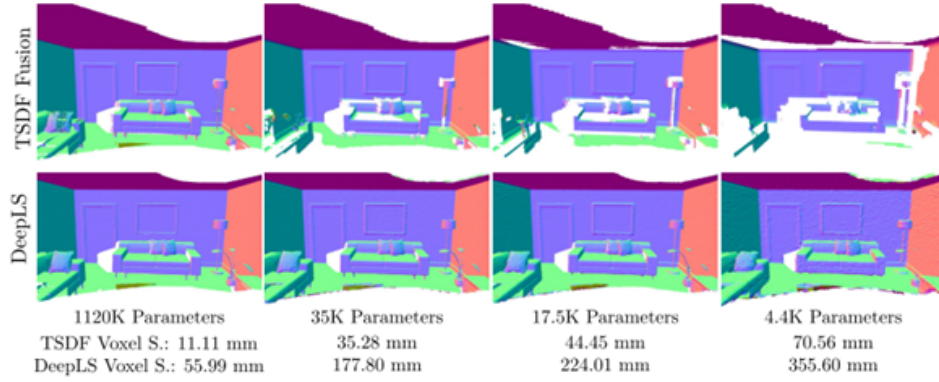


Figure 3.4: Qualitative analysis with a similar number of parameters on a synthetic scene [6]

Table 3.3: Performance comparison on 3D Scene Dataset [52]

Method	Burghers		Lounge		CopyRoom		StoneWall		TotemPole	
	Error	Comp	Error	Comp	Error	Comp	Error	Comp	Error	Comp
TSDF F. [10]	10.11	85.46	11.71	85.17	12.35	83.99	14.23	91.02	13.03	83.73
DeepLS	5.74	95.78	7.38	96.00	10.09	99.70	6.45	91.37	8.97	87.23

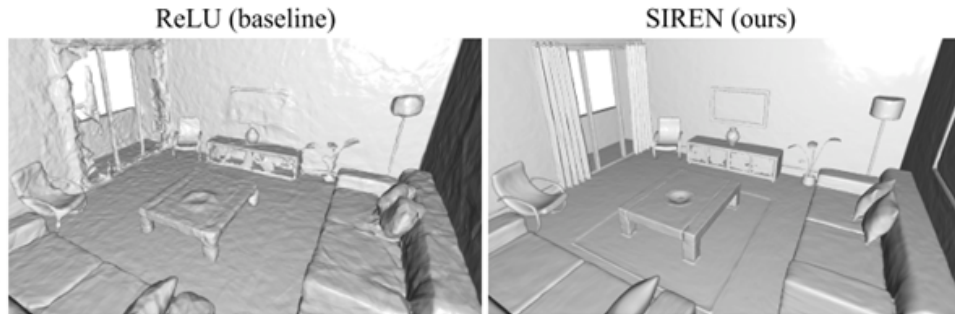


Figure 3.5: Qualitative analysis with ReLU activation function [46]

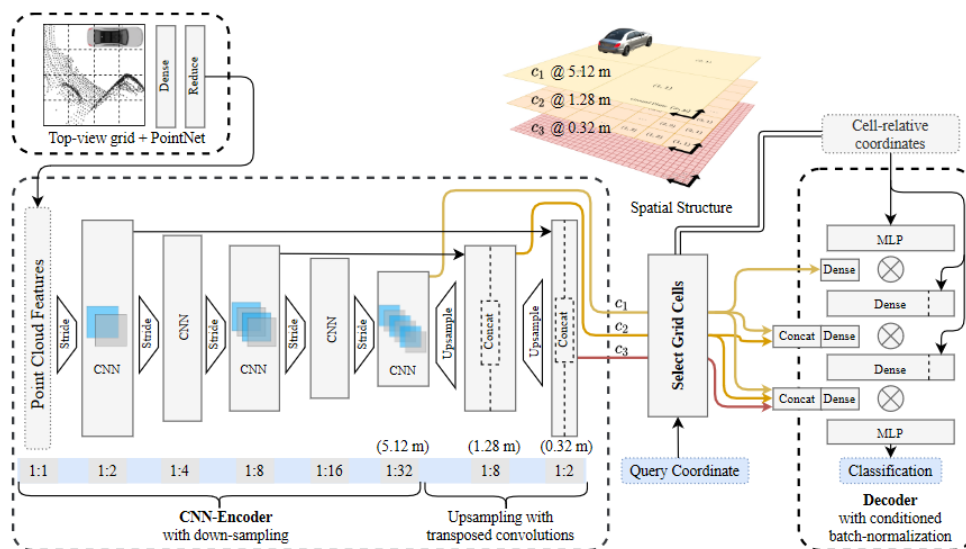


Figure 3.6: Network architecture of [41]

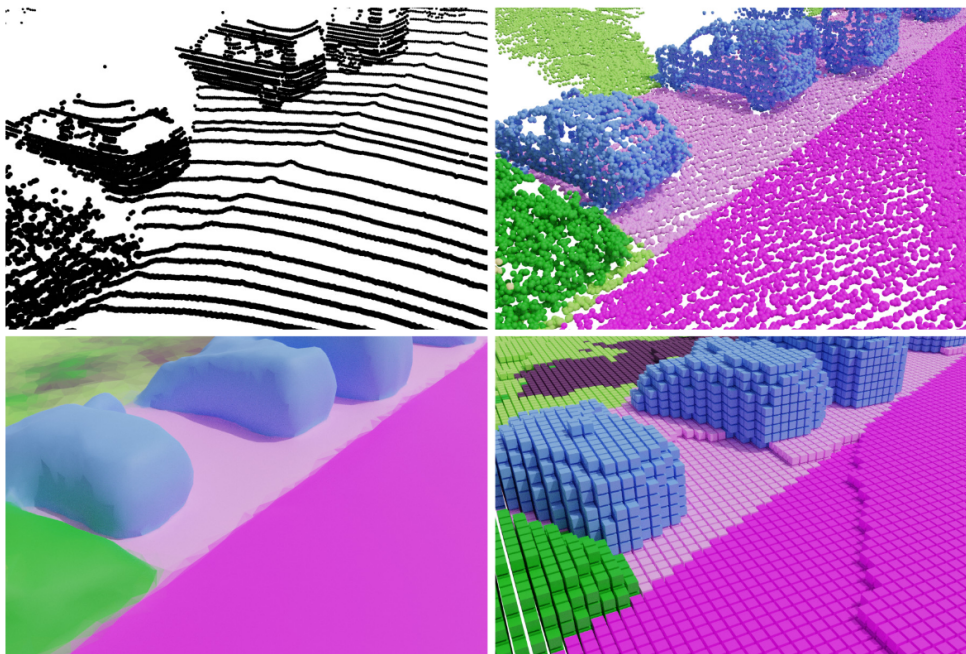


Figure 3.7: Left to right, top to bottom: Input points, ground truth accumulated points, mesh visualization of continuous output function, derived voxelization at 20 cm edge length. From [41].

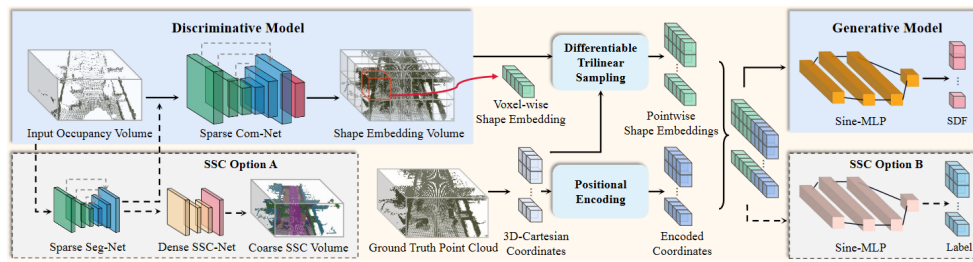


Figure 3.8: Network architecture of [27]

Table 3.4: Comparing multiple datasets (adapted from [29])

Dataset	2D Annotations			3D Annotations				
	#Smt. Img.	#Ins. Img.	Dense	#Smt. Pts.	#Ins. Pts.	FoV Azm.	FoV Plr.	#3D Bbox
Argoverse [8]	-	-	-	-	-	-	-	993K
Waymo [47]	-	-	-	-	-	-	-	12M
A*3D [39]	-	-	-	-	-	-	-	230K
KITTI [17]	200	200	yes	-	-	-	-	200K
ApolloScape [21]	144K	90K	yes	-	-	-	-	70K
nuScenes [5]	93K	93K	no	1.2B	78.9M	360°	40°	1.2M
A2D2 [18]	41K	41K	yes	387.1M	23.8M	60°	30°	43K
KITTI-360 [29]	2 * 78K	2 * 78K	yes	1.0B	172.4M	360°	120°	68K

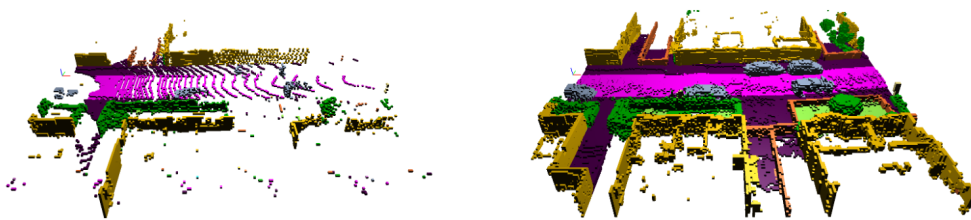


Figure 3.9: Left: Voxelized input scan with semantic information (for visualization). Right: Target output with completed and labeled scene. [3]

3.5 Discussion

As seen, there are multiple reasonable ways to represent 3D surfaces implicitly. Not all are applicable in this context, however. To be appropriate for an autonomous vehicle, the representation technique must be able to predict the environment's structure with a reasonably low inference time and have the ability to retain a good level of detail while performing scene completion on a real-world scale environment.

SIRENs [46], as presented in its paper, cannot be applied without prohibitively long inference times, having to be trained for every scene. DeepSDF [36] is only used to represent shapes being also not applicable in real-time. Occupancy Networks' [33] scene-scale performance is significantly inferior to that of Convolutional Occupancy Networks [38], and even their convolutional counterparts, which are capable of representing details at scene-scale, don't have inference time disclosed in its paper and aren't able to complete unseen parts of the scene. Another promising algorithm is DeepLS [6], which aims to perform at scene-scale with "orders of magnitude faster inference than global, object-centric methods like DeepSDF" [6], but it also will not support scene completion.

The only two viable options are the ones described in subsections 3.2.4 and 3.2.5. These use the same real-world driving dataset, named SemanticKITTI [3], and are evaluated for both scene-completion and semantic segmentation, achieving good results with the first option even reaching state-of-the-art IoU for the scene-completion task. Both these architectures could be used in this dissertation but because only SISC (see subsection 3.2.5) has publicly available code, it was chosen as the starting point of this dissertation, due to the time limitations.

In the dissertation, the next chapter will further explore the chosen technique and outline the experiments that were conducted to study its impact.

Chapter 4

Methodology

With the state-of-the-art and important concepts presented, one needs to go back to the problem definition and objectives of this work: to study the implications of applying a state-of-the-art deep implicit representation as part of the solution to perception problems in the context of autonomous driving.

The first step to achieve this goal is to choose the representation method used, and there are three main options: to create a new representation technique, re-implement an architecture described in a paper, or use one with an open repository. Due to the significant time constraints of this dissertation, the third option was the only feasible one. Next, to research the implications of the representation on perception tasks, one was chosen to run experiments. Ideally, the deep-implicit-representation would be implemented in more tasks, but considering the duration of the dissertation, only one was experimented with: LiDAR-RGB sensor-fusion.

4.1 The Representation

The model chosen was SISC [27], as presented in subsection 3.2.5, for its proven ability to handle real-world, large-scale, sparse LiDAR scans while also having its goals aligned with this dissertation by outputting the implicit representation through signed distance functions and having the ability to output semantic information: one of the perception tasks for autonomous vehicles. Another important upside of this project is that its code is publicly available, significantly speeding up the development of this dissertation by simplifying the initial process of matching the paper's results.

4.1.1 How it works

This subsection further compliments the description present in the subsection 3.2.5.

This network can be decomposed into two main parts: the discriminative model and the generative model. The first one employs a convolutional neural network to map a voxelized LiDAR

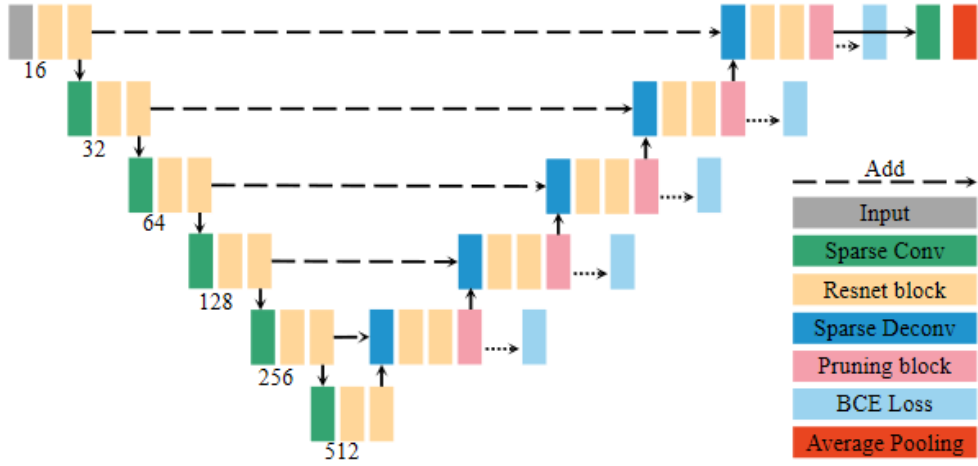


Figure 4.1: Network structure of Com-Net, from [27]

scan into a shape embedding volume. This convolutional neural network, named Com-Net, predicts these shape embeddings via a shape completion process and is composed of an encoder with convolutional and residual blocks, and a decoder that uses generative deconvolutional blocks to generate new voxels, as seen in Figure 4.1. This decoder architecture will, however, eliminate the sparsity of the voxels so, for that reason, a pruning block is present to eliminate redundant voxels. The pruning block has a convolutional layer that predicts whether a voxel should be pruned, and is supervised by the loss function 4.1. The referred shape embeddings later go through a differentiable trilinear sampling module that matches every queried point to an embedding by using trilinear interpolation to sample from the eight nearest voxels. To do this in a differentiable way is important so that the loss gradients can flow back to the shape embeddings and to the discriminative model, allowing the training of both discriminative and generative models cooperatively. Queried point coordinates are positionally encoded in order to represent more geometric detail, having this technique been proven effective in neural rendering [34]. As for the generative model, it uses a sinusoidal network, as shown in subsection 3.2.3, which has the sine as an activation function for the implicit representation. With 3 layers of 256 hidden features each, it predicts the value of the signed distance function from the concatenated encoded coordinates and pointwise shape embeddings. The SDF loss function is as described in 4.2.

The loss function that supervises this project is divided into multiple parts with different weights. The pruning block loss, in the generative model, is defined by the binary cross entropy, with equation:

$$\mathcal{L}_{com} = -\frac{1}{m} \sum_{i=1}^m \frac{1}{n_i} \sum_{j=1}^{n_i} [y_{i,j} \log(p_{i,j}) + (1 - y_{i,j}) \log(1 - p_{i,j})], \quad (4.1)$$

where m is the count of supervised blocks, n_i the count of voxels in the i -th block, and y and p the true and predicted existence probability for the respective voxel. For the SDF supervision,

however, the loss is more complex with four separate parts and respective weights. The equation that defines it is the following:

$$\begin{aligned}
\mathcal{L}_{sdf} = & \lambda_1 \int_{\Omega_0} |||\nabla_x \Phi(x, e)|_{e=\zeta(x, \Omega_1)}| - 1||dx \\
& + \lambda_2 \int_{\Omega'_1} (1 - \langle \nabla_x \Phi(x, e)|_{e=\zeta(x, \Omega_1)}, n(x) \rangle) dx \\
& + \lambda_3 \int_{\Omega'_1} ||\Phi(x, e)|_{e=\zeta(x, \Omega_1)}||dx \\
& + \lambda_4 \int_{\Omega'_3} \psi(\Phi(x, e)|_{e=\zeta(x, \Omega_1)}) dx,
\end{aligned} \tag{4.2}$$

and here λ_{1-4} are the constant weights of each component and $\langle \cdot, \cdot \rangle$ the cosine similarity. The full loss function can be defined as:

$$\mathcal{L} = \mathcal{L}_{sdf} + \lambda_5 \mathcal{L}_{com}, \tag{4.3}$$

where λ_5 is the constant weight of the $\mathcal{L}_{com}$ component.

As a means to visualize the scene outputted by the model, the project employs the Marching Cubes algorithm described in section 2.6 to define as surfaces the regions with the absolute signed distance function below a certain threshold. Previously referred to in the section 2.5, an implicit surface representation through the use of SDFs suggests that a Cartesian coordinate belongs to a surface when its signed distance function is equal to 0, however since the model is not mathematically exact, multiple thresholds are employed to define a surface, with different performances. The referred thresholds are between 0.015 and 0.002. Mathematically, an SDF output x is considered a surface with a threshold T if:

$$|x| < T. \tag{4.4}$$

This method allows for the extraction of a mesh from the model output, which is later compiled to a Polygon File Format (PLY) file that can be viewed using most 3D visualizers.

Regarding the dataset division, it contains 22 sequences that are split into three sets: the training set has sequences 0 to 7 and 9 to 10, the validation set with the sequence 8, and lastly the test set contains sequences 11 to 21. The sequences present in the test set, however, don't have the necessary files to perform local evaluation, being intended to be used to generate outputs that are evaluated in a test server. This would make the process more time-consuming and is mostly unnecessary for a proof of concept, and for that reason, evaluation was performed on the validation set: sequence 8. The evaluation logic consists of calculating the IoU (see section 3.3) for the set with the referred thresholds. Similar to the visualization logic, for the IoU calculations, the

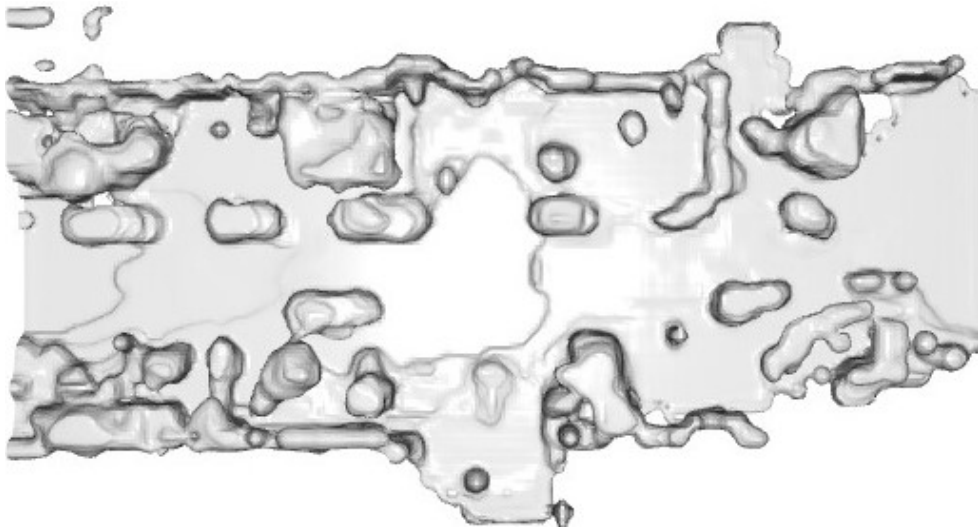


Figure 4.2: Top-down view of the model output for frame 545 from sequence 8, trained until epoch 50.

thresholds are used to determine whether a voxel is considered a surface, classifying every voxel in the grid in a binary way.

During training the input set goes through data augmentation, rotating it along the Z-axis by a random amount between forty-five and minus forty-five degrees. This value can be easily changed through the configuration file but has been kept true to the original as proposed by Li et al. in [27].

4.1.2 Visualizer

With the goal of converting these 3D models stored in files to an animated video that would make the outputted scene easier to understand and more interesting to watch in motion, a new visualization script was created.

Employing the Open3D library [53] to load, render and store each model in a file due to its robustness and comprehensive visualization capabilities, and the OpenCV library [4] to transform the stored images into a video with the desired frame rate. As an example, the colorless top-down view of the model-generated scene for the frame with id 545 of sequence eight from SemanticKITTI can be seen in Figure 4.2. To create this image, the model used was trained until epoch 50.

4.1.3 Issues with the representation

The use of this visualization made apparent some issues with the model. As can be seen in Figure 4.3, continuing to train the model beyond a certain point leads to overfitting artifacts in the areas that are most commonly occupied by buildings and obstacles: the sides of the road. For both the

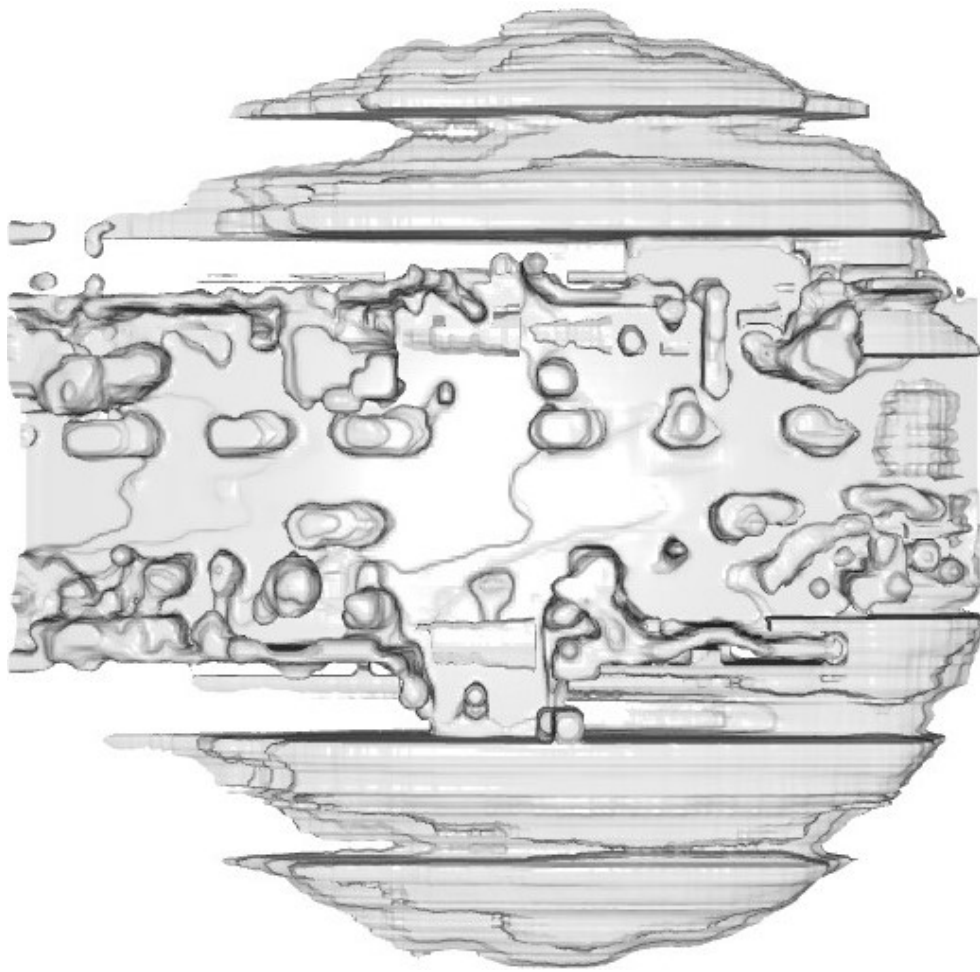


Figure 4.3: Top-down view of the model output for frame 545 from sequence 8, trained until epoch 200. Overfitting artifacts are visible on both sides of the road: the top and bottom of the image.

image at epoch fifty 4.2 and at epoch two-hundred 4.3, the models that outputted the representations were different checkpoints from the same training session, and both the scene generation logic and local visualization script were run with the same parameters: threshold set at 0.015.

Even though this problem could be mostly visually fixed by lowering the output threshold, it shows that the model is overfitting which needs further investigating and led to the changes described in the subsections 4.1.4 and 4.1.5.

4.1.4 Validation loss

After being able to locally replicate the original results, it was noticed that the validation metric used, IoU, was very unstable and that there was no validation loss. After further investigating why it was so unstable, it was discovered that the IoU calculation was only being done for a single frame per evaluation and that the frames were sequential, meaning that the frame used was not always the same.

This led to the first addition to the original model: saving the validation loss like what was being done with the training one. This enables better live metrics while the model is still being trained, allowing for example to quickly understand whether some experiment has potential or its training session can be stopped early, and to more easily detect overfitting and other possible issues with the model.

To achieve this the same logic used for computing the training loss was applied to the entire validation set. This did, however, significantly increase the training time, leading to changes in the number of frames used. The logic that allows for a specific step in the data-loader was implemented, and, through the configuration file, it is possible to only use a frame in every x number of frames, which allows for easy tuning of the training time dedicated to validation while keeping the sub-sequence diverse and deterministic. The step used was twelve, meaning that the same sixty-seven frames of the total eight-hundred and fifteen were used for every validation.

4.1.5 Dropout

Supported by a reverse in the trend of the (not very stable) SDF component of the validation loss, dropout layers were added to the generative model with the goal of preventing overfitting which could fix the visualization artifact and improve the network's performance (see subsection 2.3.1).

To this end, three dropout layers were added, one before each hidden layer of the generative model. Multiple tests were run with different dropout probabilities and with and without data augmentation. The first models tested had a dropout probability of 20% and 40% and no data augmentation. Since the validation loss graphs showed a slight advantage to the model with higher dropout probability, the following tests had moderately higher dropout probabilities, with data augmentation models using 30% and 50% dropout probabilities.

4.2 DIR and Sensor Fusion

The novelty this dissertation explores is to incorporate sensor-fusion (see section 4.2) in a deep implicit representation. Since the original architecture employs only sparse LiDAR data to create its scene representation, and with good results, it is an interesting direction of research to investigate whether to supply the model with more data from different sensors, like the RGB information from onboard cameras, makes a significant and positive difference to the model's performance and whether the model could represent more than just the scene geometry.

Roldão et al. state that "the literature consensus is that middle fusion is highly efficient for SSC" [42]. This method would be ideal for this project, possibly allowing the full RGB information to be used by a new model to infer on, and fused with the original architecture at the shape-embeddings stage (see Figure 3.8) but due to the extra complexity of adding a new model to process this information instead of, for example, use the same architecture with extra channels in the case of data-fusion, it would not be feasible to complete this experiment in the available time. For that reason, early fusion was chosen and added to the model.

Three different experiments were run, with different goals. The simplest one was to simply add RGB data to the model and compare its performance with the original, while the most complex and interesting also expected the model to output color information, attempting to create a colored 3D model of the real-world scene. The third experiment consisted of enabling the 'SSC Option B' module (see subsection 3.2.5) in the architecture and studying the impact of sensor fusion in the scene segmentation task. The next subsections will further detail the steps taken and the changes that were necessary in detail.

4.2.1 Data preprocessing

In order to fuse the originally used LiDAR data with new RGB information, we need to match occupied voxels with the color information of the matching pixel for both the sparse input and completed ground truth.

The first step to achieving this is to retrieve the matching colored images. Since the dataset used is [3] and it does not contain images, these must be fetched from the KITTI Odometry Benchmark [17], the original dataset that SemanticKITTI is based on. This is relatively simple since these datasets have matching folder structures, sequences, and frame ids.

Now having both the matching images and point clouds one needs to match every occupied voxel to a pixel in the image and extract its RGB values. To do this, the calibration files available in the original KITTI Odometry Benchmark need to be used. These allow calculating the pixel position on a given camera from the three-dimensional Cartesian coordinates of the occupied voxel relative to the car. When a point does not fit into the camera's field of view, its RGB information is set to zero and marked with a zero in a binary fourth column to allow the model to distinguish between points that are not visible and ones in a black surface. To ensure the correctness of this step, the occupied and visible voxels were projected to an image that was later visualized, as can be seen in Figure 4.4. For the input voxel grid, this is a straightforward process as there is no occlusion to consider since all occupied voxels are visible to the camera as both sensors are closely placed, with the camera used being about 27 centimeters away from the LiDAR. For the ground truth grid, however, this assumption does not apply. A simple solution is to ignore the issue and retrieve the RGB values for the first surface that intercepts, with a possible improvement being to first detect which voxels are currently occluded and employ a similar idea to the one that led to the ground truth creation: utilize the moving nature of the car and go through the other frames in the sequence searching for an image where the voxel is visible, utilizing the RGB values in that image. Time constraints only allowed the implementation of an in-between solution: occluded voxels were detected as marked as such, not contributing to the loss computation, but no future frames were used to try capturing the RGB information at a currently occluded position.

This was all performed offline with the idea of saving time during data-loading in both training and inference. The color information of every occupied voxel was stored in a file, separately for the sparse input and completed scene, to be later read by the model's data-loader more efficiently.



Figure 4.4: Occupied voxels projected into the image as black dots for debugging purposes.

4.2.2 Architecture changes

In order to feed the extra color information to the model some changes needed to be made. For the experiment without RGB output, the only necessary change was to increase the input channels from one to 5: instead of only feeding binary information to the sparse input tensor representing the occupancy of a voxel, four channels were added with the first three representing the normalized RGB values of a voxel, and the last one informing whether the voxels was visible to the camera being one if true and zero otherwise.

Regarding color output, three output channels needed to be added to the generative model for the red, green, and blue information at each position. To ensure the RGB output is normalized, a sigmoid function is applied to the three added channels.

For the scene segmentation task, module 'SSC Option B' from the original SISC architecture (see Figure 3.8) was used without changes. Option B was used instead of A because the latter is independent of the intermediary representation logic: it uses the occupancy volume as input instead of the shape embeddings and its architecture differs significantly from the Generative Model, while Option B is similar but with different output.

For some experiments, dropout layers were also added as described in subsection 4.1.5 and using a dropout probability of 30%.

4.2.3 Loss

In order to supervise the RGB output of the model a new loss component needed to be added. This component aims to minimize the difference between the red, green and blue predictions of a certain visible point with its ground truth. Mathematically it can be defined as:

$$\mathcal{L}_{RGB} = \frac{1}{n} \sum_{i=1}^n \frac{|y_{iR} - \hat{y}_{iR}| + |y_{iG} - \hat{y}_{iG}| + |y_{iB} - \hat{y}_{iB}|}{3} y_{i\text{visible}}, \quad (4.5)$$

with n being the number of queried points, y_{iR} , y_{iG} , and y_{iB} the ground truth values for each of the RGB components in a point i , $\hat{y}_{iR}$, $\hat{y}_{iG}$ and $\hat{y}_{iB}$ the predicted RGB component values for point i , and $y_{i\text{visible}}$ a binary value representing whether the 3D point i is considered visible in the ground truth: its pixel coordinates are within the camera's field of view. This last binary part of the component allows to only apply supervision on points with ground truth RGB values.

The updated complete loss function now becomes:

$$\mathcal{L} = \mathcal{L}_{sdf} + \lambda_5 \mathcal{L}_{com} + \lambda_6 \mathcal{L}_{RGB}, \quad (4.6)$$

with λ_6 being the weight of the RGB component of the loss function and $\mathcal{L}_{RGB}$ the new component that supervises the color prediction.

4.2.4 Data Efficiency

One other experiment that was made regarded the data efficiency of sensor fusion for the representation model. The premise to investigate is whether feeding data from extra sensors to the model lessens the need for more driving sequences.

With this goal in mind, two training sessions were run until epoch fifty using only half the original training set, one with only occupancy input and the other with RGB information as well. To do this, sequences 0 to 4 were used to train the model, instead of the original 0 to 7 plus 9 and 10. Validation ran on sequence 8 similarly to all other tests.

Chapter 5

Results and Discussion

This chapter describes the results obtained in the aforementioned experiments, both quantitatively and qualitatively, starting with the changes made to the original model and then regarding sensor-fusion.

5.1 Representation with LiDAR Data

5.1.1 Baseline

The baseline for this project was set by replicating the original results. As there was no seed set, the results have a margin of error due to the stochastic nature of the training of neural networks. To achieve this, the hyperparameters used were the default ones in the configuration file, as exposed in Table 5.1. In this table, G refers to parameters regarding the Generative model and D the Discriminative model (see subsection 3.2.5). These parameters were also used in the coming experiments.

Table 5.2 shows the obtained results at epoch 50, 100 and 200, for each of the evaluated thresholds. Here we can observe that results don't improve much from epoch 50 to 200, even deteriorating with thresholds above 0.01.

Qualitative results can be extracted with the newly created visualizer (see subsection 4.1.2) and are present as Picture 4.2 for epoch 50 and 4.3 for epoch 200.

From the aforementioned pictures, extracted with a threshold of 0.015, the problems referred in subsection 4.1.3 are clear. The table 5.2 corroborates this, with a decrease in the IoU score at the most permissive threshold, 0.015, as the training continues further. This phenomenon is already present at epoch 100, even though less pronounced than at further epochs, as Picture 5.1 portrays.

Table 5.1: Parameters used to replicate original paper results.

Parameter	Value
Batch size	2
Augmentation angle	45°
Non-linearity D	Elu
Non-linearity G	Sine
Width G	256
Depth G	3
Scale size	4
Shape dimension	256
Sample strategy	Trilinear

5.1.2 Validation Loss

Motivated by the artifacts made apparent through visualization and previously referred in subsection 4.1.4, the first addition to the original project was to calculate and store the model’s validation loss during training. Figure 5.2 shows the original validation metric: IoU, while figure 5.3 portrays the new validation loss. As we can see through these images by looking at the graphs with the y axis scale in mind, the original metric, partly by being calculated for only one frame at a time and a different one for most validation runs, is significantly more unstable than the validation loss calculated as described in subsection 4.1.4. This loss is, however, not as stable as desired. Increasing the batch size was considered but it could not be applied due to memory constraints.

5.1.3 Dropout

While trying to minimize the overfitting artifacts described in subsection 4.1.3, dropout (see subsection 2.3.1) was applied to the Generative model as detailed in subsection 4.1.5. Two experiments were conducted with data augmentation and different dropout probabilities, 30% and 50%, and two more without data augmentation using probabilities of 20% and 40%, with the respective validation losses present in Figure 5.4 and evaluations at epoch 50 compared to the baseline at the same epoch shown in 5.3.

In the experiments without data augmentation, the validation loss graphs showed that the models’ loss started to increase significantly very early in the training session. The results table exhibits an unstable but performing model with a dropout probability of 40%, especially when compared to the one using a 20% probability, which is more stable but has a lower IoU at the best threshold. For the experiments with augmentation, the loss graph shows better performance for the 30% experiment, and while the table 5.3 gives the 50% experiment the edge at some thresholds, like 0.006, it is overall less stable. For that reason and also considering that the original paper showed significantly better results for the models with data augmentation, only the 30% experiment’s training continued until epoch 200, with the final results compared to the baseline in the table 5.4. It shows very stable results at the different thresholds with the most permissive one having positive results, unlike the baseline that has seen a significant drop in this metric due to the artifacts present at a

Table 5.2: Replicated results: IoU score for each threshold.

Epoch	0.015	0.012	0.011	0.01	0.008	0.006	0.004	0.002
50	49.5	49.3	49.2	49.1	48.9	48.6	48.3	47.7
100	44.9	49.8	50.5	50.9	51.0	51.0	50.9	50.7
200	38.5	45.9	48.1	49.8	51.3	51.3	51.2	51.0

threshold of 0.015. Figure 5.5 corroborates this, showing no artifact at epoch 200 when dropout is applied.

However, since implementing dropout failed to increase the evaluation metric and has comparable results at epoch 200 with the baseline at epoch 50, when artifacts are not yet present, in further experiments instead of training the models until epoch 200 with dropout, these are instead conducted without dropout and stopped at epoch 50 with the goal of saving computing resources.

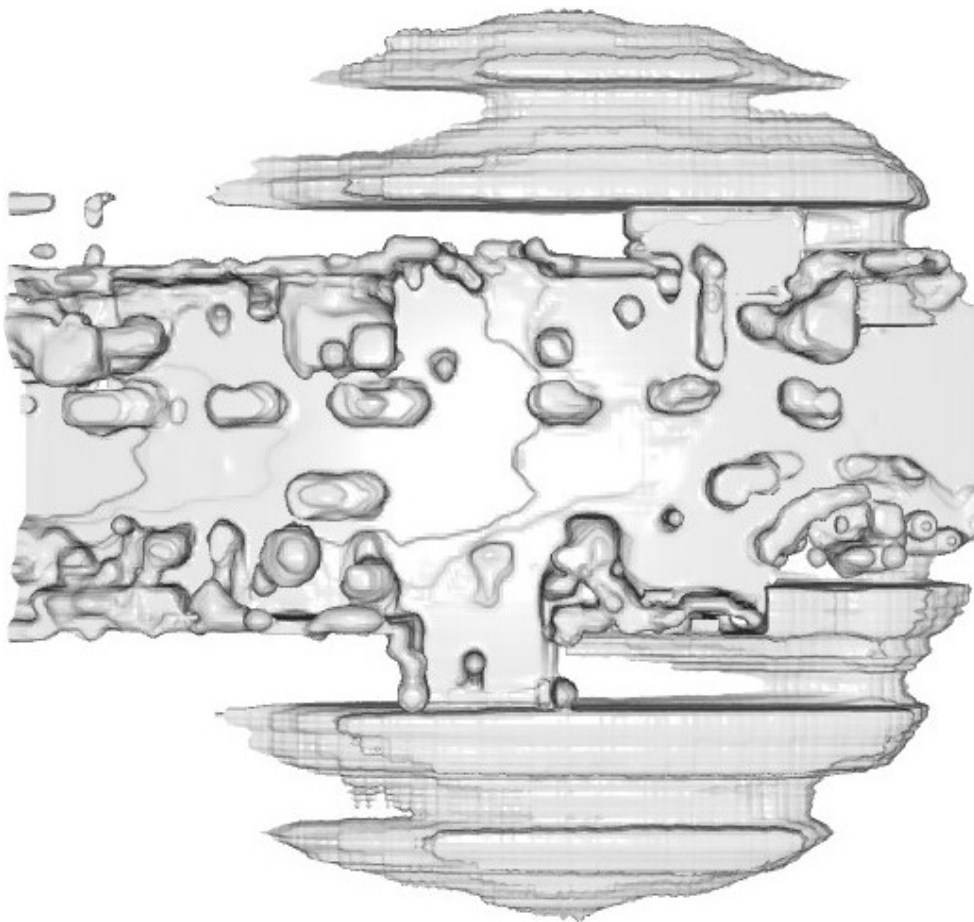


Figure 5.1: Top-down view of the model output for frame 545 from sequence 8, trained until epoch 100. Overfitting artifacts are visible on both sides of the road: the top and bottom of the image.

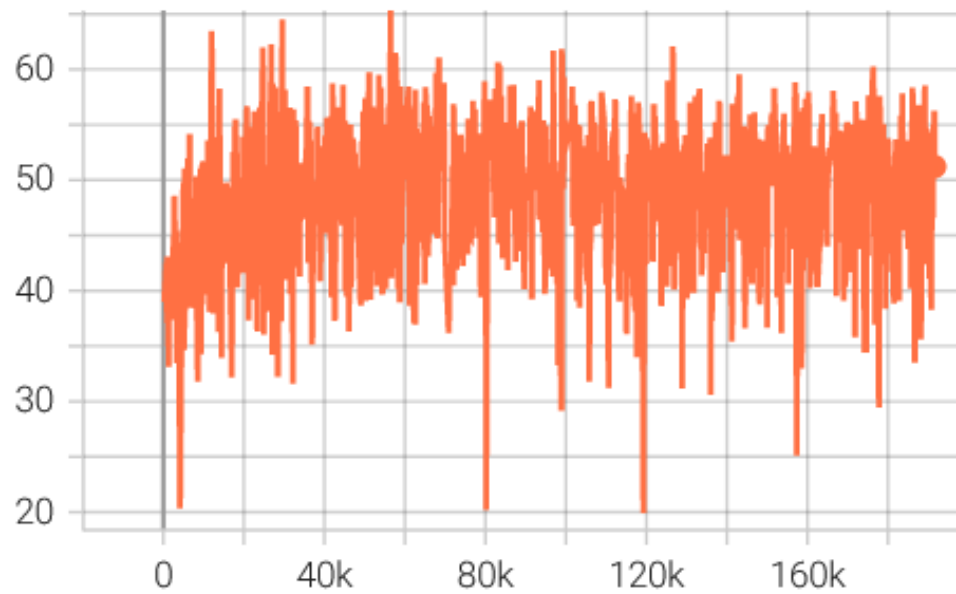


Figure 5.2: Single frame IoU used as model validation, trained until epoch 200. Value in y and number of steps in x.

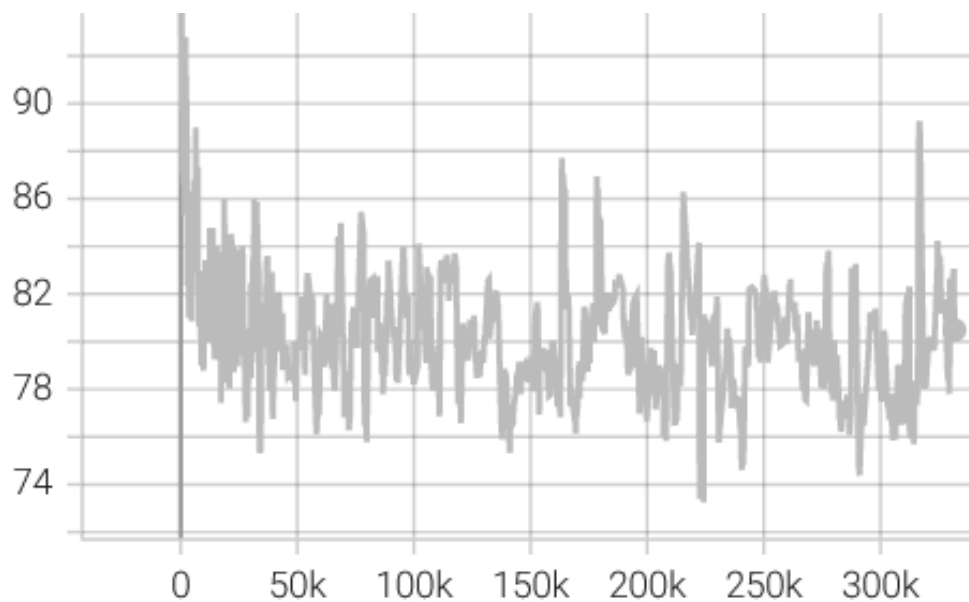


Figure 5.3: Newly implemented model's validation loss, trained until epoch 195. Value in y and number of steps in x.

Table 5.3: IoU score for each threshold for the dropout experiments and the baseline at epoch 50.

Epoch	Data Augmentation	Probability	0.015	0.012	0.011	0.01	0.008	0.006	0.004	0.002
50	Yes	0%	49.5	49.3	49.2	49.1	48.9	48.6	48.3	47.7
50	Yes	30%	47.5	47.3	47.2	47.0	46.8	46.4	46.2	45.9
50	Yes	50%	45.7	46.7	46.8	47	47.6	48.1	33.8	0.2
50	No	20%	48.5	48.6	48.6	48.6	48.6	48.5	48.4	48.3
50	No	40%	45.7	47.2	47.5	47.6	47.6	49.4	49.4	28.6

Table 5.4: IoU score for each threshold for the chosen dropout experiment and the baseline at epoch 200.

Epoch	Probability	0.015	0.012	0.011	0.01	0.008	0.006	0.004	0.002
200	0%	38.5	45.9	48.1	49.8	51.3	51.3	51.2	51.0
200	30%	49.5	49.4	49.4	49.4	49.2	49.1	49.0	48.9

5.2 Sensor-Fusion with RGB and LiDAR Data

5.2.1 Occupancy

To infer how sensor fusion (see section 2.4) affects the geometric reconstruction results, the experiments described in section 4.2 were conducted.

Table 5.5 describes the experiments' results, while also comparing them to the previously set baseline. It shows that adding RGB information to the input contributes to an earlier appearance of the overfitting artifacts, as shown by Figure 5.6, while also increasing the IoU at less permissive thresholds. Adding color output, however, seems to lessen the overfitting issues but also decreased the IoU at most tested thresholds when compared to the baseline and the color-less output experiment. Figure 5.9 shows the smaller overfitting artifacts when compared to 5.6.

This led to further experimentation with sensor fusion and dropout regularization. Table 5.6 shows results overall lower than the equivalent experiment without dropout (see table 5.5) and on par with the baseline, with the color output experiment having lower occupancy results. However, when considering that adding dropout to the baseline also slightly lowered its quantitative performance at this epoch, comparing these results with the dropout baseline gives this approach a very small advantage for the best threshold for each experiment and more stable results. Figures 5.7 and 5.10 confirm that there are no artifacts present at this stage when using dropout.

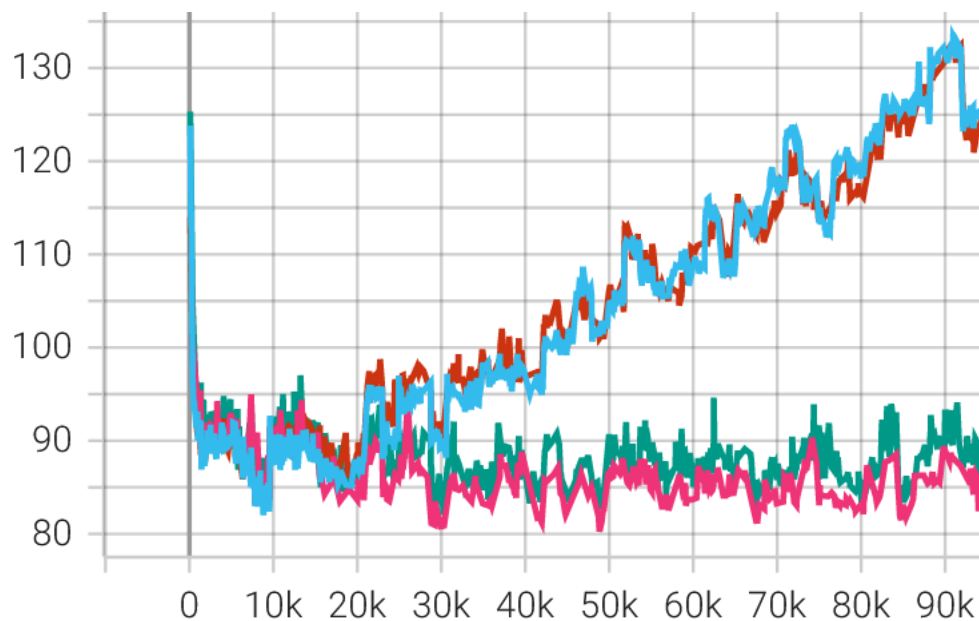


Figure 5.4: Validation loss until epoch 50 for models with data augmentation and dropout probabilities of 30% (pink) and 50% (green), and no data augmentation and dropout probabilities of 20% (blue) and 40% (red).

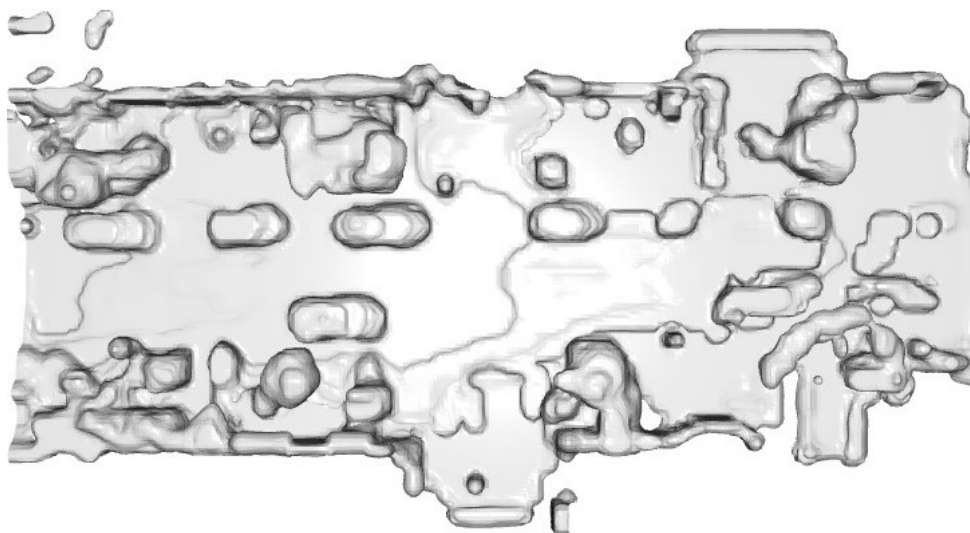


Figure 5.5: Top-down view of the model output for frame 545 from sequence 8, trained until epoch 200 with dropout probability of 30% (see subsection 4.1.5).

Table 5.5: IoU score for each threshold with sensor fusion (see section 4.2) at epoch 50.

Input Data	Output	Epoch	0.015	0.012	0.011	0.01	0.008	0.006	0.004	0.002
LiDAR	Occupancy	50	49.5	49.3	49.2	49.1	48.9	48.6	48.3	47.7
LiDAR + RGB	Occupancy	50	42	47.4	48.6	49.2	49.2	49.1	48.9	48.6
LiDAR + RGB	Occupancy + RGB	50	46.8	48.7	48.7	48.6	48.4	48.2	47.9	47.3

Table 5.6: IoU score for each threshold with sensor fusion (see section 4.2) and dropout (see subsection 4.1.5) at epoch 50.

Input Data	Output	Dropout Probability	Epoch	0.015	0.012	0.011	0.01	0.008	0.006	0.004	0.002
LiDAR	Occupancy	0%	50	49.5	49.3	49.2	49.1	48.9	48.6	48.3	47.7
LiDAR	Occupancy	30%	50	47.5	47.3	47.2	47.0	46.8	46.4	46.2	45.9
LiDAR + RGB	Occupancy	30%	50	47.8	47.8	47.7	47.7	47.5	47.3	47.1	46.9
LiDAR + RGB	Occupancy + RGB	30%	50	47.5	47.4	47.4	47.4	47.3	47.1	47.0	46.8

Table 5.7: mIoU score for each threshold with sensor fusion (see section 4.2) in the semantic scene completion task.

Input Data	Epoch	0.015	0.012	0.011	0.01	0.008	0.006	0.004	0.002
LiDAR	50	16.5	16.8	16.8	16.8	16.8	16.8	16.8	16.7
LiDAR + RGB	50	16.2	17.0	17.0	17.0	17.0	17.0	17.0	17.0

5.2.2 Color Output

Regarding the color outputting model described in subsection 4.2.2, the RGB component of the loss function detailed in subsection 4.2.3 is visible as Figure 5.8. This graph, however, shows the values after multiplying with the constant weight of 200. Even though the graph is unstable, one can see a downward trend as is expected.

Qualitatively, Figure 5.9 shows worse than desired results, with the colors not accurately represented. Directly observable sections of the road are mostly white while the most part of the scene is dark green and brown. The original picture from the RGB camera can be seen in Figure 5.11 and when comparing it to the predicted image, we can see that due to reflections from the sun, the road looks mostly white as was predicted as does the first light gray car, but the remainder of the image’s colors are mostly left unrepresented: the red cars don’t look different to the buildings and bushes, trees, and buildings look mostly dark green and brown. This shows us that the model is capable to differentiate and represent luminance differences, but struggles to reproduce color. Overfitting artifacts are also visible on the top and bottom of the image, even though these are smaller than previously observed.

However, when applying dropout to the Generative model with a probability of 30% the artifacts observable in Figure 5.9 are no longer visible. Even though the previously referred Table 5.6 shows that dropout lowers the model’s IoU, qualitatively the Figure 5.10 shows a more pleasing result, with no visible artifacts and comparable representation quality.

5.2.3 Semantic Segmentation

For the semantic scene completion task previously referred to in section 4.2, sensor fusion once again proved effective with an increase of mIoU at all but the largest threshold.

Table 5.7 confirms this, with a moderate but consistent increase of 0.2% mIoU across most thresholds.

Considering that this version of the architecture infers both semantic information and scene geometry simultaneously, it is also interesting to look at the occupancy results with and without RGB information. Table 5.8 shows that the sensor fusion approach performs worse than the baseline at larger thresholds but is better with smaller ones.

Comparing the visualization for both approaches, Figures 5.12 and 5.13 show mostly similar scenes, with both already having visible overfitting artifacts at epoch 50 and these being more pronounced in the sensor fusion approach.

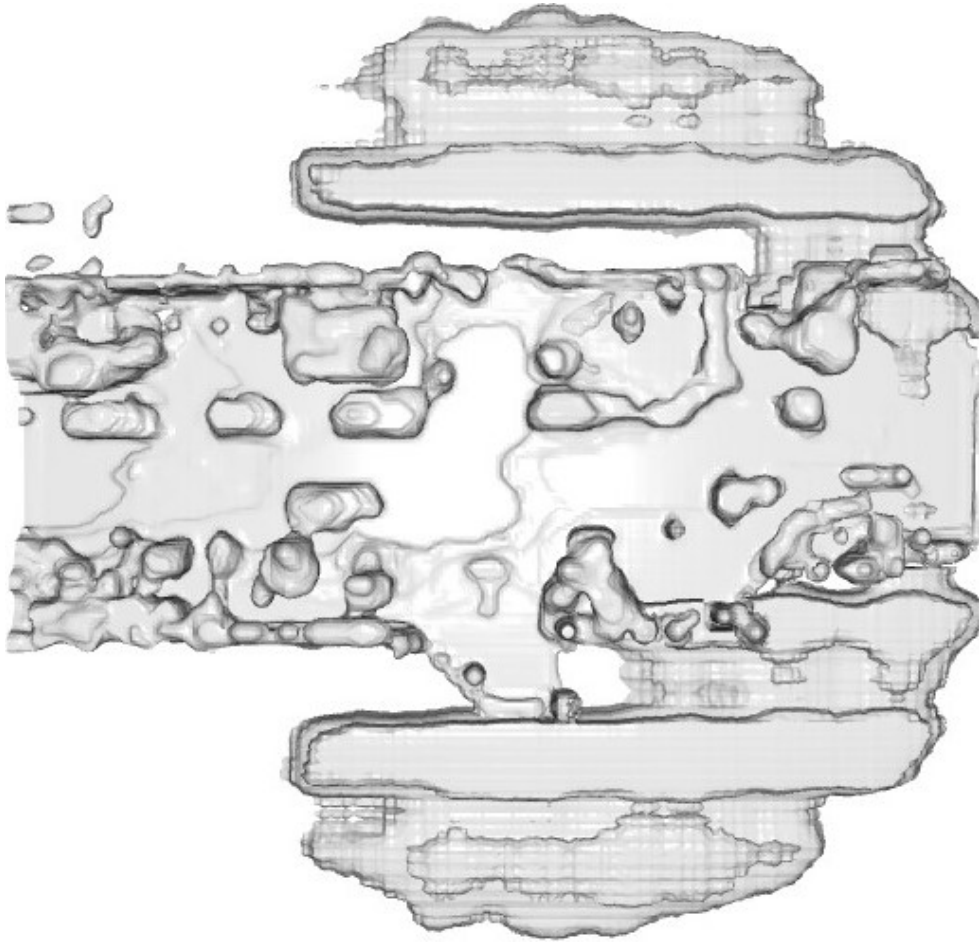


Figure 5.6: Top-down view of the model output for frame 545 from sequence 8, trained until epoch 50 with RGB input. Overfitting artifacts are visible on both sides of the road: the top and bottom of the image.

5.2.4 Data Efficiency

Regarding the data efficiency of the sensor fusion experiments, new training sessions were conducted with only a subset of the original dataset in order to infer whether the new LiDAR and RGB fusion approach can outperform the baseline while training with less data.

The validation loss graph is shown as Figure 5.14, and both approaches are very similar, with an almost complete overlap of the two losses. The graph for the experiment with RGB output is not present since it would not be comparable as it has one extra component in its loss function.

The results are present in Table 5.9, showing that there is not a significant difference between the approaches without RGB output using only part of the dataset, with an average IoU difference of 0.55%. Even though by a very small margin, the LiDAR-only approach had a better evaluation with every threshold. When also outputting color information, however, the difference increases, with this approach having the worst results, showing that it needs more data to be trained effectively.

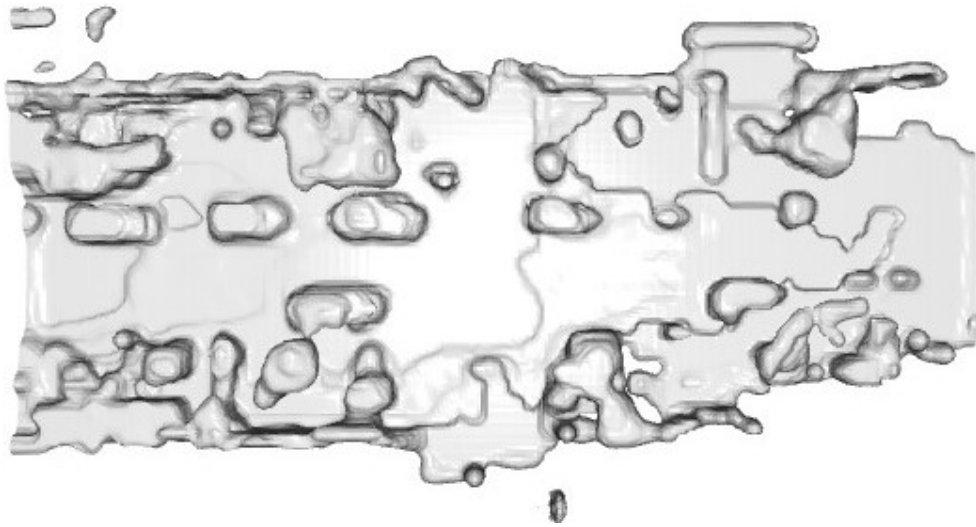


Figure 5.7: Top-down view of the model output for frame 545 from sequence 8, trained until epoch 50 with RGB input and dropout.

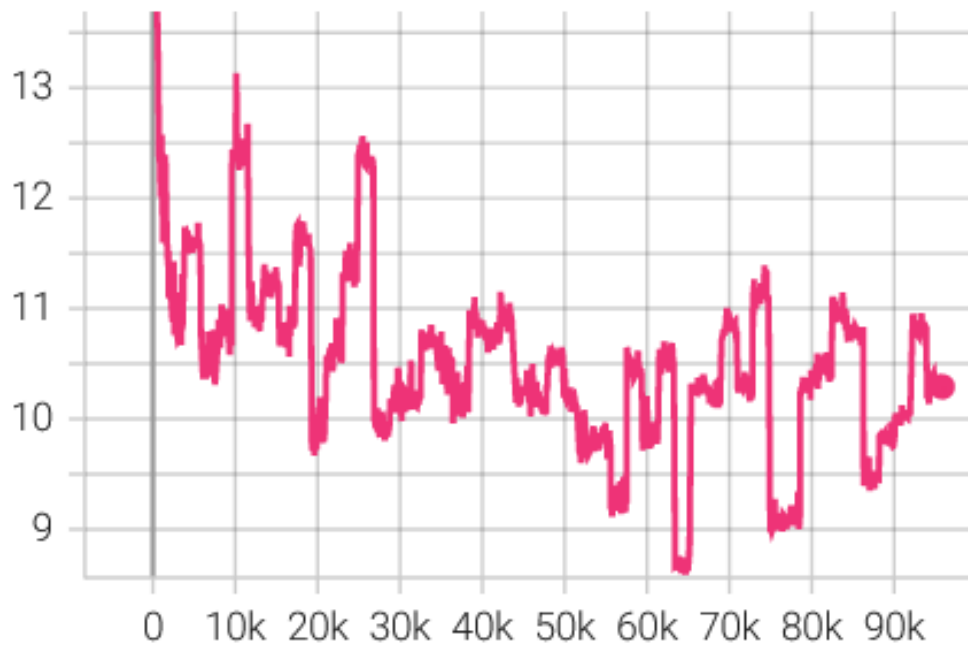


Figure 5.8: Validation loss for RGB component, trained until epoch 50.

Table 5.8: IoU score for each threshold with sensor fusion (see section 4.2) in the semantic scene completion task.

Input Data	Epoch	0.015	0.012	0.011	0.01	0.008	0.006	0.004	0.002
LiDAR	50	47.4	47.6	47.5	47.4	47.2	46.9	46.5	46.0
LiDAR + RGB	50	45.1	47.4	47.3	47.3	47.2	47.0	46.7	46.3



Figure 5.9: Top-down view of the model output for frame 545 from sequence 8, trained until epoch 50 with RGB input and output. Overfitting artifacts are visible on both sides of the road: the top and bottom of the image.

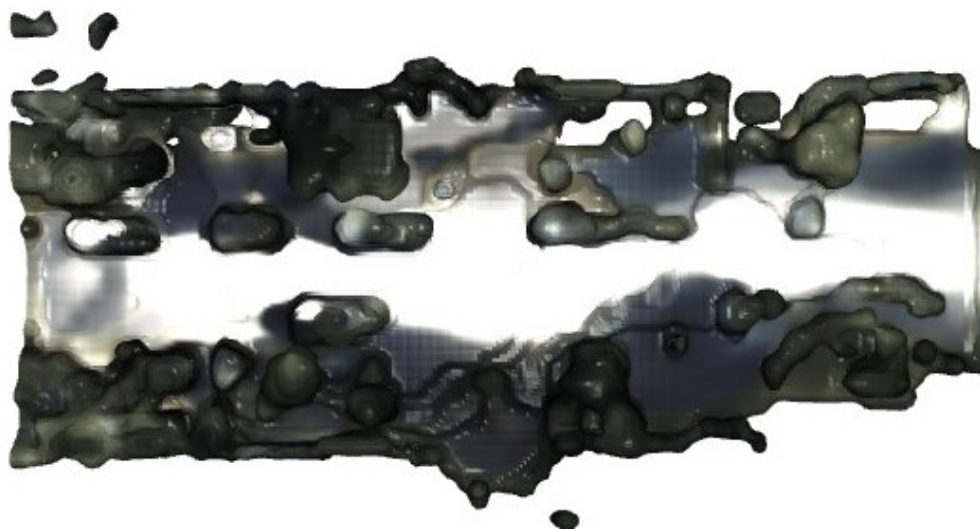


Figure 5.10: Top-down view of the model output for frame 545 from sequence 8, trained until epoch 50 with RGB input and output, and dropout.



Figure 5.11: Camera 2's image of the scene for frame 545 in sequence 8.

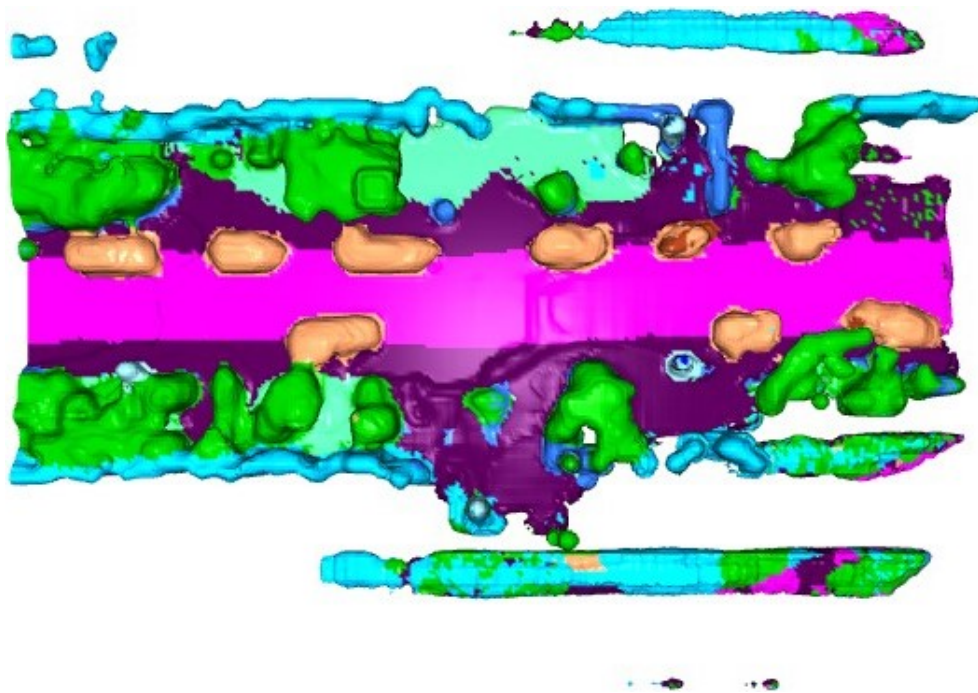


Figure 5.12: Top-down view of the model output for frame 545 from sequence 8 with semantic segmentation, trained until epoch 50. Overfitting artifacts are visible on both sides of the road: the top and bottom of the image.

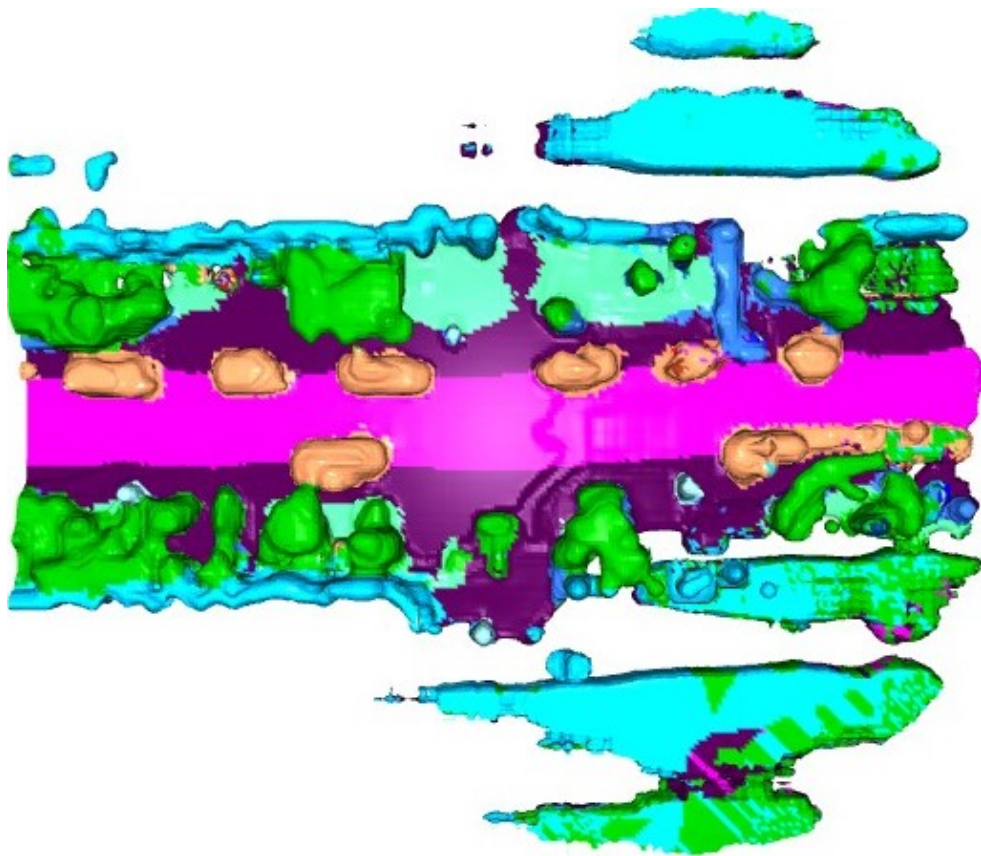


Figure 5.13: Top-down view of the model output for frame 545 from sequence 8 with semantic segmentation, trained until epoch 50 with RGB input. Overfitting artifacts are visible on both sides of the road: the top and bottom of the image.

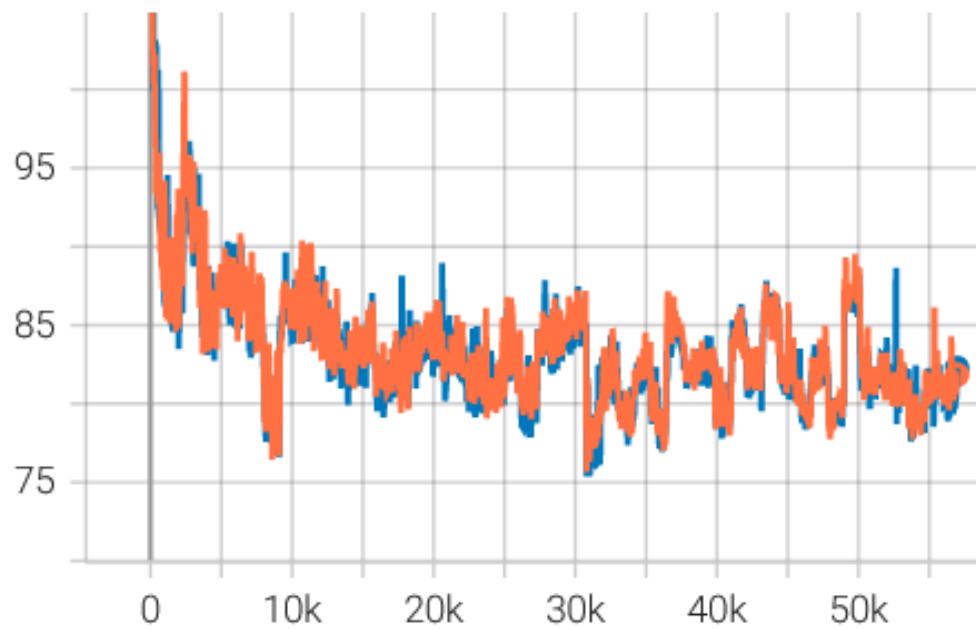


Figure 5.14: Validation loss for data efficiency experiment with sensor fusion (see subsection 4.2.4). Orange is LiDAR only and blue RGB and LiDAR fusion.

Table 5.9: IoU score for each threshold while training with half the dataset (see subsection 4.2.4) at epoch 50.

Data	Output	Epoch	0.015	0.012	0.011	0.01	0.008	0.006	0.004	0.002
LiDAR	Occupancy	50	47.8	47.8	47.7	47.6	47.4	47.2	46.8	46.2
LiDAR + RGB	Occupancy	50	47.2	47.1	47.0	46.9	46.8	46.6	46.4	46.1
LiDAR + RGB	Occupancy + RGB	50	46.6	46.6	46.5	46.4	46.2	46.0	45.8	45.4

Chapter 6

Conclusions and Future Work

The idea of utilizing deep implicit representations to reproduce three-dimensional objects and scenes from point clouds is an interesting and recent area of research with multiple publications in the last few years. However, the subset of publications applied to real-world contexts with a sparse LiDAR input is scarce. While most projects depend on a complete observation of the object, with points in all directions, any real-world application of this technique in the autonomous driving context must perform with only a single partial observation, adding a new layer of difficulty to this already challenging idea. These techniques either use signed distance functions to define surfaces and objects or the probability of occupancy at a certain point, with one taking this approach a step further by doing semantic scene completion through the addition of a free-space class to the three-dimensional segmentation problem.

This addition to the perception pipeline in autonomous vehicles could bring significant advantages for various perception problems. For example, regarding odometry estimation, the gradient of the signed distance functions or occupancy probability could hint at the direction of the movement between frames, and object detection could benefit from this intermediary step to reduce the need for labeled data since the detection challenge would be simplified by this intermediary step that is learned without supervision. This premise was, however, not confirmed nor denied by this dissertation.

Instead, this work focused on improving an existing representation while extending its representation capabilities to output color information of the scene and also improving its semantic segmentation performance. Early work on odometry estimation using the idea described was also conducted but not in time to be presented in this dissertation. To picture the generated scene in video format, a visualizer script was created that combines the outputted three-dimensional models in PLY format to create a video with frames-per-second that match the capture rate of the sensor. This made apparent some artifacts in the model which motivated the extraction of more validation metrics, and led to the addition of dropout layers in the generative part of the model. This fixed the issue, eliminating the artifacts at even at the highest epoch tested, but also slightly decreased the

model's performance, losing 2% IoU at epoch 200 when comparing with the best thresholds for each model. Adding RGB information to the model's input proved effective with 0.3% IoU gain at epoch 50 with dropout probability of 30% which, even though is positive, was an underwhelming result. Regarding color output with the described data fusion, the model proved to be unable to represent color but managed to represent luminance differences present in the provided RGB information: bright areas of the image appeared bright in the representation while dark ones also looked dim. For the semantic scene completion task, sensor fusion proved effective by increasing the mIoU by 0.2%, even though occupancy results decreased by the same margin. Increasing the model's capacity and experimenting with different types of fusion is an interesting route for future work. When comparing the data efficiency of the sensor fusion and LiDAR only approaches, results landed very close and considered within margin of error, with the original approach having an 0.6% IoU advantage using only half the training dataset.

Overall this work proved important to conclude that sensor fusion is a valid route to improve the performance and representation power of a deep implicit representation, specifically early fusion of LiDAR and RGB data was able to improve the model's geometric completion score, accurately represent luminance differences in the scene, and also improve the segmentation performance. It also showed that further work needs to be done to improve the representation quality, since the highest IoU scores were achieved using models with a tendency to overfit: before regularization was applied. However, many questions were still left unanswered. Is the initial premise true? Does this technique contribute to lessen the need for labeled data? Does odometry estimation benefit from a DIR? What other sensors' information could be fused to improve this representation?

This led to the creation of the following list with some ideas for further research:

1. Experiment with different types of fusion, for example feature and decision fusion (see section 2.4);
2. Further assess fusion with other types of sensors like thermal cameras, sonars, radars;
3. Increase the capacity of the model to infer if it increases the efficacy of the sensor fusion approaches;
4. Apply fusion to different architectures, namely the one seen in subsection 3.2.4 due to its superior occupancy results;
5. Employ the DIR to the different perception problems that were not tested in this work, for example in odometry estimation, object detection, weather estimation;
6. Experiment with using differentiable ray marching to supervise the representation result in the position of the original LiDAR points.

References

- [1] Waymo driver. <https://waymo.com/waymo-driver/>. Accessed: 2022-07-04.
- [2] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders. *arXiv preprint arXiv:2003.05991*, 2020.
- [3] 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.
- [4] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.
- [5] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11621–11631, 2020.
- [6] Rohan Chabra, Jan E Lenssen, Eddy Ilg, Tanner Schmidt, Julian Straub, Steven Lovegrove, and Richard Newcombe. Deep local shapes: Learning local SDF priors for detailed 3D reconstruction. In *European Conference on Computer Vision*, pages 608–625. Springer, 2020.
- [7] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.
- [8] Ming-Fang Chang, John Lambert, Patsorn Sangkloy, Jagjeet Singh, Slawomir Bak, Andrew Hartnett, De Wang, Peter Carr, Simon Lucey, Deva Ramanan, et al. Argoverse: 3d tracking and forecasting with rich maps. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8748–8757, 2019.
- [9] Christopher B Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In *European conference on computer vision*, pages 628–644. Springer, 2016.
- [10] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 303–312, 1996.
- [11] Angela Dai, Angel X Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5828–5839, 2017.

- [12] Haoqiang Fan, Hao Su, and Leonidas J Guibas. A point set generation network for 3d object reconstruction from a single image. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 605–613, 2017.
- [13] Jamil Fayyad, Mohammad A. Jaradat, Dominique Gruyer, and Homayoun Najjaran. Deep learning sensor fusion for autonomous vehicle perception and localization: A review. *Sensors*, 20(15), 2020.
- [14] M.W Gardner and S.R Dorling. Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric Environment*, 32(14):2627–2636, 1998.
- [15] Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. From <https://www.youtube.com/watch?v=w1Qo3bOiPaE>, May 2019.
- [16] Andreas Geiger. Convolutional occupancy networks. From <https://www.youtube.com/watch?v=EmauovgrDSM>, July 2020.
- [17] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3354–3361. IEEE, 2012.
- [18] Jakob Geyer, Yohannes Kassahun, Mentar Mahmudi, Xavier Ricou, Rupesh Durgesh, Andrew S Chung, Lorenz Hauswald, Viet Hoang Pham, Maximilian Mühlegg, Sebastian Dorn, et al. A2d2: Audi autonomous driving dataset. *arXiv preprint arXiv:2004.06320*, 2020.
- [19] Ankur Handa, Thomas Whelan, John McDonald, and Andrew J Davison. A benchmark for rgb-d visual odometry, 3d reconstruction and slam. In *2014 IEEE international conference on Robotics and automation (ICRA)*, pages 1524–1531. IEEE, 2014.
- [20] Honda. Honda launches next generation honda sensing elite safety system with level 3 automated driving features in japan. Available at <https://hondanews.com/en-US/honda-corporate/releases/release-e86048ba0d6e80b260e72d443f0e4d47-honda-launches-next-generation-honda-sensing-elite-safety-system-with-level-3-automated-driving-features-in-japan>, March 2021.
- [21] Xinyu Huang, Peng Wang, Xinjing Cheng, Dingfu Zhou, Qichuan Geng, and Ruigang Yang. The apolloscape open dataset for autonomous driving and its application. *arXiv preprint arXiv:1803.06184*, 2018.
- [22] Michael Kazhdan and Hugues Hoppe. Screened poisson surface reconstruction. *ACM Transactions on Graphics (ToG)*, 32(3):1–13, 2013.
- [23] John D Kelleher. *Deep learning*. MIT press, 2019.
- [24] R. Kesten, M. Usman, J. Houston, T. Pandya, K. Nadhamuni, A. Ferreira, M. Yuan, B. Low, A. Jain, P. Ondruska, S. Omari, S. Shah, A. Kulkarni, A. Kazakova, C. Tao, L. Platinsky, W. Jiang, and V. Shet. Level 5 perception dataset 2020. <https://level-5.global/level5/data/>, 2019.
- [25] Jelena Kocić, Nenad Jovičić, and Vujo Drndarević. Sensors and sensor fusion in autonomous vehicles. In *2018 26th Telecommunications Forum (TELFOR)*, pages 420–425, 2018.

- [26] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.
- [27] Pengfei Li, Yongliang Shi, Tianyu Liu, Hao Zhao, Guyue Zhou, and Ya-Qin Zhang. Semi-supervised implicit scene completion from sparse lidar. *arXiv preprint arXiv:2111.14798*, 2021.
- [28] Yiyi Liao, Simon Donne, and Andreas Geiger. Deep marching cubes: Learning explicit surface representations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2916–2925, 2018.
- [29] Yiyi Liao, Jun Xie, and Andreas Geiger. Kitti-360: A novel dataset and benchmarks for urban scene understanding in 2d and 3d. *arXiv preprint arXiv:2109.13410*, 2021.
- [30] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.
- [31] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. *ACM siggraph computer graphics*, 21(4):163–169, 1987.
- [32] Mercedes-Benz. S-class | conditionally automated driving with the drive pilot. From <https://www.youtube.com/watch?v=1gjweWq8qAc>, August 2021.
- [33] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4460–4470, 2019.
- [34] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision*, pages 405–421. Springer, 2020.
- [35] Stanley Osher and Ronald Fedkiw. Signed distance functions. In *Level set methods and dynamic implicit surfaces*, pages 17–22. Springer, 2003.
- [36] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. DeepSDF: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 165–174, 2019.
- [37] Scott Drew Pendleton, Hans Andersen, Xinxin Du, Xiaotong Shen, Malika Meghjani, You Hong Eng, Daniela Rus, and Marcelo H Ang. Perception, planning, control, and co-ordination for autonomous vehicles. *Machines*, 5(1):6, 2017.
- [38] Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. Convolutional occupancy networks. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16*, pages 523–540. Springer, 2020.
- [39] Quang-Hieu Pham, Pierre Sevestre, Ramanpreet Singh Pahwa, Huijing Zhan, Chun Ho Pang, Yuda Chen, Armin Mustafa, Vijay Chandrasekhar, and Jie Lin. A* 3d dataset: Towards autonomous driving in challenging environments. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2267–2273. IEEE, 2020.

- [40] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.
- [41] Christoph Rist, David Emmerichs, Markus Enzweiler, and Darius Gavrila. Semantic scene completion using local deep implicit functions on lidar data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.
- [42] Luis Roldao, Raoul De Charette, and Anne Verroust-Blondet. 3d semantic scene completion: a survey. *arXiv preprint arXiv:2103.07466*, 2021.
- [43] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.
- [44] SAE. Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles. Available at https://www.sae.org/standards/content/j3016_202104, April 2021.
- [45] Vincent Sitzmann, Eric R. Chan, Richard Tucker, Noah Snaveley, and Gordon Wetzstein. Metasdf: Meta-learning signed distance functions. In *arXiv*, 2020.
- [46] Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33, 2020.
- [47] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2446–2454, 2020.
- [48] Chiyu Max Jiang Vincent Sitzmann. Awesome implicit neural representations. <https://github.com/vsitzmann/awesome-implicit-representation>, 2021.
- [49] Favio Vázquez. Deep learning made easy with deep cognition. Available at <https://becominghuman.ai/deep-learning-made-easy-with-deep-cognition-403fbe445351>, December 2017.
- [50] Rikiya Yamashita, Mizuho Nishio, Richard Kinh Gian Do, and Kaori Togashi. Convolutional neural networks: an overview and application in radiology. *Insights into imaging*, 9(4):611–629, 2018.
- [51] Xue Ying. An overview of overfitting and its solutions. In *Journal of physics: Conference series*, volume 1168, page 022022. IOP Publishing, 2019.
- [52] Qian-Yi Zhou and Vladlen Koltun. Dense scene reconstruction with points of interest. *ACM Transactions on Graphics (ToG)*, 32(4):1–8, 2013.
- [53] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018.