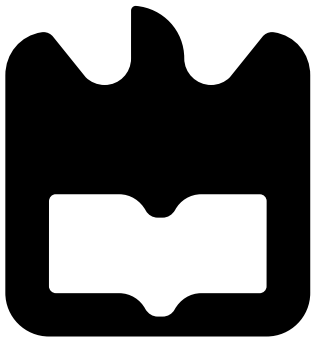
Diogo Alexandre
Amaral Conde Guedes

Identificação de objetos para veículos autónomos com base em aprendizagem automática

Object identification for autonomous vehicles based on Machine Learning



Diogo Alexandre
Amaral Conde Guedes

Identificação de objetos para veículos autónomos com base em aprendizagem automática

Object identification for autonomous vehicles based on Machine Learning

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor Miguel Drummond, Investigador Auxiliar no Instituto de Telecomunicações e da Professora Doutora Pétia Georgieva do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri / the jury

presidente / president

Prof. Doutor Armando José Formoso de Pinho
Professor Catedrático da Universidade de Aveiro

vogais / examiners committee

Doutor Hélder Filipe Pinto de Oliveira
Professor Auxiliar da Faculdade de Ciências da Universidade do Porto

Doutor Miguel Vidal Drummond
Investigador Auxiliar do Instituto de Telecomunicações de Aveiro

agradecimentos /
acknowledgements

Gostaria de agradecer,

Aos meus pais, à minha irmã e aos meus familiares mais próximos pelo total apoio que me deram ao longo de todos estes anos, nos meus melhores e piores momentos.

Ao Armando Sousa, Carlos Ribeiro, Filipe Reis, Gisela Pinto e Tiago Almeida por me terem acompanhado a enfrentar os momentos mais árduos e desafiantes do curso.

Ao Fábio Cunha, Igor Bernardo, Pedro Silva, Rafael Silva, Tiago Machado e Tiago Miguel pela amizade, encorajamento e por proporcianerem momentos de descontração que tornaram tudo mais fácil.

Ao Tiago Madeira e Diogo Duarte pelos momentos de diversão, inspiração e descoberta de novos paladares.

Em especial ao orientador e co-orientadora, Miguel Drummond e Pétia Georgieva, pela confiança e total disponibilidade para partilha de conhecimento, experiências e ajuda durante todo o processo de desenvolvimento desta dissertação.

À Bosch Car Multimedia, por ter proporcionado a oportunidade de trabalhar neste tema e fornecido conselhos no decorrer da dissertação.

À Universidade de Aveiro e ao DETI por me terem proporciado um ambiente onde pude crescer, tanto a nível pessoal como profissional, e conhecer todos os colegas com quem vivenciei a passagem por este árduo, mas enriquecedor, curso.

palavras-chave

Condução autónoma, LiDAR, simulador, point cloud, Deep Learning, identificação de objetos

resumo

Condução autónoma é uma das áreas mais ativamente estudadas em inteligência artificial. É esperado que os veículos autónomos reduzam significativamente os acidentes rodoviários e vítimas mortais quando se tornarem suficientemente maturos como opção de transporte. Atualmente, muitos dos esforços estão focados na prova de conceito de veículos autónomos serem baseados num conjunto de sensores que observam o ambiente em redor. Em particular, a camara e o LiDAR são estudados como sendo uma combinação eficiente de sensores para realização de identificação de objectos on-line nas estradas.

Identificação de objetos 2D é uma área de estudo já estabelecida no campo de Computação Visual. O sucesso na aplicação de técnicas de Deep Learning levou a que a visão 2D atingisse uma precisão ao nível Humano. No entanto, de forma a melhorar a segurança, abordagens mais avançadas sugerem que o veículo não deve depender de uma única classe de sensores. O LiDAR foi proposto como sendo um sensor adicional, particularmente devido à sua capacidade de visão 3D. Visão 3D depende dos dados capturados pelo LiDAR para reconhecer objetos em 3D. No entanto, em contraste com a identificação de objetos 2D, a identificação de objetos 3D é um campo de estudos relativamente imaturo e ainda possui muitos desafios para ultrapassar. Adicionalmente, LiDARs são sensores dispendiosos, o que também torna a aquisição de dados necessários para o treino de técnicas de reconhecimento de objetos 3D mais cara.

Neste contexto, esta tese de Mestrado tem como objetivo principal facilitar a identificação de objetos 3D, baseada em Deep Learning (DL), para veículos autónomos. As contribuições específicas deste trabalho são as seguintes.

Primeiro, uma visão global compreensiva do estado de arte relativo às arquiteturas Deep Learning para identificação de objetos 3D baseadas em point clouds. O propósito desta visão global é para perceber como melhor abordar este tipo de problema no contexto de condução autónoma.

Segundo, foi gerado um dataset sintético, mas realista, com dados capturados por um LiDAR no ambiente virtual do GTA V. Foram desenvolvidas ferramentas para converter os dados gerados no formato do dataset do KITTI, que se tornou num standard para avaliação de técnicas de deteção de objetos 3D para condução autónoma.

Terceiro, algumas das arquiteturas DL de identificação de objetos 3D revistas foram avaliadas com o dataset gerado. Apesar da sua performance com o dataset gerado ter sido pior que os resultados no dataset original do KITTI, os models chegaram a conseguir processar corretamente os dados sintéticos sem serem retreinados. O benefício futuro deste trabalho consiste nos modelos poderem ser adicionalmente treinados com dados produzidos localmente e testados em cenários variados.

O mod do GTA V implementado provou ser capaz de fornecer datasets ricos, bem estruturados e compatíveis com o estado de arte em arquiteturas de identificação de objetos 3D.

A ferramenta desenvolvida está disponível publicamente e esperamos que seja útil para o avanço da identificação de objetos 3D para condução autónoma, já que remove a dependência de datasets fornecidos por terceiros.

keywords Autonomous driving, LiDAR, simulator, point cloud, Deep Learning, object identification

abstract

Autonomous driving is one of the most actively researched fields in artificial intelligence. The autonomous vehicles are expected to significantly reduce the road accidents and casualties one day when they become sufficiently mature transport option. Currently much effort is focused to prove the concept of autonomous vehicles that is based on a suit of sensors to observe their surroundings. In particular, camera and LiDAR are researched as an efficient combination of sensors for on-line object identification on the road.

2D object identification is an already established field in Computer Vision. The successful application of Deep Learning techniques has led to 2D vision with Human-level accuracy. However, for a matter of improved safety more advanced approaches suggest that the vehicle should not rely on a single class of sensors. LiDAR has been proposed as an additional sensor, particularly due to its 3D vision capability. 3D vision relies on LiDAR captured data to recognize objects in 3D. However, in contrast to the 2D object identification, 3D object detection is a relatively immature field and still has many challenges to overcome. In addition, LiDARs are expensive sensors, which makes the acquisition of data required for training 3D object recognition techniques expensive tasks as well.

In this context, this Master's thesis has the major goal to further facilitate the 3D object identification for autonomous vehicles based on Deep Learning (DL). The specific contributions of the present work are the following.

First, a comprehensive overview of the state of the art Deep Learning architectures for 3D object identification based on Point Clouds. The purpose of this overview is to understand how to better approach such a problem in the context of autonomous driving.

Second, synthetic but realistic Lidar captured data was generated in the GTA V virtual environment. Tools were developed to convert the generated data into the KITTI dataset format, which has become standard in 3D object detection techniques for autonomous driving.

Third, some of the overviewed 3D object identification DL architectures were evaluated with the generated data. Though their performance with the generated data was worse than with the original KITTI data, the models were still able to correctly process the synthetic data without being retrained. The future benefit of this work is that the models can be further trained with home-made data and varying testing scenarios.

The implemented GTA V mod has proved to be capable of providing rich, well-structured and compatible datasets with the state of the art 3D object identification architectures.

The developed tool is publicly available and we hope it will be useful in advancing 3D object identification for autonomous driving, as it removes the dependency from datasets provided by a third party.

Contents

List of Figures

List of Tables

Acronyms

ADAS Advanced Driver-Assistance Systems
AOS Average Orientation Similarity
AP Average Precision
AVOD Aggregate View Object Detection
CCS Canonical Coordinate System
CNN Convolutional Neural Network
DL Deep Learning
E-RPN Euler Region Proposal Network
ECU Electronic Control Unit
FPN Feature Pyramid Networks
FPS Farthest Point Sampling
FTN Feature Transformation Network
GDC Graph-based Depth Correction
ILSVRC ImageNet Large Scale Visual Recognition Challenge
IoU Intersection over Union
ITN Input Transformation Network
KNN K-Nearest Neighbors
LiDAR Light Detection and Ranging
mAP Mean Average Precision
MLP Multi-Layered Perceptron
MRG Multi-Resolution Grouping
MSG Multi-Scale Grouping
MV3D Multi-View 3D Object Detection
MVCNN Multi-View Convolutional Neural Network
NHTSA National Highway Traffic Safety Administration
NMS Non-Maximum Suppression
PSMNet Pyramid Stereo Matching Network
PV-RCNN Point Voxel-RCNN
RaDAR Radio Detection and Ranging
RoI Region of Interest
RPN Region Proposal Network
SDN Stereo Depth Network
SIFT Scale-Invariant Feature Transform
SoNAR Sound Navigation and Ranging
SSD Single Shot Detection
SSG Single-Scale Grouping
STN Spatial Transformation Network
SVM Support Vector Machine
VFE Voxel Feature Encoding

Chapter 1

Introduction

1.1 Scope and motivation

Driving is an essential part of our everyday life, as vehicles constitute an accessible means of transportation for work, travel or leisure related activities. However, with it comes various dangers for both the driver and public road users alike. Dangers stem not only from fortuitous errors, but practically due to speeding, impaired driving and violation of traffic rules and regulations.

To prevent or diminish the repercussions derived from the widespread use of personal transportation vehicles, the automotive industry progressively developed a breadth of mechanisms and technologies, that would later be collectively known as Advanced Driver-Assistance Systems (ADAS). These include the standard seat belt, Traction Control System, Anti-lock Braking System and newer additions such as Adaptive Cruise Control, Lane Keeping Assist and Automatic Emergency Braking.

Statistical records published by National Highway Traffic Safety Administration (NHTSA) and other agencies support the effectiveness of these safety systems. For instance, a recent study by NHTSA [46], presents the overall tendency of the fatality rate decreasing with technology advances centered around driver and road safety, illustrated in Figure 1.1.

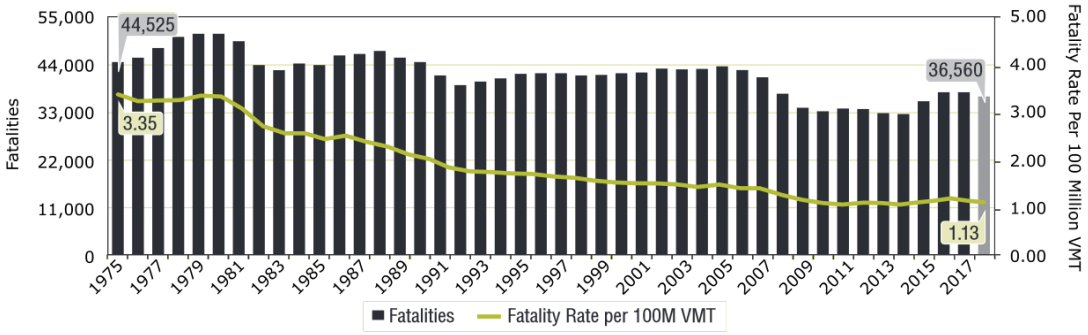


Figure 1.1: Fatalities and fatality rate per 100 Million VMT (vehicle miles traveled) in the US by year [46].

As the human factor significantly contributes to driving related injuries and casualties, the next step towards a more secure transportation environment may rely on autonomous vehicles. Recent advances in Computer Science in general, and in pattern recognition in particular, constitute a major stepping stone towards this goal, enabling accurate object identification. This means that it is now possible for a vehicle to recognize other vehicles, pedestrians, lanes and other objects of interest, with substantial accuracy.

In practice, the pattern recognition algorithms used in self-driving vehicles are paired with a multi-modal sensor suite that allows for the perception of the surrounding 3D environment. The comprised sensors, that may include cameras, SoNAR, RaDAR and LiDAR, acquire information using different techniques, that take advantage of their penetration capabilities and range, to capture information only available to them. Figure 1.2 briefly highlights their strengths and weaknesses.

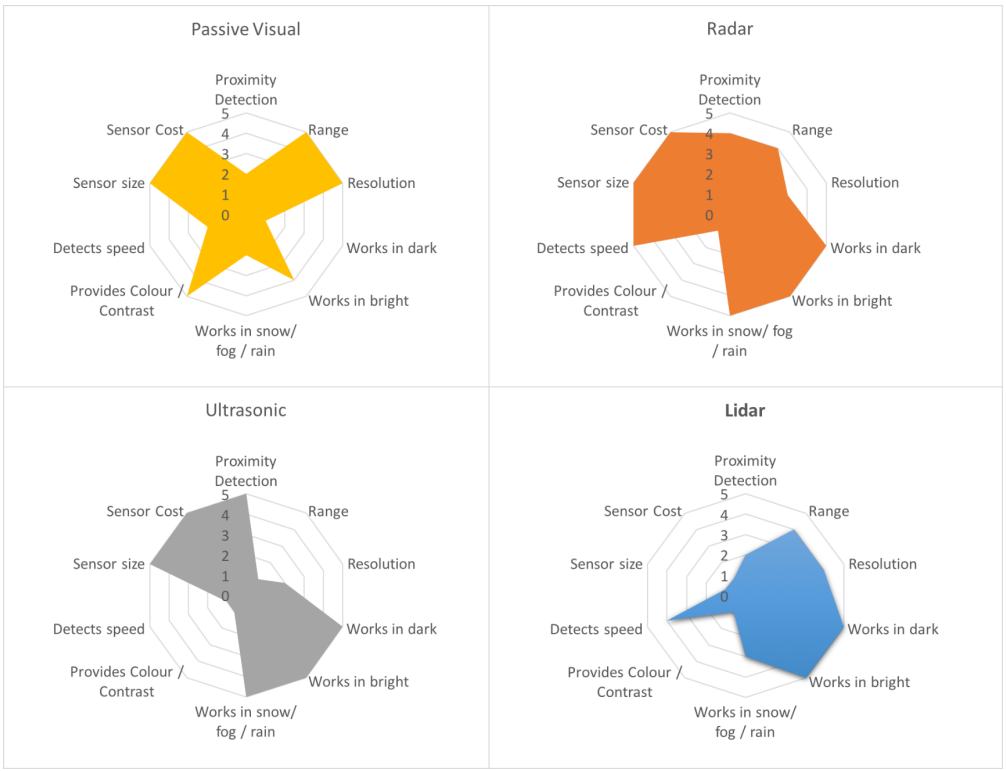


Figure 1.2: Performance comparison between camera (passive visual), RaDAR, Ultrasonic and LiDAR sensors [3].

Camera / Passive Visual. Digital cameras are the sensors responsible for capturing visual information of the environment. These are very sensitive to light, small and low cost due to their mass production, and are considered to be the main sensor of a self-driving vehicle. However, due to their inability of capturing environment information in poor lighting and weather conditions, and only providing indirect depth estimation for 3D mapping, RaDAR and LiDAR are used as additional sensors.

RaDAR. RaDAR systems are small, have a reasonable cost and are useful in situations where bad weather conditions such as fog, rain and snow, impair the measurements of the

camera and LiDAR sensors. RaDAR typically resorts to short pulses or frequency-modulated continuous wave (FMCW) modulation to enable time-of-flight (ToF) depth estimation for 3D mapping. Because the cost significantly increases with the angular resolution of the RaDAR, the phased-array antenna used in vehicles tends to have a limited number of elements, thus resulting in poor angular resolution and 3D mapping capabilities. At very short distances, they are also less effective than ultrasonic sensors.

Ultrasonic. Ultrasonic sensors are small and inexpensive sensors that emit high-frequency sound for low-range 3D mapping, being restricted to very short distances ($< 2\,\mathrm{m}$). These sensors are capable of detecting small differences of a centimeter or less.

LiDAR. LiDAR measurements are based on visible or infrared light, which enables very high angular resolution due to the simplicity of generating collimated laser beams. Laser modulation also enables ToF depth estimation. Both features make it possible to obtain a very detailed point cloud. However, the use of the visible and infrared light spectrums comes at a disadvantage. The rays are not capable of penetrating fog, for example. The high cost and size are also aspects that are hindering their deployment in current self-driving vehicles.

The synchronization and analysis of the information flows produced by the vehicle sensors is performed by a central Electronic Control Unit (ECU), illustrated in Figure 1.3. It is an embedded system tasked with the steering, acceleration and braking of the vehicle, based on an interpretation of the scene. The data frames captured by the vehicle sensors are supplied to Machine Learning algorithms in order to perform object identification, thus allowing the ECU to act according to the current situation.

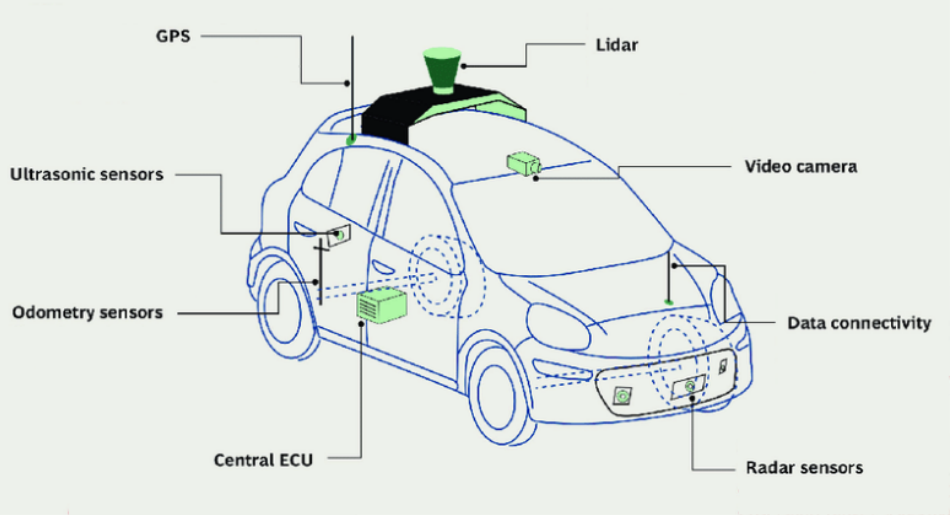


Figure 1.3: A simplified drawing of the various sensors present in Google's self-driving cars [2].

The Deep Learning algorithms for object identification require the development of models trained on labeled datasets. Real-life datasets are often used for training, as they mirror real-world scenarios and environments, and the complexity associated with them. Also, Deep Learning models for complex problems require massive amounts of data as these are comprised of a huge number of parameters. Otherwise, the models would be susceptible to overfitting.

In the context of autonomous vehicles, gathering such a sized dataset is not a particular limitation, as it is just a matter of collecting data from many prototype cars deployed in various locations. However, for such data to be valuable it must be accurately labeled. At best, this requires Human-supervised labeling, and, at worst manual labeling. Both are painstaking and expensive tasks that hamper the generation of and access to high-quality datasets.

Such downsides of real-life datasets have motivated many R&D groups to work on synthetic datasets. Synthetic datasets' most important advantages are:

- the ability to leverage the maturity in generating realistic scenarios brought by game engines.
- the ability to produce automatic and yet highly precise labeling.
- the simplicity involved in generating scenarios that could hardly be captured in real-life situations, such as life-threatening accidents.
- the ability to easily replicate scenarios where the model performs worst, with the objective to further improve its performance on less favorable settings.

The main motivation behind this thesis is to contribute to the generation of high-quality labeled datasets for autonomous driving.

1.2 Objectives

The main objective of this Master's thesis consists in the development of software to generate synthetic datasets for object identification, in the context of autonomous vehicles. An analysis of the performance of selected Deep Learning models (originally trained using real-life datasets) on a synthetically generated dataset is also performed, in order to evaluate the quality of the developed software's output and the adaptability of the models to fully synthetic datasets.

To this end, the following intermediate objectives were proposed:

1. Study of the state of the art Deep Learning (DL) architectures for object identification for autonomous vehicles.
2. Selection of a subset of the studied architectures to be used for the evaluation of a generated synthetic dataset.
3. Study of the publicly available solutions for the generation of synthetic datasets.
4. Development of software that leverages the environments provided by GTA V to generate labeled synthetic datasets.
5. Conversion of generated datasets to the desired KITTI dataset format.
6. Generation of a small synthetic dataset using the developed software.
7. Evaluation of the generated dataset using the selected state of the art DL architectures.

1.3 Contributions

The topic behind this thesis was originally discussed with an R&D team of Bosch Car Multimedia Portugal, in order to gain knowledge in synthetic datasets, as well as in Deep Learning architectures for object identification in the context of autonomous vehicles. The work done in this thesis contributes to such a knowledge gain.

The performed study of the state of the art is also a pioneering work in the University of Aveiro, involving Deep Learning approaches to 3D computer vision in the context of autonomous vehicles.

The developed software to meet the objective of this thesis is based of previous work developed in the 3rd year course of Informatics Engineering Project, available on `https://github.com/gta-project/mod`, and can be found on the GitHub page `https://github.com/Diogo525/dataset-generation-gtav`. This repository contains all the developed projects in the course of this thesis and the associated documentations on how to setup the various environments and use the projects. It includes:

- GTA V mod for the generation of synthetic datasets.
- Python project used to convert the GTA V mod output to the KITTI dataset format.
- Python scripts to generate ground truth 2D object detection results for Frustum PointNet evaluation and to visualize segmented objects directly on point clouds.
- Python project for the calculation of the IoU of inferred 2D and 3D bounding boxes, along with bird's eye view visualizations of the overlapping area.

The synthetic dataset generated from the developed software solution is publicly available in the project's repository and consists of 50 samples, in the KITTI dataset format, based on the virtual environments provided by GTA V.

The first-ever comparison between the performance of state of the art object identification architectures on a GTA V-based synthetic dataset and the original KITTI testing set.

1.4 Document structure

This document is divided into 5 chapters:

Chapter 1 - *Introduction*: definition of the theme and scope of this document, along with the motivation for the developed work and the provided functionalities. It describes the contributions associated with the study of the state of the art in 3D object identification and the developed software solution. A summary of the document's structure is also presented.

Chapter 2 - *State of the Art*: a brief overview of relevant 2D object identification architectures, the datasets and metrics commonly used to train and evaluate the 3D object identification models, a review of the state of the art for 3D object identification in the context of autonomous vehicles, a comparison between the reviewed architectures and the selection of a subset for experimenting with the generated synthetic dataset.

Chapter 3 - *Dataset Generation for 3D Object Identification*: description of the process performed to generate synthetic datasets from GTA V and correspondent conversion to the KITTI dataset format.

Chapter 4 - *Evaluation of Generated Dataset*: the generated dataset is evaluated with two different approaches to 3D object identification: image-based and LiDAR-based.

Chapter 5 - *Conclusion and Future Work*: an overview of the results and outcomes presented throughout the document, the limitations of the provided solution and a discussion towards future work.

Chapter 2

State of the art

In this chapter we present the foundation of this work, consisting in a thorough review of the current state of the art Deep Learning architectures for object identification, in the context of autonomous vehicles, and the correspondent analysis towards an understanding of relevant approaches to the unsolved 3D object identification problem. It is also discussed the reasons behind the selected architectures for the evaluation of the generated synthetic dataset and the engine used for its generation.

This chapter is divided into the following sections. Section 2.1 contains a brief overview of current Deep Learning architectures that accurately perform 2D object identification, as these are frequently used and adapted by networks aiming to solve the same problem but in a higher dimension. Section 2.2 briefly introduces the datasets used to train and evaluate the 3D object detection models analysed in Section 2.4. Section 2.3 describes the commonly used metrics to evaluate 3D object identification models. Finally, Section 2.4 explores the state of the art approaches to 3D object identification and discusses their advantages and downsides in the context of autonomous vehicles.

2.1 Deep Learning for 2D Object Identification

Machine Learning algorithms have completely outperformed handcrafted techniques in object identification tasks. The Machine Learning models can be easily adapted to more specific scenarios simply by feeding the models with suitable training datasets; in addition, the lack of hard-wired handcrafted features enables a greater generalization to the input data.

More precisely, Deep Learning architectures streamlined the approaches for dealing with several challenging self-driving tasks that required a processing pipeline to break these down into explicit stages, such as lane marking detection, path planning and control for vehicle steering [5].

From predicting steering decisions, to object identification for traffic sign classification and pedestrian detection, many of the proposed DL models were initially developed to process 2D image datasets.

In general, Deep Learning algorithms for 2D image datasets can be classified according to their inference granularity:

- **DL for 2D Image Classification** aims to classify the entire input image with a single output label.
- **DL for 2D Object Detection** aims to infer the location of multiple objects present in the input image by classifying and generating bounding boxes around possible targets.
- **DL for Image Segmentation** aims to classify every pixel of the input image to output a segmentation map that distinguishes regions belonging to different object classes.

2.1.1 Deep Learning for Image Classification

Deep Learning for 2D image classification became extremely popular since the introduction of the AlexNet architecture in 2012 [33], which significantly outperformed previous record holders of the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [58], with a classification error of 15.3% [29], through the use of a deep convolutional neural network.

AlexNet is the successor to the first practical implementation of convolutional neural networks, published in 1998 by Yann Lecun et al [37], and after its remarkable success, the following years were met with several publications of CNN architectures that continued to reduce the classification error. In the ILSVRC 2017, the SENet [28] was able to reach an error of 2.3% [29], greatly surpassing the human top-5 classification error rate of 5.1%, reported in [58], on the same dataset. Figure 2.1 presents the evolution of the classification error results of the ImageNet Challenge competitors over the years.

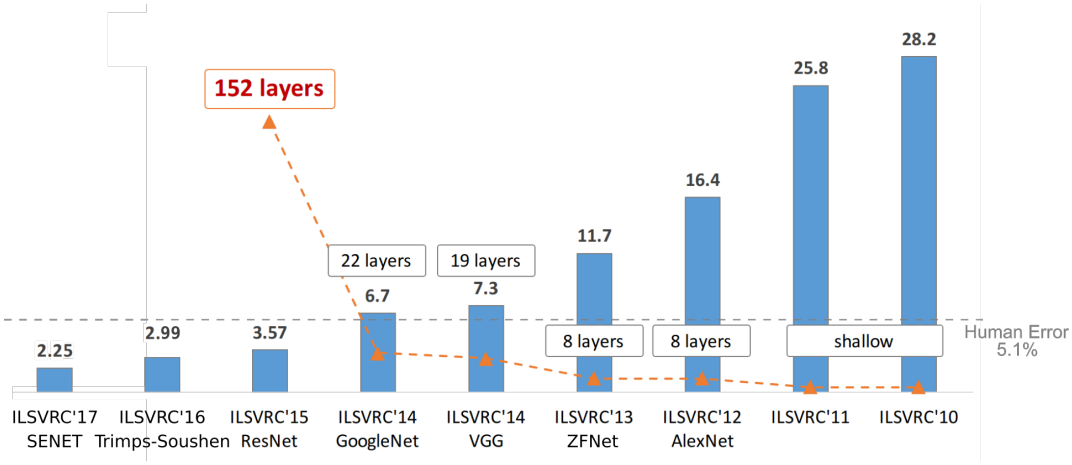


Figure 2.1: ImageNet Classification top-5 error (%), adapted from [25].

This rapid evolution of the Deep Learning approaches can be attributed to the increased availability of large scale annotated datasets, high performance GPU clusters and significant design improvements of network structures and training strategies [77].

Compared to CNNs, fully-connected neural networks have several disadvantages when operating with higher dimensional datasets, such as images. The interaction between all the previous and next layer neurons of the network in combination with this type of input data, originates a network with a significant number of parameters and difficult to train, due to its increased capacity and larger dataset requirements [38].

The usage of CNNs over fully-connected neural networks for image-based Deep Learning tasks

is encouraged by the following simplifications:

Sparse interactions. In a fully connected Neural Network, the output of each unit is dependent on the interactions between all the units of the previous layer, so the number of required parameters grows with the input image resolution. Because convolution layers use filters, which are generally several orders of magnitude smaller than the size of the input feature maps, to extract meaningful local features, the input image resolution does not have a significant impact on the number of parameters of these layers, leading to smaller memory requirements and less computational costs [24].

Parameter sharing. In a fully connected Neural Network, each weight is only used once to calculate the result of the connected output unit, whereas in a Convolution layer the same kernel parameters are used throughout the entire input feature map, according to the kernel size, padding and stride configurations [24].

Translation equivariance. It is a property that directly relates input transformations to feature vector representations. A convolution layer is translation equivariant because the application of the same translation before or after the convolution results in the same output. This property is achieved by the reuse of the kernel throughout the entire input feature map and it is useful to generate predictable transformations in the output feature vectors through corresponding input image transformations, making the learning process easier [74].

Translation invariance. The translation invariance property of a CNN means that translation transformations applied to the input image do not significantly affect the output results of the network.

Hierarchical representation. Hierarchical feature extraction is achieved by stacking multiple convolution layers, which enables the formation of intermediate representations of the input image from low to high levels of abstraction [32, 33]. In an effort to understand the representations learned by these networks, several methods were devised to visualize the receptive field of the units present in the hidden layers [43].

Current state of the art approaches to image classification are the Res2Net [17], which introduces a new CNN building block for increasing the multi-scale representation ability of the networks, and the EfficientNet [68], that studies model scaling and proposes a method to uniformly scale the depth, width and resolution of a network. The top-5 classification accuracy on ILSVRC of both publications are 94.43% and 97.1%, respectively, and the latter was capable of achieving a score of 84.4%, the highest score in top-1 classification accuracy.

2.1.2 Deep Learning for 2D Object Detection

The objective of 2D object detection is to locate and classify multiple objects in a given input image, and can be generally divided into the following categories [77]:

- **Region proposal based methods**: they are composed of a sequence of correlated stages comprising region proposal generation, feature extraction using a CNN, classification and bounding box regression, usually trained separately. One precursor to these

methods is the Overfeat architecture [59], that combines a CNN with the sliding window method to predict bounding boxes according to the classification score for each object class. Relevant architectures in this category are the R-CNN network variants [23, 22, 56, 26] and the FPN network [39].

- **Regression / Classification based methods**: the YOLO [54], YOLOv2 [55] and SSD [41] are amongst the architectures encapsulated by this category. They consist in one-step frameworks that directly map the input images to the predicted bounding box coordinates and classification probabilities, and thus have faster inference times and are trainable end-to-end.

The progression of the 2D object detection architectures in the past few years is illustrated in Figure 2.2.

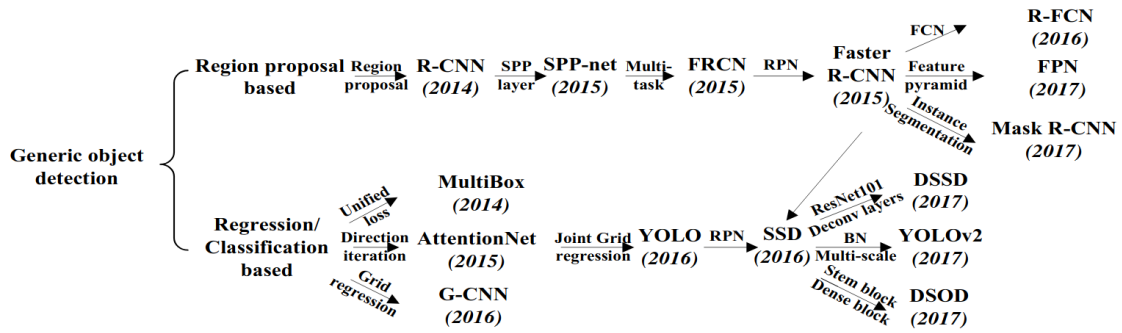


Figure 2.2: Evolution of the two 2D object detection frameworks. SPP: Spatial Pyramid Pooling [39], FRCN: Faster R-CNN [26], RPN: Region Proposal Network [56], FCN: Fully Convolutional Network [39], BN: Batch Normalization [30], Deconvolution layers [76], sourced from [77].

2.1.3 Deep Learning for Image Segmentation

Image Segmentation studies also benefited from the popularization of convolutional neural networks, and can divided into three different categories [18]:

- **Semantic Segmentation**: classification of every pixel in an image without differentiating between objects within the same class.
- **Instance Segmentation**: each segmented object of the same class has a unique label. In images with multiple objects, this task is performed by first masking the regions where objects have been detected, followed by the segmentation step.
- **Part-based Segmentation**: consists in a low-level decomposition into parts of already segmented entities.

The paper [42] published by Jonathan Long et al, was the first to adapt CNNs for image segmentation. From the GoogLeNet and VGG16 classification networks, the authors replaced the fully connected layers with convolution layers in order to produce an output heatmap, With this transformation, the authors were able to apply transfer learning and fine-tuning for the segmentation task, and being fully convolutional networks it became possible to take input maps of variable sizes to produce correspondent output maps.

Other relevant approaches in the context of this thesis are Encoder-Decoder Based Models, Multi-Scale and Pyramid Network Based Models and R-CNN Based Models for Instance Segmentation, as described in [45].

2.1.4 Summary

This section presented a brief overview of the various 2D object identification approaches released in the last decade, and provides a guideline to the Deep Learning architectures that serve as the bases for the 3D object identification approaches described in the Section 2.4.

The strategies used to develop 2D object identification architectures have reached a point where the models are able to outperform Human accuracy in image classification and reliably detect and segment target images. The same cannot be said in relation to 3D object identification architectures, which are still in early stages of research and development.

2.2 3D Object Identification Datasets Overview

Before proceeding with the analyses of the Deep Learning architectures for 3D object identification, it is important to first get to know the datasets handled by such models.

2.2.1 Modelnet40 dataset

The ModelNet dataset is provided by Princeton University [70] and contains 127,915 CAD models of regular objects, with 662 object categories. The models are represented in a mesh format, as illustrated in Figure 2.3.

Figure 2.3: Sample object of the ModelNet dataset [70].

The ModelNet40, in particular, is a subset consisting of 12,311 shapes and defines 40 object categories. The reviewed state of the art that makes use of this subset, the MVCNN, the PointNet and the PointNet++, perform a dataset split of 80/20 for the training and testing sets, respectively.

2.2.2 KITTI Object detection dataset

The KITTI Object Detection dataset was published in 2012 by Andreas Geiger et al, [21] and consists of a dataset focused on autonomous vehicles. The samples provide information to perform the benchmarks for 2D object detection and orientation estimation, the 3D and bird's eye view object detection.

The authors also hold a competition on 3D object identification and, as such, divide the supplied dataset into two parts:

- train set: consists of 7,481 samples and is used by competitors for training and validating the Deep Learning models.
- test set: consists of 7,518 unlabeled samples. This dataset is used by the KITTI team to evaluate submitted models, referred to as online evaluation. The competitors do not have access to the labels and use this set to perform offline evaluation of their models.

The dataset is labeled with 8 different classes, but only the car, pedestrian and cyclist classes are used in the benchmarking process due to their higher appearance count. Figure 2.4 illustrates a KITTI object detection sample.

Figure 2.4: Front view and point cloud of a KITTI object detection sample.

The majority of the reviewed state of the art performs a dataset split of 50/50 for training and validation sets, respectively, as proposed by [11]. With the exception of Complex-YOLO that uses a 85/15 dataset split.

The authors also defined 3 difficulty levels (Easy, Moderate and Hard) for the samples, according to an object's minimum 2D bounding box height, occlusion and truncation levels. The associated field values for each difficulty level are shown in Table 2.1.

Difficulty	Min. 2D bounding box height	Occlusion level	Max. Truncation
Easy	40 px	Fully visible	15%
Moderate	25 px	Partly occluded	30%
Hard	25 px	Difficult to see	50%

Table 2.1: KITTI object detection difficulty levels. This partition into different sample categories is a stark contrast to the ModelNet dataset. It allows us to almost intuitively infer that training a model is highly dependent on its application and scenarios, as well as how the labeling is defined. A more thorough description of this dataset is given in Section 2.5.

2.3 3D Object Identification Evaluation Metrics

This section presents the various metrics used to evaluate the reviewed state of the art 3D object detection models. The Average Precision and Mean Average Precision are calculated from the Intersection-over-Union metric, that is commonly used to evaluate the performance of 3D bounding box, bird's eye view bounding box (on the top-to-bottom view) and 2D bounding box (on the front view image) detections.

2.3.1 Precision

The precision metric measures the accuracy of a model in classifying a sample as positive [16], and is calculated using Equation 2.1.

$$P = \frac{\text{TP}}{\text{TP} + \text{FP}}, \tag{2.1}$$

where TP is the number of True Positives and FP the number of False Positives.

2.3.2 Recall

The recall metric measures the ability of the model to detect positive samples, independently of how the negative samples are classified. It is calculated using Equation 2.2.

$$R = \frac{\text{TP}}{\text{TP} + \text{FN}}, \tag{2.2}$$

where TP is the number of True Positives and FN the number of False Negatives.

2.3.3 Precision-Recall Curve

The Precision-Recall curve shows the trade-off between the precision and recall values for different thresholds. It is used to determine which threshold value maximizes the precision and recall.

2.3.4 Intersection over Union

The Intersection over Union (IoU) metric, or Jaccard Index, is used to calculate the percentage of overlap between the ground truth and the predicted bounding boxes. It is calculated using the Equation 2.3.

$$\frac{\text{Area of Overlap}}{\text{Area of Union}} = \frac{\text{GT Area} \cap \text{Predicted Area}}{\text{GT Area} \cup \text{Predicted Area}} \tag{2.3}$$

By computing a normalized result focusing on the bounding boxes' areas, or volumes, the IoU metric is invariant to scale. Due to this crucial property, the IoU metric is the base for the majority of the performance measures for segmentation, object detection and tracking [57].

2.3.5 Average Precision

(Approximated) Average Precision

The Average Precision summarizes the Precision-Recall curve as the weighted mean of precision's achieved at each IoU threshold, where the weight is the increase in recall from the previous threshold [36]. It is equal to the area under the Precision-Recall curve, calculated using Equation 2.4.

$$AP = \int_0^1 p(r)dr \approx \sum_{k=1}^{N} P(k)\Delta r(k), \tag{2.4}$$

where $P(k)$ is the precision and $r(k)$ the recall at threshold k, and $\Delta r(k)$ is the difference between the current and the next recall.

(Interpolated) Average Precision

Interpolated Average Precision is another way to calculate the Average Precision. Instead of using the exact precision at threshold k in each iteration, it uses the maximum precision observed across all thresholds with higher recall [44]. It is calculated using Equation 2.5.

$$AP = \sum_{k=1}^{N} \max_{\tilde{k} \geq k} P(\tilde{k})\Delta r(k), \tag{2.5}$$

where $\tilde{k}$ represents the threshold, greater than or equal than the current iteration's threshold, at which the recall is the highest. So, the selected $\tilde{k}$ corresponds to when the recall is the highest, for $\tilde{k} \geq k$.

11-Point and 40-Point Interpolated Average Precision

Other variations, instead of using the exact precision value at $\tilde{k}$, define a range of sampled recall values on which to observe the $\tilde{k}^{th}$ value corresponding to the highest sampled recall value. The sampled recall values used in both variations are uniformly spaced, with AP_{11} using 11 recall values in total, $R_{11} = \{0, 0.1, ..., 0.9, 1\}$, and AP_{40} using 41 values in total, $R_{40} = \{\frac{1}{40}, \frac{2}{40}, ..., \frac{39}{40}, 1\}$.

KITTI originally used the AP_{11} metric to evaluate the submitted models, as proposed by the original Pascal VOC benchmark [14]. However, in 2019, in the paper [63], published by Andrea Simonelli et al, the proposition to use AP_{40} was made due to an unwanted consequence of the AP_{11}'s recall interval starting at 0. In this case, if a single correctly matched prediction occurs, in a given IoU level, the precision will be 100% at the recall value 0, resulting in a defective evaluation of the quality of the algorithm.

2.3.6 Mean Average Precision

Whereas the Average Precision is calculated for each class, the Mean Average Precision is the mean of the Average Precision's for all the classes, given by Equation 2.6.

$$mAP = \frac{\sum_{c=1}^{C} \mathrm{AP}(c)}{C}, \tag{2.6}$$

where C is the number of object classes.

2.3.7 Average Orientation Similarity

The AOS is the metric used by KITTI to evaluate the orientation of the 3D rotated bounding box detections, calculated using Equation 2.7.

$$AOS = \frac{1}{11} \sum_{r \in \{0, 0.1, \ldots, 1\}}^{N} \max_{\tilde{r}:\tilde{r} \geq r} s(\tilde{r}), \tag{2.7}$$

where r is the recall, taking into account the 0.5 IoU threshold used on the inferred 2D bounding boxes, and $s \in [0, 1]$ is the method to compute the orientation similarity, as described in the KITTI object detection paper [21].

2.4 Deep Learning for 3D Object Identification

The previous Sections 2.2 and 2.3 provided an introduction to commonly used datasets and evaluation metrics to perform training, validation and testing of state of the art 3D object identification models.

In contrast to 2D object identification, 3D object identification is a relatively new subject in the field of Computer Vision, that emerged with the affordability of 3D mapping sensors, such as LiDAR and RGB-D cameras.

As presented in Section 1.1, many systems rely on 3D mapping technologies to take advantage of their ability to build a more accurate 3D representation of the scene and to provide assistance in situations of poor visibility conditions. These sensors collect and store the information gathered from the 3D environment scans in a point cloud representation, which consists of a set of points in euclidean space, where each point not only contains its 3D coordinates, but can also be associated with additional information such as color, surface normals and reflectivity.

In spite of the success of Convolutional Neural Networks (CNNs) for Computer Vision tasks based on 2D images, these architectures are not suitable for processing data in the form of point clouds, mainly due to their unstructured nature. There are also other characteristics of the point cloud representation that makes the direct application of CNN architectures non-trivial, including variable number of points due to object occlusions, material reflectivity, atmospheric conditions, range and other equipment limitations, measurement errors, for example due to noise, the susceptibility to suffer rotations around the local axes of the sensor's coordinate system and point permutation.

To address the unconventional characteristics of the point cloud representation, many architectures rely on pre-processing the point cloud samples in order to convert these into more structured representations, such as voxels or image views, that allow for the application of

already proven architectures used in 2D object identification problems. Amongst the various representations used to store and process 3D data, Figure 2.5 illustrates the commonly used ones in 3D object identification architectures.

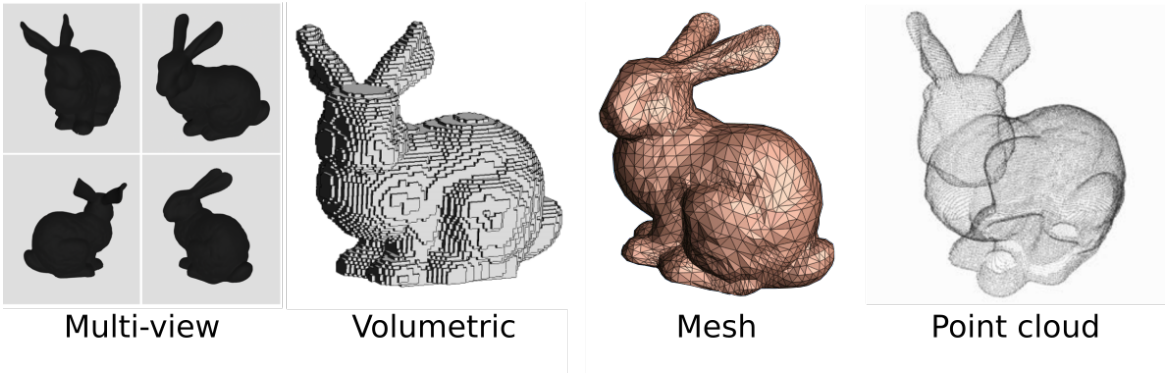


Figure 2.5: Representations of 3D data: multi-view [71], volumetric [49], mesh [50] and point cloud [52].

In order to understand the progress of 3D object identification algorithms and the motivations behind the approaches to this unsolved problem, the rest of this section involves the analysis, in chronological order, of relevant state of the art architectures that contributed to this field of study.

2.4.1 Multi-View Convolutional Neural Network (MVCNN)

The MVCNN architecture is one of the first to tackle 3D object classification. It was published by Hang Su et al [66], in 2015, and approached this challenge by creating multiple rendered views of the input 3D object, with mesh representation, and using these for the classification and retrieval tasks.

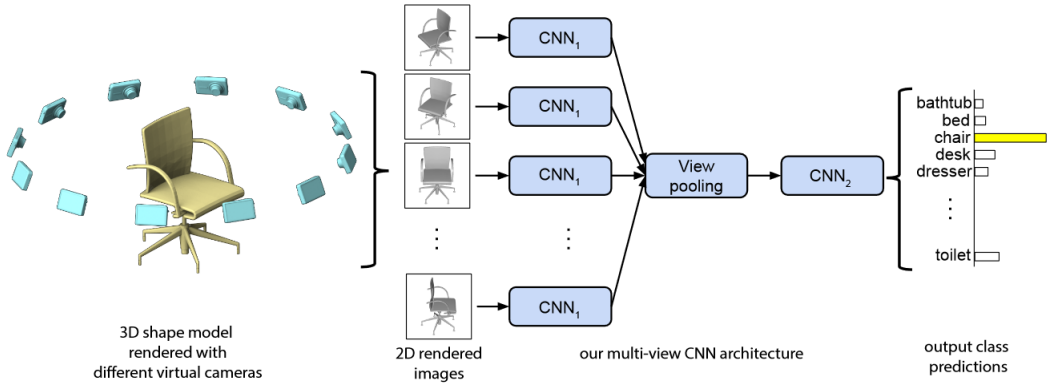


Figure 2.6: MVCNN architecture diagram [66].

The decision to use the multi-view representation relied on several factors:

- convolution operations over 2D representations are less computationally expensive than over voxel-based representations, where it requires a 3D kernel to operate over the voxel grid.

- the size of the volumetric representation grows cubically with respect to the input resolution, preventing the use of models with higher resolutions due to memory and computational constraints.
- 3D object datasets are scarce compared to the massive datasets of image samples.

The authors proposed a model for combining multiple view-based descriptors of an object into a single "compiled" object descriptor. A descriptor is a float or binary vector that represents the signature of an object, i.e. a feature vector, and can be classified into 2 general categories:

- **3D shape descriptors**: describe native representations of 3D objects, such as meshes, voxel grids and point clouds. They present some drawbacks in regard to their high-dimensionality and low availability of labeled 3D object datasets.
- **View-based descriptors**: describe 3D objects from image views. They were chosen for their low-dimensionality, relatively higher efficiency and robustness against artifacts present in 3D representations of shapes, such as holes, imperfect tessellations and surface noise.

In addition, the view-based descriptors are obtained through CNN activation features, instead of handcrafted features based on geometric properties of the object. This approach enables the network to be more generalized and to better learn which views are the most informative for more accurate object classification.

Two camera setups were experimented with in order to create the multi-view representations of the 3D objects:

- the first camera configuration used 12 virtual cameras pointed at a 30^{o} angle towards the centroid of the object and are positioned around the mesh every 30 degrees, as show in Figure 2.6. With this setup, every 3D object was positioned upright.
- the second camera configuration used a total of 80 views. This setup placed a camera at each of the 20 vertices of an icosahedron, also pointing at the centroid of the model, and applied rotations of 0, 90, 180 and 270 degrees, along the axis passing through each camera and the object centroid. For this camera configuration, due to the greater number of captured views, there is no assumption of consistent upright orientation of shapes.

To increase the transformation invariance of the network, data augmentation was performed by applying affine transformations to the multi-views of the input object.

Architecture Overview

The operations of the MVCNN architecture can be summarized in 3 main steps, illustrated in Figure 2.6, as follow:

1. The architecture starts by passing each individual object view through the CNN_1 to produce a set of view descriptors, one per view.
2. The resulting descriptors are then fed through the View-Pooling layer, which uses the element-wise maximum operation to aggregate these into a single object descriptor.

3. The object descriptor is finally given to the CNN_2 in order to produce the classification results.

Performance Analysis

The authors experimented with two approaches for the feature extraction step:

- **Fisher Vectors (FV)**: Fisher vectors with multi-scale SIFT to extract handcrafted features for object classification [67].
- **CNN**: a VGG-M network to extract "learnable" features taken from the last hidden layer nodes of the network [64].

All the developed approaches were compared between each other and with state of the art image 3D shape descriptors of the time. The results are shown in in Figure 2.7.

Method	Training Config.			Test Config.	Classification (Accuracy)	Retrieval (mAP)
	Pre-train	Fine-tune	#Views	#Views		
(1) SPH [16]	-	-	-	-	68.2%	33.3%
(2) LFD [5]	-	-	-	-	75.5%	40.9%
(3) 3D ShapeNets [37]	ModelNet40	ModelNet40	-	-	77.3%	49.2%
(4) FV	-	ModelNet40	12	1	78.8%	37.5%
(5) FV, 12×	-	ModelNet40	12	12	84.8%	43.9%
(6) CNN	ImageNet1K	-	-	1	83.0%	44.1%
(7) CNN, f.t.	ImageNet1K	ModelNet40	12	1	85.1%	61.7%
(8) CNN, 12×	ImageNet1K	-	-	12	87.5%	49.6%
(9) CNN, f.t.,12×	ImageNet1K	ModelNet40	12	12	88.6%	62.8%
(10) MVCNN, 12×	ImageNet1K	-	-	12	88.1%	49.4%
(11) MVCNN, f.t., 12×	ImageNet1K	ModelNet40	12	12	89.9%	70.1%
(12) MVCNN, f.t.+metric, 12×	ImageNet1K	ModelNet40	12	12	89.5%	**80.2**%
(13) MVCNN, 80×	ImageNet1K	-	80	80	84.3%	36.8%
(14) MVCNN, f.t., 80×	ImageNet1K	ModelNet40	80	80	**90.1**%	70.4%
(15) MVCNN, f.t.+metric, 80×	ImageNet1K	ModelNet40	80	80	**90.1**%	79.5%

* f.t.=fine-tuning, metric=low-rank Mahalanobis metric learning

Figure 2.7: Classification and retrieval results on the ModelNet40 dataset [66].

In regards to the classification task, there are three main comparisons between all the approaches:

- **3D shape vs view-based descriptors**: all the models using view-based descriptors outperform the 3D shape descriptor models, even when only one object view is supplied.
- **handcrafted features vs "learnable" features**: all the models using a CNN to extract object features were able to achieve better accuracy than the models using handcrafted features.
- **feature aggregation**: both the Fisher Vector and standalone CNN models produce image descriptors to be separately sent to a one-vs-rest linear SVM for classification. Because there could be multiple views for a given input object, the various SVM results are aggregated by simply summing up the individual class values, being the class with the highest score returned.

Comments

The MVCNN models are capable of classifying 3D objects with relatively high accuracy, reaching the 90% mark. However, this architecture is not capable of operating with input

data representing a scene, being able to classify a single input object at a time, and the transformation of the 3D mesh to 2D views may result in the loss of important features that would help to describe the input sample. The pre-processing of the 3D objects also hinders the ability of the architecture to be deployed in real-time applications that are bound by inference time.

2.4.2 PointNet

The PointNet architecture, published by Charles Qi et al, was the first architecture to directly process and extract relevant features directly from point cloud data. Compared to other representations, PointNet is able to extract features from point clouds in their raw format, without any pre-processing steps required to transform them into multi-view or volumetric representations.

The goal of this architecture was to achieve transformation invariance, such as to rotations and scaling transformations, and permutation invariance, in order for the network to output the similar results independently of the point order of each point cloud sample.

Two iterations of PointNet were conceived:

- **PointNet (Vanilla)**, which is a classification network with the point permutation invariance property.
- **PointNet**, which is the full version of the network with both permutation and transformation invariance, capable of not only classifying objects but also perform semantic scene and object part segmentation.

PointNet (Vanilla) Architecture

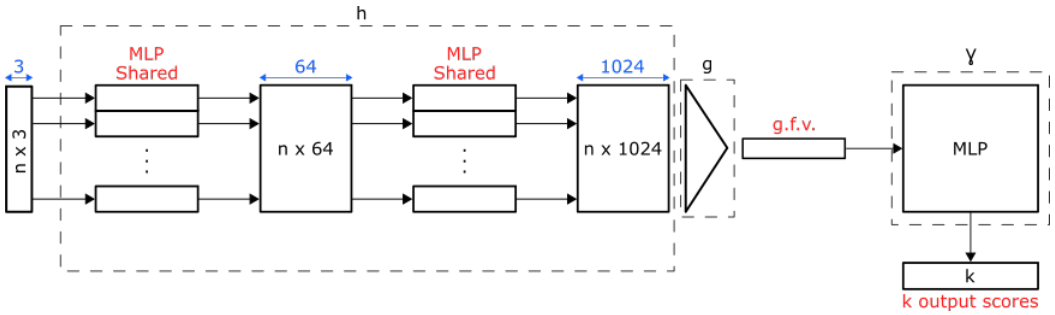


Figure 2.8: PointNet (Vanilla) architecture diagram.

To achieve the permutation invariance property, the authors' approach was to find a symmetric function to aggregate all the local (point) feature vectors into a single global feature vector, originating a vector independent of point order that describes the entire input point cloud. The chosen symmetric function was max pooling, represented by g in Figure 2.8.

Each local feature vector represents the features of a point in a higher dimensional embedding space, which corresponds to the projection of the input points from 3-dimensional vectors to 1024-dimensional vectors. This transformation is represented by h, where n points with 3 dimensions are first projected into a 64-dimensional space, before being finally projected to 1024-dimensional vectors.

This transformation is achieved through shared MLPs, which are fully connected layers with the same weight shared across all the connections. At the end, the generated global feature vector, which is the structure containing the features that characterize the entire input point cloud, is passed through a final MLP, represented by γ, for point cloud classification.

PointNet Architecture

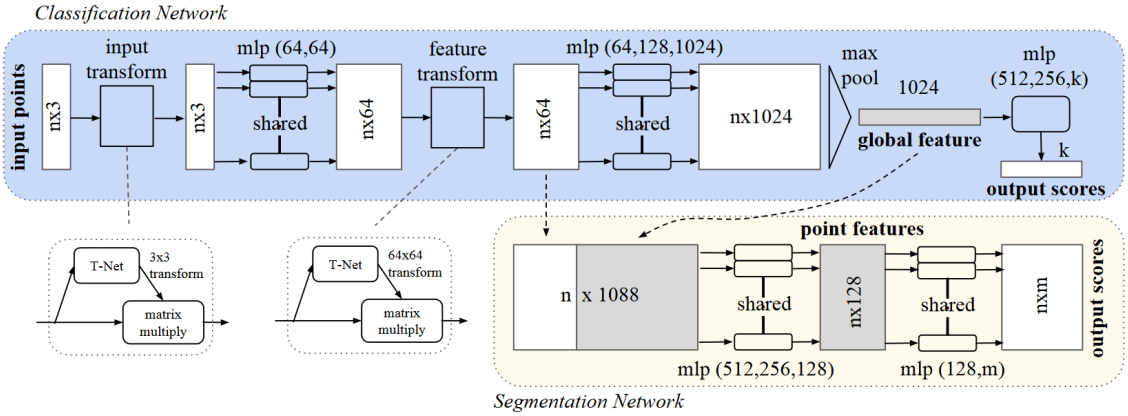


Figure 2.9: Pointnet architecture diagram [52].

While still being invariant to point permutations, the full PointNet architecture is also invariant to point cloud transformations. This property was achieved by the introduction of two networks based of the Spatial Transformation Network (STN) [31]:

- **Input Transformation Network (ITN)**: transforms the entire point cloud comprised of points with 3 dimensions.
- **Feature Transformation Network (FTN)**: transforms the entire point cloud comprised of 64-dimensional local feature vectors, that represent the point cloud points in a higher dimensional embedding space.

The purpose of these networks is to find a transformation matrix that is capable of transforming the input point cloud into its canonical pose, an ideal pose that has not been affected by any transformations.

Segmentation Network

The PointNet architecture is also capable of processing points with additional channels such as color and surface normals. For the scene semantic segmentation task, the authors used a dataset of point clouds comprised of points represented by 9-dimensional vectors, corresponding to the 3D coordinates, RGB values and normalized location. In order to compare determine the effects of handcrafted features on the inference results, the authors created a baseline model that used the same 9 channels, plus 3 additional dimensions for the local point density, local curvature and surface normal.

The portion corresponding to the Segmentation Network performs the following processing steps:

1. First, the global feature vector is separately concatenated with every local feature vector, resulting in a 1088-dimensional vector.

2. The vectors are fed to a shared MLP for a new feature extraction stage, where the previous combination of local features with the global feature vector allows the network to learn some information about the local geometry and assign quantities to each point.

3. The last shared MLP outputs a classification score for each point, where m is the number of classes with which to segment the target point cloud.

Performance Analysis

The classification accuracy of the PointNet architecture was compared with other methods that used common representations of 3D data, which were multi-view, volumetric and mesh. An analysis of the Figure 2.10 concludes that PointNet was able to outperform the state of the art mesh and volumetric approaches by a slight gap compared to the MVCNN model. The difference in performance compared to the MVCNN architecture is due to the high maturity level of the current Deep Learning architectures for image classification.

	input	#views	accuracy avg. class	accuracy overall
SPH [11]	mesh	-	68.2	-
3DShapeNets [28]	volume	1	77.3	84.7
VoxNet [17]	volume	12	83.0	85.9
Subvolume [18]	volume	20	86.0	**89.2**
LFD [28]	image	10	75.5	-
MVCNN [23]	image	80	**90.1**	-
Ours baseline	point	-	72.6	77.4
Ours PointNet	point	1	86.2	**89.2**

Figure 2.10: Classification results on ModelNet40 dataset [52].

The scene segmentation performance was also measured and compared to the baseline model, that used 3 additional handcrafted point features.

	mean IoU	overall accuracy
Ours baseline	20.12	53.19
Ours PointNet	**47.71**	**78.62**

Figure 2.11: Results on semantic segmentation in scenes provided by the Stanford 3D semantic parsing dataset [52].

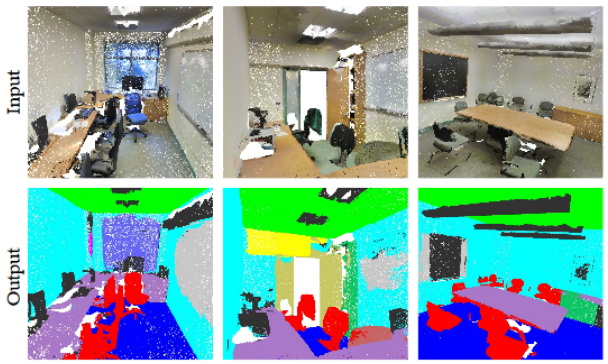


Figure 2.12: Qualitative results for semantic segmentation [52].

It can be seen in Figure 2.11 that PointNet was able to significantly outperform the baseline approach, in both mean IoU and overall accuracy, supporting the significance of "learnable" features over handcrafted ones.

By observing the qualitative results of the scene segmentation shown in Figure 2.12, it can be concluded that the network is capable of returning smooth predictions of the scene's surfaces, even when subjected to missing points and object occlusions.

Comments

The PointNet architecture is a major advancement on 3D object identification. It can directly process and evaluate raw point cloud samples, avoiding the loss of important features that occurs when converting to discrete data representations (voxels) and to a lower dimensional space (image views), it is invariant to point permutation and point cloud transformations, and supports variable point attributes.

The downsides to this architecture are the support for only single object classification, it is not capable of detecting objects in a scene, does not take into consideration local features of neighboring points, only aggregating all individual point feature vectors into a single global feature vector, may be computationally expensive when operating with higher resolution point clouds due to the use of MLPs.

2.4.3 Multi-View 3D Object Detection (MV3D)

MV3D, published by Xiaozhi Chen et al [12], is a fusion framework that receives LiDAR point clouds and RGB images as input, and predicts oriented 3D bounding boxes around targeted objects. It aims to localize and recognize 3D objects with high precision in self-driving car scenarios.

The MV3D architecture accepts as input an RGB image of the front view of the vehicle and a multi-view representation of the point cloud, consisting in a bird's eye view and a front view.

The **bird's eye view representation** is the result of encoding the point cloud using the height, intensity and density of its points into a grid. The point cloud is discretized onto a 2D grid, with resolution of 0.1m, and several maps are created using the three attributes previously mentioned:

- **Height maps**: each cell of the grid can contain multiple points of the point cloud. To build a height map, each cell is given the highest height value that it contains. In order to decrease data loss due to the selection of one height value per cell, the point cloud is vertically subdivided into equally spaced slices. For every cell, the highest height value within a slice is determined, and the result will be M height maps for M slices.
- **Intensity map**: each cell of the grid contains the highest reflectivity value from all the points within, resulting in a single intensity map.
- **Density map**: the point density is calculated for each grid cell. In order to normalize the density values into the range [0, 1], Equation 2.8 is used.

$$\min(1.0, \frac{\log(N+1)}{\log(64)}); \tag{2.8}$$

where N is the number of points in the cell. This means that the bird's eye view is encoded with M+2 channel features.

The **front view representation** of the point cloud is obtained by projecting the point cloud into a cylinder plane, and not onto an image plane, in order to generate a dense map. The projected coordinates of a point in the point cloud can be calculated with the following equations:

$$c = \left\lfloor \frac{\text{atan2}(y,\, x)}{\Delta\theta} \right\rfloor \tag{2.9}$$

$$c = \left\lfloor \frac{\text{atan2}(z,\, \sqrt{x^2 + y^2})}{\Delta\phi} \right\rfloor; \tag{2.10}$$

where $\Delta\theta$ and $\Delta\phi$ are the horizontal and vertical resolution of the LiDAR sensor, and (x, y, z) are the coordinates of a point of the point cloud. The front view map is encoded with the height, distance and intensity values.

MV3D network architecture

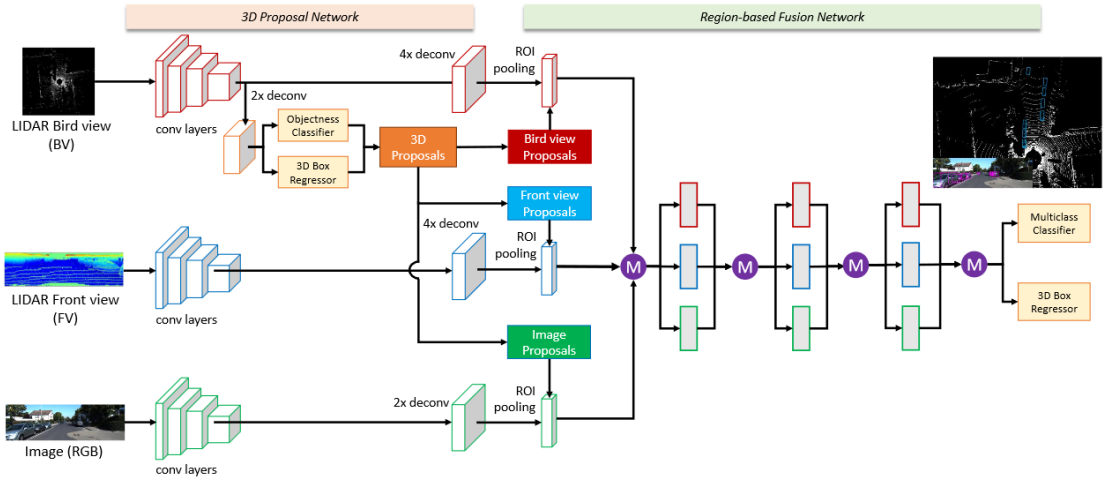


Figure 2.13: MV3D architecture diagram [12].

The MV3D architecture is divided in two parts:

1. **3D Proposal Network:** the first part of the framework extracts feature maps from the input data (bird's eye view, front view and RGB image), through successive convolution layers followed by deconvolution layers, and creates 3D bounding box proposals from the bird's eye view, to be projected onto the three input representations.

    - The bird's eye view was selected as the source for the object proposals because it preserves the objects' dimensions, they are better distributed and separate across the scene, reducing the occlusion problem, and are typically located at similar heights on the ground plane. Due to this characteristics, it is possible to obtain accurate 3D bounding boxes from the bird's eye view representation.

    - The 3D Proposal Network uses a Bounding Box Regressor to create candidate bounding boxes and determines if each proposed bounding box contains an actual object or not, which is known as Objectness Score. The regressor operates on an upsampled feature map, created by passing the bird's eye view feature map through through deconvolution layers.

2. **Region-based Fusion Network:** aims to fuse the multimodal features in order to perform multi-class classification on the 3D object proposals and predict oriented 3D bounding boxes.

    - The fusion network layers operate on aggregated feature vectors, which are the result of the element-wise mean operation between the multimodal features derived from the different views. Because of their different resolutions, the Region of Interest (RoI) Pooling operation is applied to each view feature map, resulting in three feature vectors with shared dimensions. The element-wise mean operation is marked as M in Figure 2.13.

    - The developed approach to combine multimodal feature vectors is called *Deep Fusion*. It uses a hierarchy of join operations (for example: concatenation, summation) and feature transformation functions before reaching the final multiclass classifier and 3D box regressor networks.

    - A feature transformation function is a technique used to create new features from existing ones. For each hierarchy level exists 3 of these feature transformation functions, which are networks trained to capture features from the bird's eye view, front view and RGB image.

    - It is at the last 3D box regressor network that orientation regression is considered and, in order to simplify the training of proposal bounding box regression, the orientation is restricted to $\{0^{o}, 90^{o}\}$, taking into account the most common vehicle orientations.

Performance Analysis

The MV3D network was evaluated on the KITTI object detection benchmark and uses 3D bounding box recall as the metric for the evaluation of 3D object proposals. The framework was only tested on the vehicle category. To compute the IoU, two cuboids are used (the predicted cuboid and ground truth cuboid) instead of two 2D bounding boxes.

Box proposal recall is an important evaluation metric as only the regions enveloped by the proposed bounding boxes will be subjected to further processing and predictions. So, in order to reduce the number of falsely discarded proposed bounding boxes, i.e. reduce the number of false negatives, a higher recall value is the goal.

Figure 2.14 illustrates the evolution of the recall over three different scenarios and the results of MV3D framework are compared with the Mono3D [10] and 3DOP [9] approaches:

- the left graph shows the recall value as the IoU threshold increases from 0 to 1. The number of proposals used for this comparison is 300.
- the middle graph shows the recall value for various numbers of proposals, with a fixed IoU threshold of 0.25.
- the right graph also shows the recall value for various numbers of proposals, but with a fixed IoU threshold of 0.5.

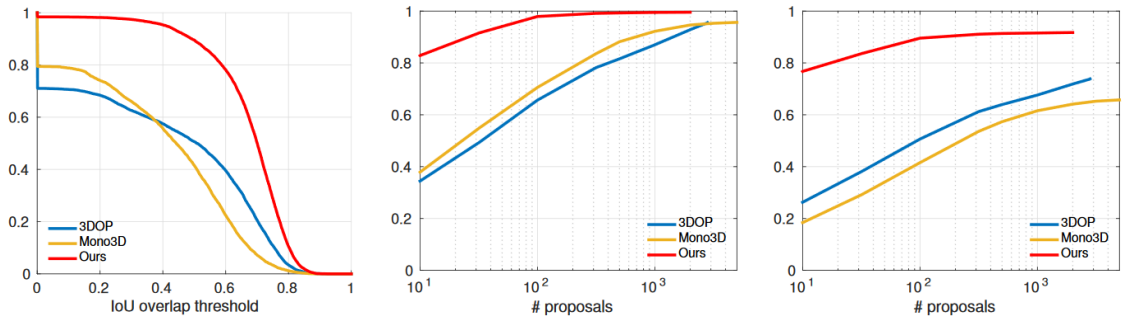


Figure 2.14: 3D bounding box proposal Recall [12].

The Average Precision score (AP_{loc}) is used as the metric for evaluating 3D bounding boxes projected onto the bird's eye view, where their orientation is taken into account.

Method	Data	IoU=0.5			IoU=0.7		
		Easy	Moderate	Hard	Easy	Moderate	Hard
Mono3D [3]	Mono	30.5	22.39	19.16	5.22	5.19	4.13
3DOP [4]	Stereo	55.04	41.25	34.55	12.63	9.49	7.59
VeloFCN [17]	LIDAR	79.68	63.82	62.80	40.14	32.08	30.47
Ours (BV+FV)	LIDAR	95.74	88.57	88.13	86.18	77.32	76.33
Ours (BV+FV+RGB)	LIDAR+Mono	**96.34**	**89.39**	**88.67**	**86.55**	**78.10**	**76.67**

Figure 2.15: 3D localization performance on vehicles, using the KITTI validation set [12].

The 3D object detection evaluation consists in calculating the Average Precision taking into account both predicted 3D bounding boxes and bird's eye view bounding boxes. Their orientation is implicitly considered as both types have oriented bounding boxes.

Method	Data	IoU=0.25			IoU=0.5			IoU=0.7		
		Easy	Moderate	Hard	Easy	Moderate	Hard	Easy	Moderate	Hard
Mono3D [3]	Mono	62.94	48.2	42.68	25.19	18.2	15.52	2.53	2.31	2.31
3DOP [4]	Stereo	85.49	68.82	64.09	46.04	34.63	30.09	6.55	5.07	4.1
VeloFCN [17]	LIDAR	89.04	81.06	75.93	67.92	57.57	52.56	15.20	13.66	15.98
Ours (BV+FV)	LIDAR	96.03	88.85	88.39	95.19	87.65	80.11	71.19	56.60	55.30
Ours (BV+FV+RGB)	LIDAR+Mono	**96.52**	**89.56**	**88.94**	**96.02**	**89.05**	**88.38**	**71.29**	**62.68**	**56.56**

Figure 2.16: 3D detection performance on vehicles, using the KITTI validation set [12].

An ablation study aims to evaluate the impact of each input representation and their combinations on the performance of the model. As observed in Figure 2.17, the bird's eye view had the most impact in the majority of the Average Precision scores, followed by the RGB view. The relevance of both bird's eye view and RGB view can be verified due to their pair combination resulting in the highest AP scores.

Overall, the best performance achieved by the architecture was with the feature combination from all the views.

Data	AP_{3D} (IoU=0.5)			AP_{loc} (IoU=0.5)			AP_{2D} (IoU=0.7)		
	Easy	Moderate	Hard	Easy	Moderate	Hard	Easy	Moderate	Hard
FV	67.6	56.30	49.98	74.02	62.18	57.61	75.61	61.60	54.29
RGB	73.68	68.86	61.94	77.30	71.68	64.58	83.80	76.45	73.42
BV	92.30	85.50	78.94	92.90	86.98	86.14	85.00	76.21	74.80
FV+RGB	77.41	71.63	64.30	82.57	75.19	66.96	86.34	77.47	74.59
FV+BV	95.19	87.65	80.11	95.74	88.57	88.13	88.41	78.97	78.16
BV+RGB	**96.09**	88.70	80.52	**96.45**	89.19	80.69	89.61	**87.76**	79.76
BV+FV+RGB	96.02	**89.05**	**88.38**	96.34	**89.39**	**88.67**	**95.01**	87.59	**79.90**

Figure 2.17: Ablation study of multi-view features, for 3D object detection on vehicles [12].

Comments

The MV3D architecture was developed for 3D object detection in scenes represented by sparse point clouds, as the ones obtained from the LiDAR sensor.

However, the used approach relies on the projection of the point cloud onto a grid in order to create height maps, a density map and an intensity map. This projection and discretization of the original point cloud results in loss of information and data distortion. MV3D also uses the VGG-16 network [65], an already outdated image classification network, that requires a large number of parameters, thus not suitable for real-time application in autonomous vehicles.

2.4.4 PointNet++

The original PointNet architecture attempted to learn a spatial encoding of each point of the point cloud and aggregate all of their individual signatures into a global feature vector.

By design, the original PointNet did not capture features from local structures of the point cloud but, as demonstrated by the success of CNNs, this is an important step towards a better understanding of the scene. As such, PointNet++, published by Charles Qi et al [53], focuses on the hierarchical feature extraction from a set of sampled points of the point cloud, in euclidean space.

PointNet++ network architecture

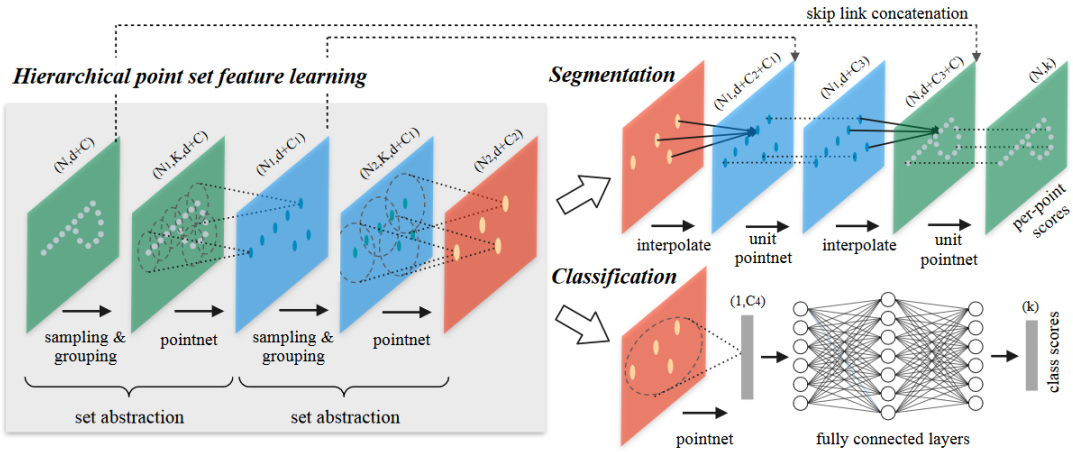


Figure 2.18: PointNet++ architecture diagram [53].

The hierarchical feature extraction consists in a sequence of "Set Abstracion" levels, which abstract point cloud local regions by encoding these into feature vectors. For single scale feature extraction, where the variability of point density in the local regions is not considered, each level is comprised of the following layers and operations:

- **Sampling layer:** The first step to group local sets of points is to determine their centroids. The sampling of the point cloud can be accomplished by randomly selecting N' points. However, this can lead to the selection of points close to each other and can potentially miss prominent local structures. Instead, in order to better cover the entire point cloud, the iterative Furthest Point Sampling algorithm is used. This technique finds the N' furthest points of the point cloud to create the set of cluster centroids.
- **Grouping layer:** Given the centroids found in the previous layer, the Grouping layer employs the Ball Query approach, illustrated in Figure 2.19, to group the neighboring points of each cluster centroid. Ball query was chosen instead of the K-Nearest Neighbors (KNN) algorithm because of its ability to guarantee a fixed scale for the neighboring volume from where the points are selected.

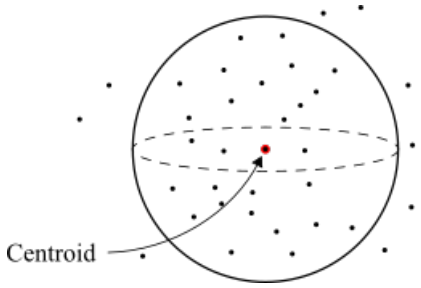


Figure 2.19: Formation of point groups using the ball query approach.

- **PointNet layer:** Each set of points, comprising a local region and the correspondent centroid, is separately passed to the PointNet layer for the feature extraction step. The output is a pair formed by the centroid and the local region feature vector that encodes its neighborhood.

The PointNet implementation used in this layer does not include the feature transformation networks, and it is called "mini PointNet". It needs to be capable of outputting a feature vector of fixed length for local regions with variable number of points, that are obtained by using the Ball Query algorithm.

The non-uniform sampling density, caused by density variations of points on the local region volumes, can affect the performance of the architecture due to the inability of the learned features to generalize well for both sparse and dense regions. This is the case when using Single-Scale Grouping (SSG), where a fixed range is used for the ball query algorithm to group neighboring points.

To overcome this problem, the authors propose two types of density adaptive layers to extract and combine multiple scales of local pattern features:

- **Multi-Scale Grouping (MSG):** This approach consists in defining several ranges / scales for the Ball Query algorithm to find points in the neighborhood of the cluster centroids, i.e. each cluster centroid will be associated to several local regions of different scales, as illustrated in Figure 2.20. The PointNet layer receives each local region's set of points separately and generates local region feature vectors for each scale. At the end, all the local region feature vectors associated to same centroid are concatenated into a single multi-scale feature vector. This approach has the disadvantage of being computationally expensive because large scale neighborhoods need to be processed for every centroid

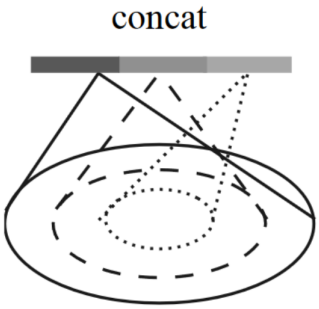


Figure 2.20: Creation of the multi-scale feature vector using MSG [53].

- **Multi-Resolution Grouping (MRG):** the feature vector of a local region, at some scale level L_i, is the result of the concatenation of two feature vectors, as illustrated in Figure 2.21:
  - the right part of the feature vector is obtained by passing each subregion through the set abstraction level and summarizing the resulting feature vectors.
  - the left part of the feature vector corresponds to the global feature vector returned by a mini PointNet, after being fed the region's set of points.

  The ability of the two vectors to capture useful features varies according to the density of the local region: the left feature vector is more reliable when the local region is denser in contrast to the right feature vector, which becomes more reliable when the local region is sparser.

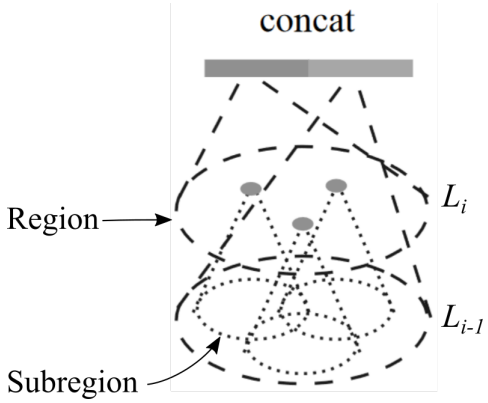


Figure 2.21: Creation of the multi-scale feature vector using MRG, adapted from [53].

Similar to PointNet, PointNet++ is capable of classifying and segmenting the input point clouds:

- **Classification submodule**: for the classification step, the network uses each local region feature vector as input to the mini PointNet, as if they were points. They are then passed through am MLP to produce the classification scores.

- **Segmentation submodule**: given that the Set Abstraction layers subsample the original point cloud, and the need for feature extraction per point for achieving semantic segmentation, the applied solution involved the propagation of the features from the subsampled points to the originals.

  Like the hierarchical approach to feature learning, the authors adopted a hierarchical propagation strategy for extracting features for each point of the original point cloud. Each level of the hierarchy applies the same two processing steps to the corresponding input data, and the following description is related to the first level of the Segmentation submodule:

  1. **Interpolation**: the output of the second Set Abstraction layer, which consists of signatures for the correspondent subsampled group of points, is subjected to interpolation of type inverse distance weighted average based on k nearest neighbors, in order to generate the same number of feature vectors outputted by the first Set Abstraction layer.

  2. **Unit PointNet**: after generating the feature vectors, each is concatenated with the corresponding feature vector that belonged to the input of the second Set Abstraction layer. To update these new feature vectors, whose parts are known due to the concatenation, they are passed through a Unit PointNet. Each feature vector is passed individually and, because the Unit PointNet is similar to one-by-one convolutions in CNNs, the output is a feature vector of the same length.

  This process is repeated until the network originates a feature vector per point of the original point cloud.

Performance Analysis

To train and test the proposed architecture variations, the authors converted the ModelNet40 mesh samples to point clouds by samples points on the surface of the 3D models.

In an effort to increase the model's performance, a second test was performed by modifying the existing dataset. Instead of 1024 points per point cloud, 5000 points were sampled from each sample of the ModelNet40 dataset, and the corresponding surface normals were added to each point. The results correponds to the entry "Ours (with normals)" in Figure 2.22.

Method	Input	Accuracy (%)
Subvolume [21]	vox	89.2
MVCNN [26]	img	90.1
PointNet (vanilla) [20]	pc	87.2
PointNet [20]	pc	89.2
Ours	pc	90.7
Ours (with normal)	pc	**91.9**

Table 2: ModelNet40 shape classification.

Figure 2.22: Classification results on the ModelNet40 dataset [53].

The new approach was able to achieve better performance than the original PointNet architectures in both classification experiments, with the same number of sampled 1024 points per point cloud and only using point coordinate features.

A bigger increase in performance occurs when the point clouds are denser, with 5000 sampled points, and with the added surface normals information to the points' features, being able to surpass the performance of the MVCNN network.

Because of the constant presence of sampling irregularities in the data gathered by sensors, the model was tested by randomly dropping points during test time. The results of this experiment are shown in Figure 2.23, which compares the performance difference between training with and without random input dropout (DP), and between the approaches for addressing sampling density variations: Single-Scale Grouping (SSG), MSG and MRG.

By observing the graph in Figure 2.23, it is possible to make the following conclusions:

- the performance of vanilla PointNet drops relatively fast with sparser point clouds but, with DP it is able to maintain its accuracy until the 256 points mark.

- the PointNet++ with SSG is able to achieve good performance with a dense point cloud of 1024 points, but it quickly falls with sparser point clouds. With DP it is able to better maintain the performance but, at the 512 points mark, starts to decrease due to the single-scale feature vectors.

- the best performance is achieved by the PointNet++ with MSG, followed by the more efficient PointNet++ with MRG.

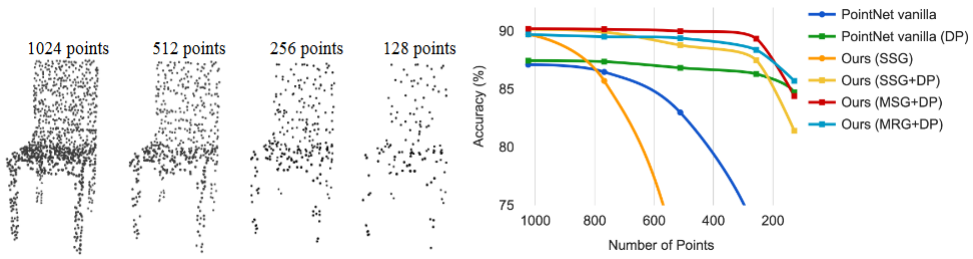


Figure 2.23: Point clouds with random point dropout on the left, and the accuracy across different grouping and training techniques with varying number of sampled points per point cloud [53].

Comments

By taking advantage of the neighboring points and considering different search scales, the PointNet++ was able to outperform the MVCNN and the PointNet architectures.

On the other hand, the performance on more sparse point clouds is very poor, as shown in Figure 2.23, probably because the network could not take advantage of the new strategy that considers neighboring points, as there are significantly less. This is important because LiDAR sensors produce highly sparse point clouds, sometimes ending up representing an object with very few points. As the previous iteration, PointNet++ does not perform object detection, it only classifies a single point cloud object.

2.4.5 VoxelNet

Previous attempts on Region Proposal Networks for sparse point clouds focused on hand-crafted feature engineering, such as the bird's eye view projection in MV3D.

In contrast, VoxelNet, published by Yin Zhou et al [79], was developed to be a generic framework for 3D object detection, performing both feature extraction and bounding box prediction of sparse point clouds with an end-to-end trainable network.

VoxelNet Architecture

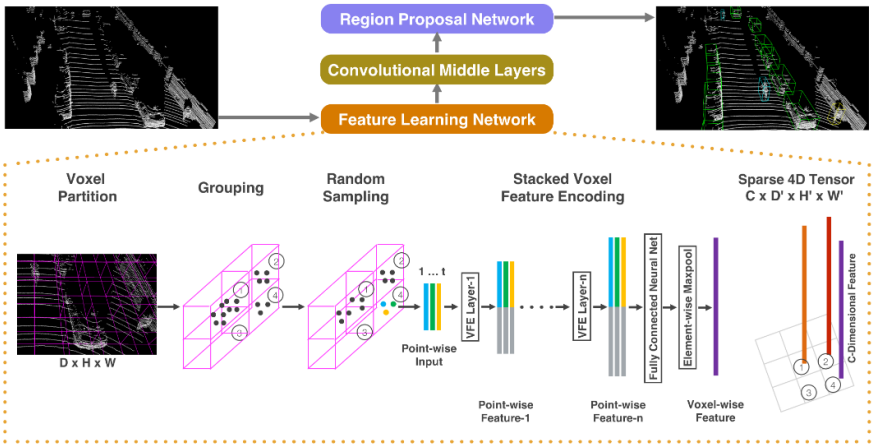


Figure 2.24: VoxelNet architecture diagram [79].

The VoxelNet architecture is subdivided into 3 functional blocks:

1. **Feature Learning Network:**

   (a) **Voxel Partition:** this step receives the raw point cloud as input and partitions the 3D space into a 3D voxel grid.

   (b) **Grouping of points:** points of the point cloud that reside on the same voxel are grouped together. Each voxel will have a variable number of points due to the non-uniform sampling, object occlusion, among other factors related to the sensor's measurements.

   (c) **Random Sampling:** high resolution LiDAR sensors build point clouds with hundreds of thousands of points, which leads to high memory requirements and computational costs. To avoid these problems, random sampling is applied to every voxel to limit the number of points within.

   (d) **Stacked Voxel Feature Encoding:** a Voxel Feature Encoding (VFE) layer encodes the interactions between the points within a voxel and concatenates the resulting feature vector with each point-wise input of the layer, forming a point-wise feature, as shown in the Figure 2.25.

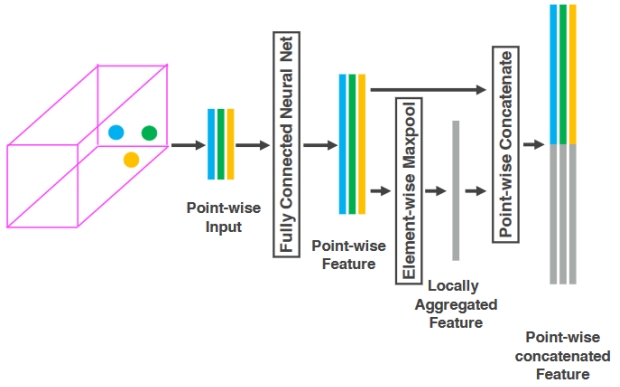


Figure 2.25: Architecture of a Voxel feature encoding layer [79].

To capture features of a voxel, the process starts by passing the point-wise input (the group of points within the voxel) to an MLP in order to map each input point vector to a higher dimensional vector, resulting in point-wise features. Then, the resulting feature vectors are combined, using the element-wise maxpool operation, in order to produce a locally aggregated feature vector, which is the signature of the voxel currently being processed. Finally, to allow the chaining of VFE layers, each point feature vector obtained from the MLP is concatenated with the locally aggregated feature vector, resulting in point-wise concatenated features. Taking into account all the voxels of the grid, the result of a VFE layer is a list of voxel feature vectors (one vector per voxel).

   (e) **Sparse Tensor Representation:** each voxel feature on the list is associated with the spatial coordinates of a non-empty voxel.

   To represent the list of voxel-wise features, a sparse 4D tensor is used, where for

each voxel will be a feature vector. Figure 2.26 illustrates a voxel feature vector within the 4D tensor:

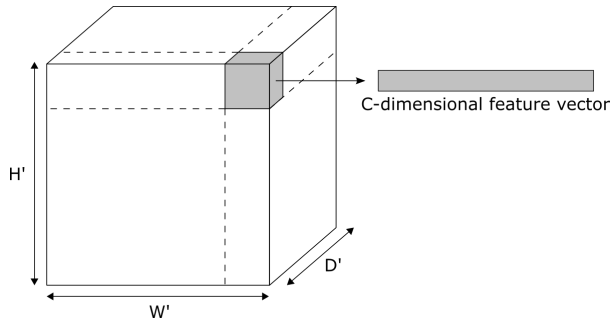


Figure 2.26: Illustration of a voxel feature vector in a 4D tensor. The total number of voxels in the grid is equal to H' ∗ W' ∗ D'.

2. **Convolutional Middle Layers:** this functional block contains a sequence of 3D convolution layers, each followed by a Batch Normalization layer and ReLU, to aggregate the voxel-wise features with an expanding filter size, enabling a hierarchical feature extraction at increasing abstraction levels.

3. **Region Proposal Network:** the RPN used by VoxelNet is a modified version of the Faster RCNN, in order to adapt to point cloud input data. It is integrated with the 2 previous blocks to enable end-to-end learning, and outputs a probability score map and a regression map.

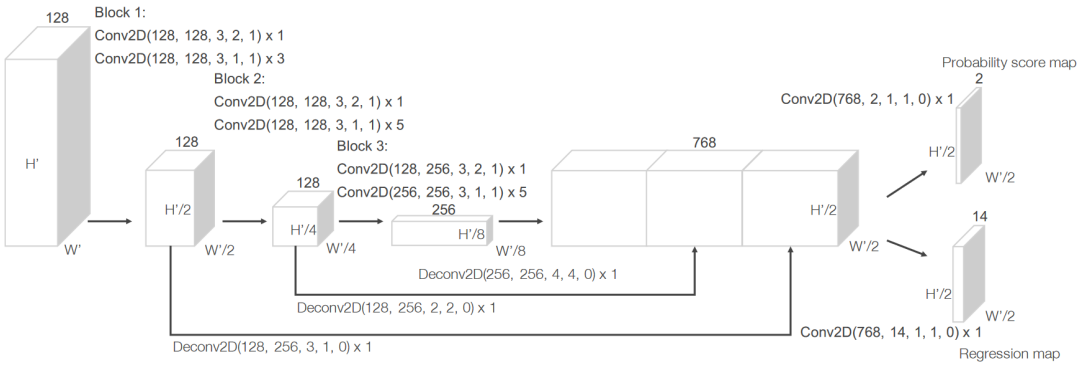


Figure 2.27: Region proposal network architecture [79].

Performance Analysis

The VoxelNet is evaluated on the KITTI 3D object detection benchmark, for the *Car*, *Pedestrian* and *Cyclist* categories. The network is trained with only the LiDAR data from KITTI. To understand the performance advantage of end-to-end learning, a VoxelNet-based model using hand-crafted features was implemented and used as the test's baseline. This architecture was called HC-baseline, and uses the bird's eye view approach described in [12] for the handcrafted feature extraction step.

The Figure 2.28 shows the performance results in the bird's eye view detection, which evaluates the localization of objects in the ground plane (2D plane).

The HC-baseline was able to outperform the MV3D network in the bird's eye view detection, which demonstrated the effectiveness of the implemented RPN. However, the VoxelNet achieved the greatest AP across all the categories and difficulty levels, proving its dominance over the hand-crafted approach used by the baseline network.

The dimensions of the detectable objects is an important factor that contributes to better or worst performance. When a small object is sampled by the LiDAR sensor, it is represented by a small number of points, which hinders the performance of the network.

Method	Modality	Car			Pedestrian			Cyclist		
		Easy	Moderate	Hard	Easy	Moderate	Hard	Easy	Moderate	Hard
Mono3D [3]	Mono	5.22	5.19	4.13	N/A	N/A	N/A	N/A	N/A	N/A
3DOP [4]	Stereo	12.63	9.49	7.59	N/A	N/A	N/A	N/A	N/A	N/A
VeloFCN [22]	LiDAR	40.14	32.08	30.47	N/A	N/A	N/A	N/A	N/A	N/A
MV (BV+FV) [5]	LiDAR	86.18	77.32	76.33	N/A	N/A	N/A	N/A	N/A	N/A
MV (BV+FV+RGB) [5]	LiDAR+Mono	86.55	78.10	76.67	N/A	N/A	N/A	N/A	N/A	N/A
HC-baseline	LiDAR	88.26	78.42	77.66	58.96	53.79	51.47	63.63	42.75	41.06
VoxelNet	LiDAR	**89.60**	**84.81**	**78.57**	**65.95**	**61.05**	**56.98**	**74.41**	**52.18**	**50.49**

Figure 2.28: Performance comparison in bird's-eye view detection: AP (in %) on KITTI validation set [79].

Figure 2.29 shows the performance results in 3D object detection, which is a more challenging test that requires localization of objects in 3D space.

Method	Modality	Car			Pedestrian			Cyclist		
		Easy	Moderate	Hard	Easy	Moderate	Hard	Easy	Moderate	Hard
Mono3D [3]	Mono	2.53	2.31	2.31	N/A	N/A	N/A	N/A	N/A	N/A
3DOP [4]	Stereo	6.55	5.07	4.10	N/A	N/A	N/A	N/A	N/A	N/A
VeloFCN [22]	LiDAR	15.20	13.66	15.98	N/A	N/A	N/A	N/A	N/A	N/A
MV (BV+FV) [5]	LiDAR	71.19	56.60	55.30	N/A	N/A	N/A	N/A	N/A	N/A
MV (BV+FV+RGB) [5]	LiDAR+Mono	71.29	62.68	56.56	N/A	N/A	N/A	N/A	N/A	N/A
HC-baseline	LiDAR	71.73	59.75	55.69	43.95	40.18	37.48	55.35	36.07	34.15
VoxelNet	LiDAR	**81.97**	**65.46**	**62.85**	**57.86**	**53.42**	**48.87**	**67.17**	**47.65**	**45.11**

Figure 2.29: Performance comparison in 3D object detection using AP (in %) on KITTI validation set, with an IoU of 0.7 for the car category, and 0.5 for the pedestrian and cyclist categories [79].

Even though VoxelNet only uses point cloud data to perform object detection, whereas MV3D uses a combination of point cloud, fornt view image and handcrafted features in form of different grid maps, it is capable of outperforming the MV3D architecture on 3D object detection of vehicles with an IoU of 0.7.

Comments

In contrast to MV3D, VoxelNet only makes use of point cloud data to perform object detection in a scene and is capable of detecting cars, pedestrians and cyclist, when MV3D is only capable of detecting cars. VoxelNet also does not rely on handcrafted features, increasing the generalization of the trained model.

The downside of this architecture is the performance. Even though it surpasses previous state of the art, this architecture does not reach high enough performance to be deployed in the real world.

2.4.6 Frustum PointNet

Previous approaches to 3D object localization use the sliding window or the 3D RPN technique, which can be computationally expensive for larger scenes, thus not suitable for real-time applications that require high reaction times.

The Frustum PointNet framework, published by Charles Qi et al, addresses this issue by first predicting 2D bounding boxes on the front view image of the scene, using already established 2D object detection architectures instead of generating object proposals from the point cloud, reducing the proposal generation time. This is illustrated in Figure 2.30.

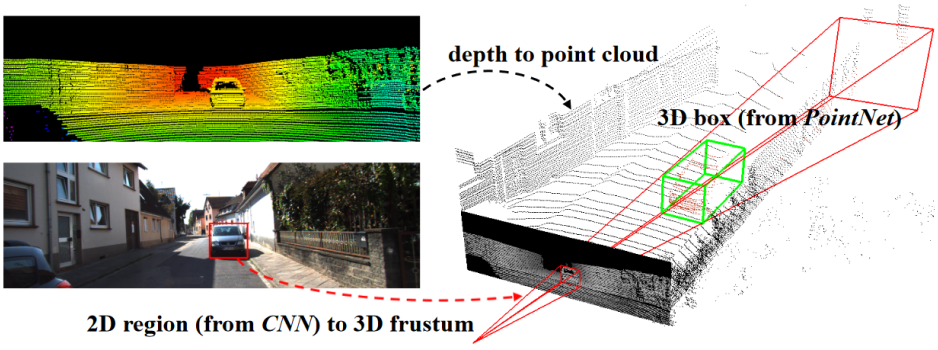


Figure 2.30: Depiction of the process performed by Frustum PointNet in order to detect a vehicle in a sample [51].

The inference of amodal bounding boxes, performed by Frustum PointNet, consists on inferring 2D or 3D bounding boxes that enclose the entire objects, even when parts of said objects are missing in the sample data due to object occlusion, reflectivity, among others.

Frustum PointNet Architecture

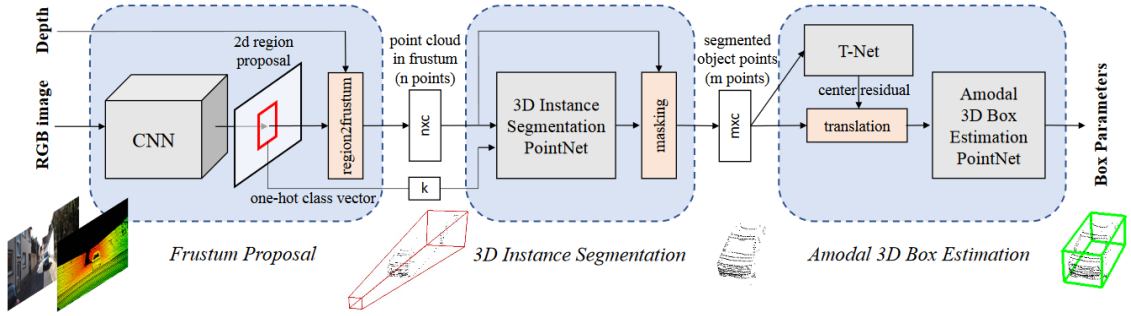


Figure 2.31: Frustum PointNet architecture diagram [51].

The Frustum PointNet architecture is divided into 3 modules, as illustrated in Figure 2.31:

- **Frustum Proposal:** the input of the 2D object detection model is the front view RGB image. The authors used the Feature Pyramid Networks (FPN) architecture, but the framework is independent of the 2D object detection algorithm used. This network is pre-trained using ImageNet classification and COCO object detection datasets, and is fine-tuned using the KITTI 2D object detection dataset for classification and prediction of amodal 2D bounding boxes.

Through a transformation matrix, the 2D bounding boxes can be stretched depth-wise in order to obtain frustums, with a near and far planes specific to the range of the depth sensor. Each frustum envelopes a group of points of the point cloud, leading to the creation of a set of frustum point clouds.

- **3D Instance Segmentation:** each frustum point cloud is given to this module in order to predict a probability score for each point, to distinguish if it belongs to the object, or to the background, within the frustum. "Masking" is the name given to the process of extracting only the points of the frustum point cloud that correspond the detected object.

  This algorithm is implemented using a PointNet-based Network, and uses the object classification result obtained from the 2D object detection network to know the class of the object associated to the frustum point cloud.

- **3D Amodal Bounding Box Estimation:** this module receives the segmented object points as input and uses a bounding box regression PointNet with a pre-processing transformer network to estimate the amodal oriented 3D bounding box of the object.

Performance Analysis

The authors tested Frustum PointNet using two versions of their previous works:

- **Frustum PointNet v1**: uses the PointNet architecture as the backbone for feature extraction.
- **Frustum PointNet v2**: uses the PointNet++ architecture as the backbone.

Both versions are first evaluated using the KITTI test dataset, and the results of the 3D object localization and 3D object detection are shown in Figures 2.32 and 2.33.

The analysis of the bird's eye view detection test results, shown in Figure 2.32, reveals that the Frustum PointNet method was able to outperform state of the art approaches that used 3D CNNs on voxelized point clouds and handcrafted feature extraction through the bird's eye view.

Method	Cars			Pedestrians			Cyclists		
	Easy	Moderate	Hard	Easy	Moderate	Hard	Easy	Moderate	Hard
DoBEM [42]	36.49	36.95	38.10	-	-	-	-	-	-
3D FCN [17]	69.94	62.54	55.94	-	-	-	-	-	-
MV3D [6]	86.02	76.90	68.49	-	-	-	-	-	-
Ours (v1)	87.28	77.09	67.90	55.26	47.56	42.57	73.42	59.87	52.88
Ours (v2)	**88.70**	**84.00**	**75.33**	**58.09**	**50.22**	**47.20**	**75.38**	**61.96**	**54.68**

Figure 2.32: 3D object localization AP score (bird's eye view) on KITTI test set [51].

The Figure 2.33 also shows remarkable improvements on on 3D object detection, where both versions accomplish better performance over the state of the art approaches.

As seen in the analysis of the PointNet++ paper, the performance of the network degrades with progressively smaller objects, where the number of points that represent them is scarce.

Method	Cars			Pedestrians			Cyclists		
	Easy	Moderate	Hard	Easy	Moderate	Hard	Easy	Moderate	Hard
DoBEM [42]	7.42	6.95	13.45	-	-	-	-	-	-
MV3D [6]	71.09	62.35	55.12	-	-	-	-	-	-
Ours (v1)	80.62	64.70	56.07	50.88	41.55	38.04	69.36	53.50	52.88
Ours (v2)	**81.20**	**70.39**	**62.19**	**51.21**	**44.89**	**40.23**	**71.96**	**56.77**	**50.39**

Figure 2.33: 3D object detection 3D AP on KITTI test set [51].

Even though the Frustum PointNet has no direct sensor fusion or multi-view aggregation steps, it is able to achieve better performance across all object categories.

To compare the performance of the two Frustum PointNet versions with a larger number of architectures, they were also evaluated with the KITTI validation set, producing the results shown in Figure 2.34.

Method	Easy	Moderate	Hard
Mono3D [4]	5.22	5.19	4.13
3DOP [5]	12.63	9.49	7.59
VeloFCN [17]	40.14	32.08	30.47
MV3D (LiDAR) [6]	86.18	77.32	76.33
MV3D [6]	86.55	78.10	**76.67**
Ours (v1)	87.82	82.44	74.77
Ours (v2)	**88.16**	**84.02**	76.44

3D object detection AP on KITTI vallidatio set (cars only).

Method	Easy	Moderate	Hard
Mono3D [4]	2.53	2.31	2.31
3DOP [5]	6.55	5.07	4.10
VeloFCN [17]	15.20	13.66	15.98
MV3D (LiDAR) [6]	71.19	56.60	55.30
MV3D [6]	71.29	62.68	56.56
Ours (v1)	83.26	69.28	62.56
Ours (v2)	**83.76**	**70.92**	**63.65**

3D object localization AP on KITTI validation set (cars only).

Figure 2.34: 3D object detection and localization using the KITTI val set [51].

The Frustum PointNet architecture, in contrast to previous state of the art approaches, is capable of 3D object detection in both indoor and outdoor scenes. For instance, the performance of approaches like the MV3D, which rely on the separation of objects in bird's eye view, do not extend well to indoor scenes. Other approaches focused on indoor scenes may also not be able to be extended to highly sparse point clouds created from outdoor scenes.

Comments

Frustum PointNet is capable of generating object proposals relatively faster than other architectures by leveraging state of the art 2D object detection architectures, and is able of outperforming the previous architectures in both 3D object detection and localization.

The downside of this architecture are still the performance results, not being high enough to guarantee reliability when deployed in autonomous vehicles, and the fact that it relies on the camera. If the camera is obstructed or malfunctions, the autonomous vehicle would not have a redundancy sensor for object detection to fallback. RaDAR alone is not capable of providing enough information to allow the continuous navigation of the vehicle.

2.4.7 Complex-YOLO

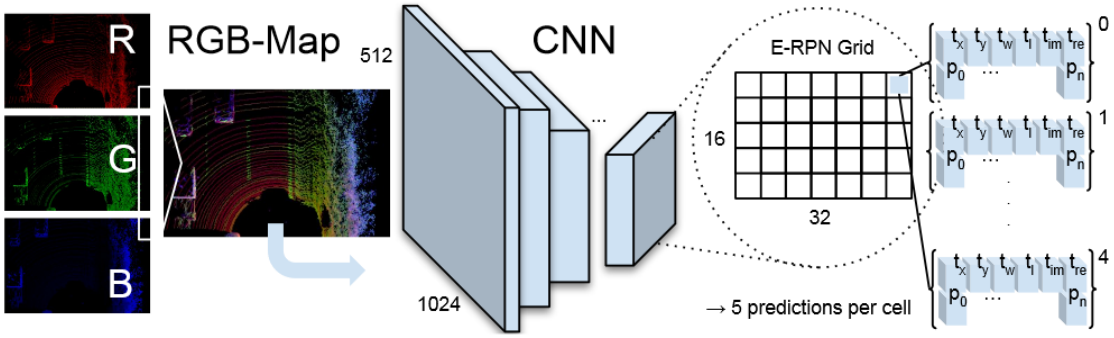


Figure 2.35: Complex-YOLO pipeline [62].

Previous 3D object detection architectures for autonomous driving, mainly the VoxelNet, Frustum PointNet and MV3D, are not suitable for deployment due to their limited inference rate on high-end GPUs, which is around 7 fps. The Complex-YOLO architecture, published by Martin Simon et al [62], is capable of running at 50 fps on a Titan X GPU, and not only predicts the trajectory of surrounding object, using the Euler-Region-Proposal Network (E-RPN), but also simultaneously estimates all classes in one forward path.

Complex-YOLO Architecture

This framework, illustrated in Figure 2.35, is divided into 2 stages:

- **Point Cloud Preprocessing stage:** the preprocessing stage transforms the point cloud into an RGB image by projecting and discretizing it into a 2D grid, based on the height (G), intensity (B) and density (R) of the points. To capture more detail and reduce the errors of point cloud quantization, the grid cells are $0.08 \times 0.08\text{m}^2$, in contrast to the MV3D's $0.1 \times 0.1\text{m}^2$ grid cells. Only part of the entire point cloud is considered by the establishment of predefined coordinate ranges (the LiDAR is 1.73m above ground and the considered maximum detection height is 3m). The result of this stage is a bird's eye view RGB map that covers an area of $80 \times 40\text{m}^2$ in front of the origin of the LiDAR sensor, as illustrated in Figure 2.36.

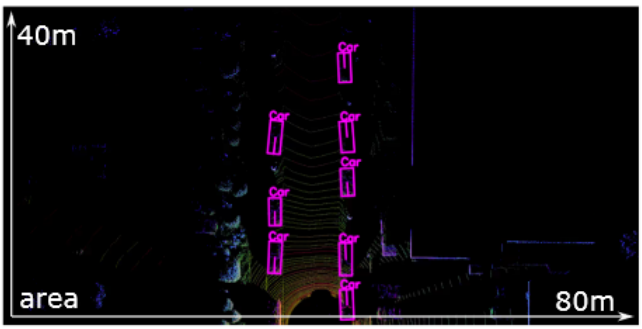


Figure 2.36: Example of a bird's eye view RGB map, disregarding the bounding boxes [62].

- **Complex-YOLO Network:** the obtained RGB map is supplied to a simplified YOLOv2 CNN network [55], extended by Complex Angle Regression, via the E-RPN, to predict five oriented 3D boxes per grid cell of the output feature map of the CNN (the transition from 2D to 3D is done by a predefined height based on each object class), including a probability score and class score.

  The E-RPN parses the 3D object position, dimensions, class scores and orientation from the output feature map of the CNN.

Performance Analysis

All the tested architectures were run on a Nvidia Titan X and on a Titan XP, with the exception of the "Complex-YOLO TX2", which was deployed in a dedicated embedded platform (TX2).

The results show that while keeping similar performance levels, the Complex-YOLO architecture is capable of running 5 times faster on accelerated-hardware, compared to its pears.

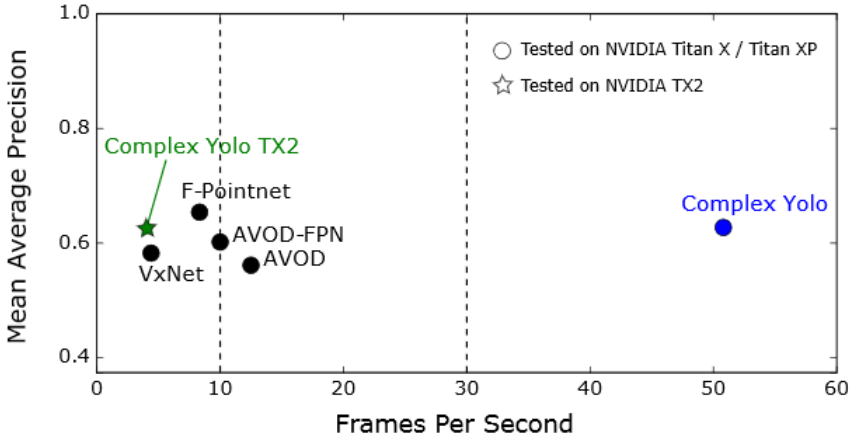


Figure 2.37: Runtime performance comparison using mAP [62].

It is important to consider that the following results of the Complex-YOLO architecture were achieved through only point cloud data, without the aid of additional input information such as images.

The Complex-YOLO network was tested using the KITTI validation dataset, with a split of 85% for training and 15% for validation, whereas the other architectures used the official KITTI test set and the dataset split of 50% for training and 50% for validation.

Its performance was similar to other state of the art approaches, only able to outperform in the *Cyclist* category by 10%. However these results were achieved while running at 50.4 fps, as shown in Figure 2.38.

Method	Modality	FPS	Car			Pedestrian			Cyclist		
			Easy	Mod.	Hard	Easy	Mod.	Hard	Easy	Mod.	Hard
MV3D [2]	Lidar+Mono	2.8	86.02	76.90	68.49	-	-	-	-	-	-
F-PointNet [5]	Lidar+Mono	5.9	88.70	84.00	75.33	**58.09**	**50.22**	**47.20**	**75.38**	61.96	54.68
AVOD [7]	Lidar+Mono	12.5	86.80	**85.44**	77.73	42.51	35.24	33.97	63.66	47.74	46.55
AVOD-FPN [7]	Lidar+Mono	10.0	88.53	83.79	**77.90**	50.66	44.75	40.83	62.39	52.02	47.87
VoxelNet [3]	Lidar	4.3	**89.35**	79.26	77.39	46.13	40.74	38.11	66.70	54.76	50.55
Complex-YOLO	Lidar	**50.4**	85.89	77.40	77.33	46.08	45.90	44.20	72.37	**63.36**	**60.27**

Figure 2.38: Performance on bird's eye view detection using AP [62].

The 3D object detection is a more challenging task, resulting in less favorable results for the car category, but reasonable results for the other two categories relative to the other networks, as shown in Figure 2.39.

Method	Modality	FPS	Car			Pedestrian			Cyclist		
			Easy	Mod.	Hard	Easy	Mod.	Hard	Easy	Mod.	Hard
MV3D [2]	Lidar+Mono	2.8	71.09	62.35	55.12	-	-	-	-	-	-
F-PointNet [5]	Lidar+Mono	5.9	81.20	70.39	62.19	**51.21**	**44.89**	**40.23**	**71.96**	56.77	50.39
AVOD [7]	Lidar+Mono	12.5	73.59	65.78	58.38	38.28	31.51	26.98	60.11	44.90	38.80
AVOD-FPN [7]	Lidar+Mono	10.0	**81.94**	**71.88**	**66.38**	46.35	39.00	36.58	59.97	46.12	42.36
VoxelNet [3]	Lidar	4.3	77.47	65.11	57.73	39.48	33.69	31.51	61.22	48.36	44.37
Complex-YOLO	Lidar	**50.4**	67.72	64.00	63.01	41.79	39.70	35.92	68.17	**58.32**	**54.30**

Figure 2.39: Performance comparison for 3D object detection using AP [62].

Comments

Complex-YOLO is capable of detecting objects significantly faster than previous state of the art, reaching 50 fps, and the accuracy is not significantly affected by the low inference time, as its performance closely matches the other architectures. It is only based on point cloud input data.

Aside from the lower performance scores, other downsides of this architecture rely on the fact that it does not operate directly on the input point cloud and applies a pre-processing step that discretizes it, which could limit further improvements to the model.

2.4.8 PointRCNN

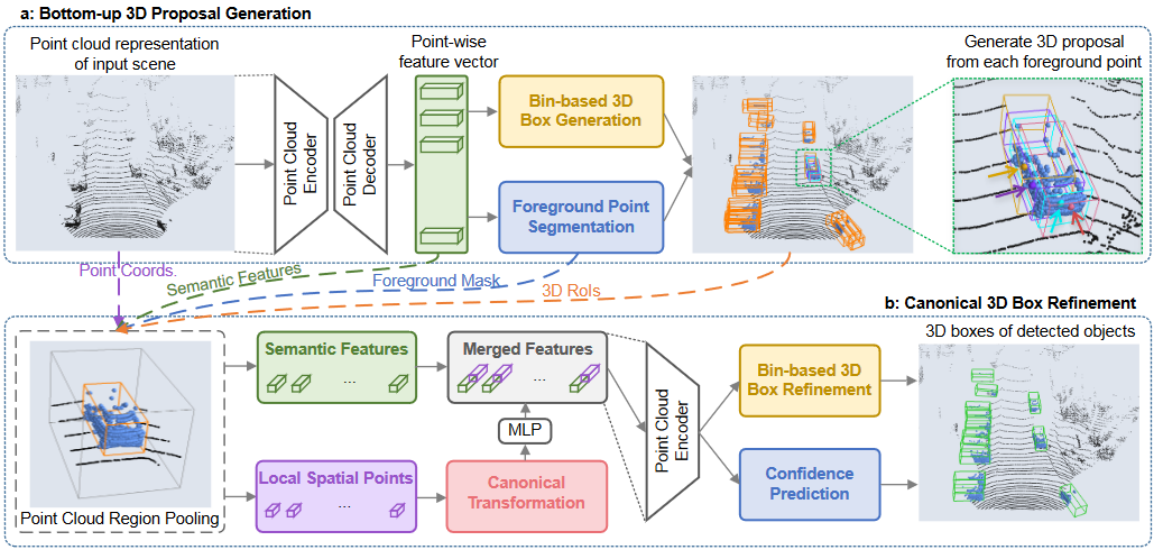


Figure 2.40: PointRCNN pipeline [61].

In contrast to previous approaches, that usually start by generating 3D proposals from image views, volumetric representations or bird's eye views, the PointRCNN architecture, published by Shaoshuai Shi et al [61], generates 3D bounding box proposals directly from the input point cloud.

PointRCNN Architecture

The PointRCNN framework is illustrated in Figure 2.40 and is composed of 2 stages:

1. **Bottom-up 3D proposal generation:**

   (a) **Learning point cloud representation:** to capture point-wise features from the input point cloud, the PointRCNN uses PointNet++, with MSG as the backbone network. As previously studied, the PointNet++ is able to capture features from local structures of the point cloud, and the segmentation step is used to obtain the point-wise features.

   (b) **Foreground point segmentation:** to each point-wise feature is added a segmentation head (a field for indicating if the point belongs to the foreground or to the background), which is evaluated by a ground truth segmentation mask created from ground truth 3D bounding boxes.

   (c) **Bin-based 3D bounding box generation:** one box regression head is also added to each point-wise feature for generating 3D proposals. A box regression head is composed of (x, y, z, h, w, l, θ) values, representing the object's center location in the LiDAR coordinate system, the bounding box dimensions and rotation around the Y-axis (i.e. from the bird's eye view). During training, only the 3D bounding box location values (of the box regression head) are regressed from the foreground points.

The estimation of the center location of an object on the X-Z plane is performed by associating a square area around each foreground point of the object and subdividing it into equally spaced bins, as illustrated in Figure 2.41. Each bin represents a different (x, z) location for a possible center of the object.

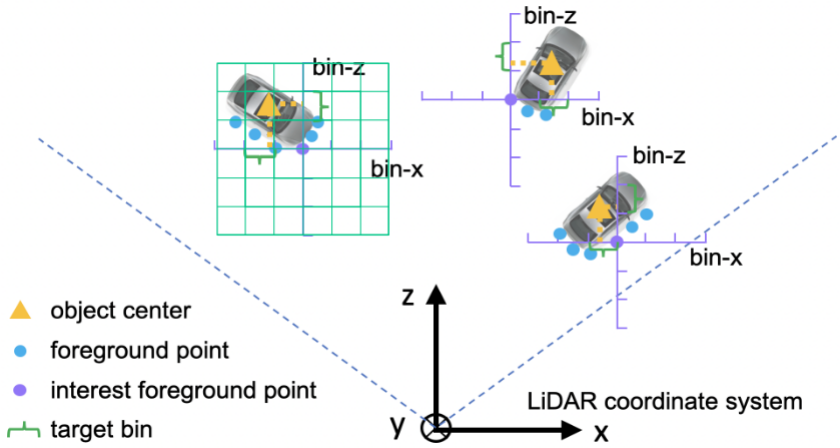


Figure 2.41: Illustration of bin-based localization, adapted from [61].

The bin-based approach is also used to predict the orientation of the object by dividing the 360 degrees into n bins and performing bin classification.

The object dimensions are directly regressed by calculating the residual with respect to the average object size of each class in the training set.

The selection of high-quality proposals is performed by eliminating redundant bounding boxes using Non-Maximum Suppression (NMS), based on the oriented IoU from the bird's eye view.

2. **Canonical 3D Box Refinement:**

   (a) **Point cloud region pooling:** The aim of this stage is to refine the bounding box location, orientation and foreground object confidence. To do so, each 3D box proposal is enlarged to create bigger bounding boxes that contain additional information about local context from the new enveloped points. All the points within the enlarged bounding box become associated to it and proceed for the refinement stage of this box.

   (b) **Canonical transformation:** The original coordinate system of all the points of the point cloud is the LiDAR coordinate system. To only estimate the residuals of the box parameters of the proposals, a transformation is applied to all the points of each proposal to transform them to the Canonical Coordinate System (CCS) of the correspondent proposal. In the CCS, the origin is at the center of the box proposal, the X and Z-axis are parallel to the ground plane, the X-axis is oriented towards the head direction of the proposal (front of the vehicle) and the Y-axis is the same as that of the LiDAR coordinate system. This is illustrated in Figure 2.42.

   By using the canonical coordinate system, the box refinement stage will be able to

learn better local spatial features for each proposal.

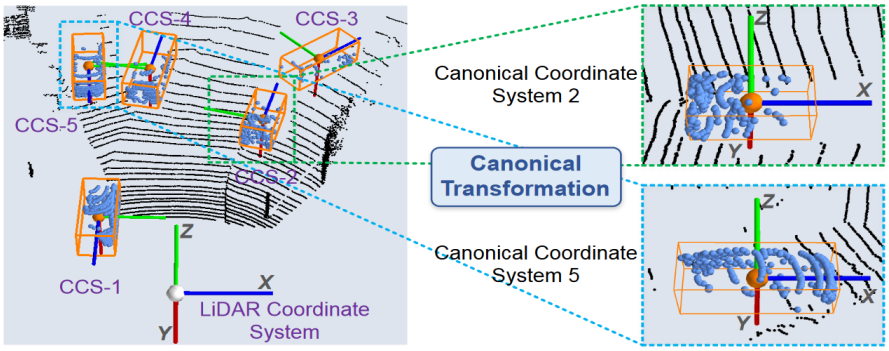


Figure 2.42: Illustration of canonical transformation [61].

(c) **Feature learning for box proposal refinement:** For the refinement of the bounding box proposals and confidence, a combination of the transformed local spatial points with their global semantic features is performed. However, before the feature concatenation is performed, three feature values are added to the coordinates of the transformed points:

- the distance of point to the LiDAR sensor in order to compensate for the loss of depth information after the transformation to the CCS.
- the correspondent reflection intensity and the segmentation mask.

The resulting concatenated features are supplied to a network with a similar structure to PointNet++, in order to obtain a feature vector for the following confidence classification and box refinement steps.

Performance Evaluation

In Figure 2.43, although most of the previous methods use both RGB images and point cloud as input, the PointRCNN method achieves better performance with an architecture that only uses point clouds as input. For the pedestrian detection, the PointRCNN has similar performance to other LiDAR-only methods but is surpassed by multimodal methods which can take advantage of the detail found in RGB images to better detect pedestrians.

Method	Modality	Car (IoU=0.7)			Pedestrian (IoU=0.5)			Cyclist (IoU=0.5)		
		Easy	Moderate	Hard	Easy	Moderate	Hard	Easy	Moderate	Hard
MV3D [4]	RGB + LiDAR	71.09	62.35	55.12	-	-	-	-	-	-
UberATG-ContFuse [17]	RGB + LiDAR	82.54	66.22	64.04	-	-	-	-	-	-
AVOD-FPN [14]	RGB + LiDAR	81.94	71.88	66.38	50.80	42.81	**40.88**	64.00	52.18	46.61
F-PointNet [25]	RGB + LiDAR	81.20	70.39	62.19	**51.21**	**44.89**	40.23	71.96	56.77	50.39
VoxelNet [43]	LiDAR	77.47	65.11	57.73	39.48	33.69	31.51	61.22	48.36	44.37
SECOND [40]	LiDAR	83.13	73.66	66.20	51.07	42.56	37.29	70.51	53.85	46.90
Ours	LiDAR	**85.94**	**75.76**	**68.32**	49.43	41.78	38.63	**73.93**	**59.60**	**53.59**

Figure 2.43: 3D object detection performance on KITTI test set using AP [61].

For 3D object detection of cars, the PointRCNN surpassed all the prior approaches at every difficulty level of the KITTI dataset, as shown in Figure 2.44.

Method	AP(IoU=0.7)		
	Easy	Moderate	Hard
MV3D [4]	71.29	62.68	56.56
VoxelNet [43]	81.98	65.46	62.85
SECOND [40]	87.43	76.48	69.10
AVOD-FPN [14]	84.41	74.44	68.65
F-PointNet [25]	83.76	70.92	63.65
Ours (no GT-AUG)	88.45	77.67	76.30
Ours	**88.88**	**78.63**	**77.38**

Figure 2.44: 3D object detection performance on the car class of KITTI validation set [61].

PointRCNN

PointRCNN is also an architecture that directly processes point clouds as input, not relying on the front view image or bird's eye view representations and achieves the best performance accross all car and cyclist detection difficulties.

However, the authors don't mention the inference rate of the network, and the performance scores are not high enough to be used in critical applications like in autonomous vehicles.

2.4.9 PointPillars

The PointPillars architecture, published by Alex H. Lang et al [35], is an encoder network that uses PointNets to learn a point cloud representation organized in vertical columns (pillars). The vertical column features are used for the prediction of oriented 3D bounding boxes and allow end-to-end learning by only relying on 2D convolutional layers. The proposed architecture was able to reach an inference rate of 62Hz, and a faster version reached the 105Hz mark.

PointPillars Architecture

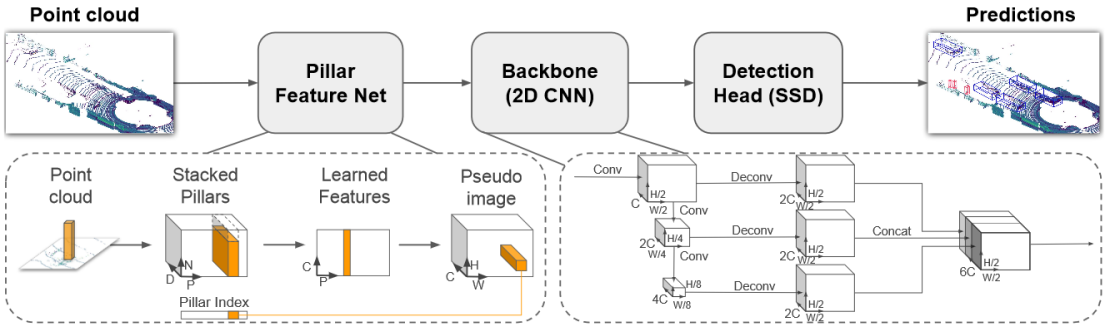


Figure 2.45: PointPillars pipeline [35].

The network uses point clouds as the input data and estimates oriented 3D bounding boxes around cars, cyclists and pedestrians. The architecture is subdivided into 3 stages:

1. **Pillar Feature Net:** It is a feature encoder network that converts a point cloud into a sparse pseudo-image.

   (a) The point cloud is discretized into an equally spaced grid on the XY plane (ground plane).

(b) The points in each pillar are augmented to 9-dimensional vectors (x, y, z, r, x_c, y_c, z_c, x_p, y_p):

- (x, y, z) are the position coordinates of the point in the point cloud.
- r is the reflectance.
- (x_c, y_c, z_c) is the distance to the arithmetic mean of all points in the pillar.
- (x_p, y_p) is the the offset from the pillar's (x,y) center.

The majority of the pillars will be empty due to the sparsity of the LiDAR point cloud, and the non-empty pillars will, in general, not contain a large number of points. The sparsity of the point cloud is handled by establishing limits to the number of non-empty pillars per point cloud sample (P) and the number of points per pillar (N), in order to create a dense tensor with size (D, P, N), where D is the number of dimensions of the points.

(c) A PointNet-based encoder is used to learn a set of features, from the stacked pillars, that can be scattered to create a 2D pseudo-image (with C channels) for the next stage 2D CNN.

2. **Backbone (2D CNN):** A 2D CNN processes the created pseudo-image and creates a feature map. This stage uses a similar backbone network as used in VoxelNet, and is composed of 2 sub-networks:
   - a top-down network to produce features with decreasing resolutions.
   - a network that performs upsampling and feature concatenation from the different resolutions.
3. **Detection Head (SSD):** This stage uses the Single Shot Detection (SSD) network to perform 3D bounding box regression over the feature map obtained from the 2D CNN.

Performance Analysis

Figures 2.46 and 2.47 show the mAP results of the BEV and 3D object detection, respectively, on the KITTI test set. PointPillars was able to outperform LiDAR-only methods in almost all categories and difficulties and achieve better performance on the cars and cyclists categories compared to the fusion-based methods.

Method	Modality	Speed (Hz)	mAP	Car			Pedestrian			Cyclist		
			Mod.	Easy	Mod.	Hard	Easy	Mod.	Hard	Easy	Mod.	Hard
MV3D [2]	Lidar & Img.	2.8	N/A	86.02	76.90	68.49	N/A	N/A	N/A	N/A	N/A	N/A
Cont-Fuse [15]	Lidar & Img.	16.7	N/A	88.81	85.83	77.33	N/A	N/A	N/A	N/A	N/A	N/A
Roarnet [25]	Lidar & Img.	10	N/A	88.20	79.41	70.02	N/A	N/A	N/A	N/A	N/A	N/A
AVOD-FPN [11]	Lidar & Img.	10	64.11	88.53	83.79	77.90	**58.75**	**51.05**	**47.54**	68.09	57.48	50.77
F-PointNet [21]	Lidar & Img.	5.9	65.39	88.70	84.00	75.33	58.09	50.22	47.20	75.38	61.96	54.68
HDNET [29]	Lidar & Map	20	N/A	89.14	**86.57**	78.32	N/A	N/A	N/A	N/A	N/A	N/A
PIXOR++ [29]	Lidar	35	N/A	**89.38**	83.70	77.97	N/A	N/A	N/A	N/A	N/A	N/A
VoxelNet [31]	Lidar	4.4	58.25	89.35	79.26	77.39	46.13	40.74	38.11	66.70	54.76	50.55
SECOND [28]	Lidar	20	60.56	88.07	79.37	77.95	55.10	46.27	44.76	73.67	56.04	48.78
PointPillars	Lidar	**62**	**66.19**	88.35	86.10	**79.83**	58.66	50.23	47.19	**79.14**	**62.25**	**56.00**

Figure 2.46: Results on the KITTI test BEV detection benchmark [35].

Method	Modality	Speed (Hz)	mAP	Car			Pedestrian			Cyclist		
			Mod.	Easy	Mod.	Hard	Easy	Mod.	Hard	Easy	Mod.	Hard
MV3D [2]	Lidar & Img.	2.8	N/A	71.09	62.35	55.12	N/A	N/A	N/A	N/A	N/A	N/A
Cont-Fuse [15]	Lidar & Img.	16.7	N/A	82.54	66.22	64.04	N/A	N/A	N/A	N/A	N/A	N/A
Roarnet [25]	Lidar & Img.	10	N/A	**83.71**	73.04	59.16	N/A	N/A	N/A	N/A	N/A	N/A
AVOD-FPN [11]	Lidar & Img.	10	55.62	81.94	71.88	66.38	50.80	42.81	40.88	64.00	52.18	46.61
F-PointNet [21]	Lidar & Img.	5.9	57.35	81.20	70.39	62.19	51.21	**44.89**	40.23	71.96	56.77	50.39
VoxelNet [31]	Lidar	4.4	49.05	77.47	65.11	57.73	39.48	33.69	31.5	61.22	48.36	44.37
SECOND [28]	Lidar	20	56.69	83.13	73.66	66.20	51.07	42.56	37.29	70.51	53.85	46.90
PointPillars	Lidar	**62**	**59.20**	79.05	**74.99**	**68.30**	**52.08**	43.53	**41.49**	**75.78**	**59.07**	**52.92**

Figure 2.47: Results on the KITTI test 3D detection benchmark [35].

To evaluate the orientation of the 3D bounding boxes, the authors used Average Orientation Similarity (AOS), which is a method that involves the projection of the 3D bounding boxes into the image, followed by 2D detection matching and the orientation assessment of the matched bounding boxes. The results are compared in Figure 2.48.

Considering that the accuracy of the projected bounding boxes can be affected by the orientation of their 3D boxes counterparts, PointPillars is generally outperformed by the image-only methods. However, in spite of this fact, it is able to achieve better performance on the cyclist category.

Method	Modality	Speed (Hz)	mAOS	Car			Pedestrian			Cyclist		
			Mod.	Easy	Mod.	Hard	Easy	Mod.	Hard	Easy	Mod.	Hard
SubCNN [27]	Img.	0.5	**72.71**	**90.61**	88.43	78.63	**78.33**	**66.28**	**61.37**	71.39	63.41	56.34
AVOD-FPN [11]	Lidar & Img.	10	63.19	89.95	87.13	79.74	53.36	44.92	43.77	67.61	57.53	54.16
SECOND [28]	Lidar	20	54.53	87.84	81.31	71.95	51.56	43.51	38.78	80.97	57.20	55.14
PointPillars	Lidar	**62**	68.86	90.19	**88.76**	**86.38**	58.05	49.66	47.88	**82.43**	**68.16**	**61.96**

Figure 2.48: Results of Average Orientation Similarity (AOS) [21] detection benchmark [35], on the KITTI test set.

Comments

PointPillars was able to considerably outperform the inference rate of Complex-YOLO, while achieving the highest scores in 3D object detection on the car and cyclist categories. It is another step forward towards high inference rate 3D object detection, which is not only needed to achieve high reliability, but also because the embedded systems tasked to run the Deep Learning architectures in autonomous vehicles will be much cheaper and slower than the GPUs used for training and testing the models.

As stated in the paper, even though the inference rate of 105 Hz is excessive compared to the 20 Hz limit imposed by the sampling rate of the LiDAR, it is important to consider that every evaluation made on the KITTI dataset only takes into consideration the points of the point cloud that are projected onto the front view image. In a real world scenario, the 3D object detection architectures would have to process and detect objects present in the entire point cloud, becoming important the use of highly optimized architectures that are capable of achieving inference rates, on the KITTI dataset, many times faster than the LiDAR's sampling rate.

2.4.10 Pseudo-LiDAR

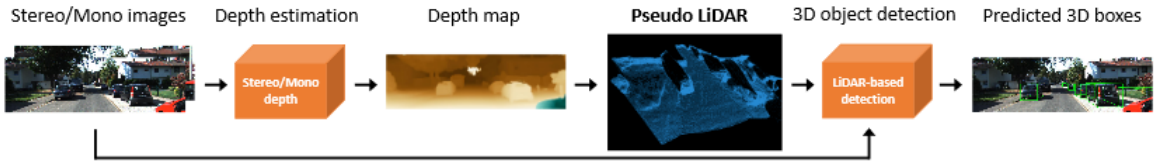


Figure 2.49: Pseudo-LiDAR 3D object detection pipeline [72].

The majority of the previously studied architectures perform 3D object detection on LiDAR point clouds. In contrast, the Pseudo-LiDAR approach, published by Yan Wang et al [72], focuses its attention to image-based 3D object detection on point clouds generated from monocular or stereo vision.

LiDAR-based approaches for 3D object detection have been dominating the field with ever increasing performance results, in comparison to image-based techniques. This is in part due to the stereo-based 3D depth estimation error growing quadratically with the distance, whereas the error of time-of-flight sensors like LiDAR grows linearly.

However, in recent years, the accuracy of the depth estimation techniques have greatly improved, with the Deep Ordinal Regression Network (DORN) [15] for monocular and Pyramid Stereo Matching Network (PSMNet) [8] for stereo depth estimation. And recent developments improved the efficiency of these algorithms, allowing for accurate disparity estimation in real-time.

Pseudo-LiDAR Architecture

To take advantage of these new depth estimation techniques, the authors propose a two-step approach by first estimating the dense pixel depth from stereo (or monocular) vision and back-projecting pixels into a 3D point cloud. By doing so, it is then possible to apply any LiDAR-based 3D object detection architectures to the data. This process is illustrated in Figure 2.49.

The experiments were performed using two multimodal 3D object detection architectures for combining an image view with the correspondent LiDAR point cloud: the AVOD [34] and Frustum PointNet architectures.

Performance Analysis

Figure 2.50 shows that the Pseudo-LiDAR approach has the best performance compared to other monocular and stereo-based approaches.

The stereo-based approaches are also able to obtain similar results, on the car category, as the 3D object detection results obtained from real LiDAR signals, but quickly degrades with the distance of the vehicles and for an IoU of 0.7. This is due to the decreasing accuracy of depth estimation for greater distances and to the low resolution of the camera used (0.4 MP).

Detection algorithm	Input signal	IoU = 0.5			IoU = 0.7		
		Easy	Moderate	Hard	Easy	Moderate	Hard
MONO3D [4]	Mono	30.5 / 25.2	22.4 / 18.2	19.2 / 15.5	5.2 / 2.5	5.2 / 2.3	4.1 / 2.3
MLF-MONO [33]	Mono	55.0 / 47.9	36.7 / 29.5	31.3 / 26.4	22.0 / 10.5	13.6 / 5.7	11.6 / 5.4
AVOD	Mono	61.2 / 57.0	45.4 / 42.8	38.3 / 36.3	33.7 / 19.5	24.6 / 17.2	20.1 / 16.2
F-POINTNET	Mono	70.8 / 66.3	49.4 / 42.3	42.7 / 38.5	40.6 / 28.2	26.3 / 18.5	22.9 / 16.4
3DOP [5]	Stereo	55.0 / 46.0	41.3 / 34.6	34.6 / 30.1	12.6 / 6.6	9.5 / 5.1	7.6 / 4.1
MLF-STEREO [33]	Stereo	-	53.7 / 47.4	-	-	19.5 / 9.8	-
AVOD	Stereo	89.0 / 88.5	77.5 / 76.4	68.7 / 61.2	74.9 / 61.9	56.8 / 45.3	49.0 / 39.0
F-POINTNET	Stereo	89.8 / 89.5	77.6 / 75.5	68.2 / 66.3	72.8 / 59.4	51.8 / 39.8	44.0 / 33.5
AVOD [17]	LiDAR + Mono	90.5 / 90.5	89.4 / 89.2	88.5 / 88.2	89.4 / 82.8	86.5 / 73.5	79.3 / 67.1
F-POINTNET [25]	LiDAR + Mono	96.2 / 96.1	89.7 / 89.3	86.8 / 86.2	88.1 / 82.6	82.2 / 68.8	74.0 / 62.0

Figure 2.50: Pseudo-LiDAR car 3D object detection performance on the KITTI validation set, reported as AP_{BEV} / AP_{3D} (%), IoU = 0.5. **Mono** stands for monocular. The Pseudo-LiDAR estimated by PSMNET (stereo) or DORN (monocular) are in blue. Methods with LiDAR are in gray [72].

The Pseudo-LiDAR pipeline was also tested for 3D pedestrian and cyclist detection, as shown in Figure 2.51, and resulted in a significant performance difference compared to the models using real LiDAR signals. This is due to the increased difficulty of 3D object detection on smaller objects in conjunction to the stereo-based technique.

Input signal	Easy	Moderate	Hard
Pedestrian			
Stereo	41.3 / 33.8	34.9 / 27.4	30.1 / 24.0
LiDAR + Mono	69.7 / 64.7	60.6 / 56.5	53.4 / 49.9
Cyclist			
Stereo	47.6 / 41.3	29.9 / 25.2	27.0 / 24.9
LiDAR + Mono	70.3 / 66.6	55.0 / 50.9	52.0 / 46.6

Figure 2.51: Pseudo-LiDAR pedestrian and cyclist 3D object detection performance on the KITTI validation set, IoU = 0.5, reported as AP_{BEV} / AP_{3D} (%) [72].

Considering the results of the test set on the car category, shown in Figure 2.52, we can conclude that the performance of the architecture is significantly inferior when compared to the LiDAR-based pointclouds.

Input signal	Easy	Moderate	Hard
AVOD			
Stereo	66.8 / 55.4	47.2 / 37.2	40.3 / 31.4
†LiDAR + Mono	88.5 / 81.9	83.8 / 71.9	77.9 / 66.4
F-POINTNET			
Stereo	55.0 / 39.7	38.7 / 26.7	32.9 / 22.3
†LiDAR + Mono	88.7 / 81.2	84.0 / 70.4	75.3 / 62.2

Figure 2.52: Pseudo-LiDAR car 3D object detection performance on the KITTI test set, reported as AP_{BEV} / AP_{3D} (%), with IoU = 0.7 [72].

2.4.11 Pseudo-LiDAR++

The performance of Pseudo-LiDAR in 3D object detection is heavily affected by the quality of the depth estimation approaches. Recent algorithms that leverage the power of Deep Learning techniques, like the aforementioned PSMNet, are able to achieve better stereo disparity

estimation, allowing for more accurate 3D point clouds.

As shown in Figures 2.50 and 2.51, the 3D object detection performance of Pseudo-LiDAR decreases significantly for objects at greater distances. This problem arises from the fact that stereo depth networks (SDNs) are designed to learn how to minimize the disparity error. Because the pixel disparity is more pronounced at closer distances, the algorithm can give more relevance to nearby objects' disparity errors than to faraway objects'. This is illustrated in Figure 2.53.

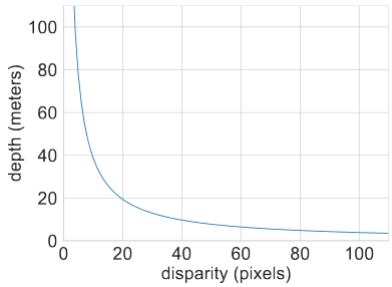


Figure 2.53: The disparity-to-depth transform using the KITTI dataset [75].

And, as mentioned in the paper published [75] by Yurong You et al , a single pixel disparity error implies only a 0.1m depth error at a 5m distance, but a 5.8m depth error at a 50m distance.

Pseudo-LiDAR++ Architecture

To counter this discrepancy between the impact of the errors at different distances from the stereo camera, the authors propose a pipeline for accurate depth learning, rather than for disparity learning.

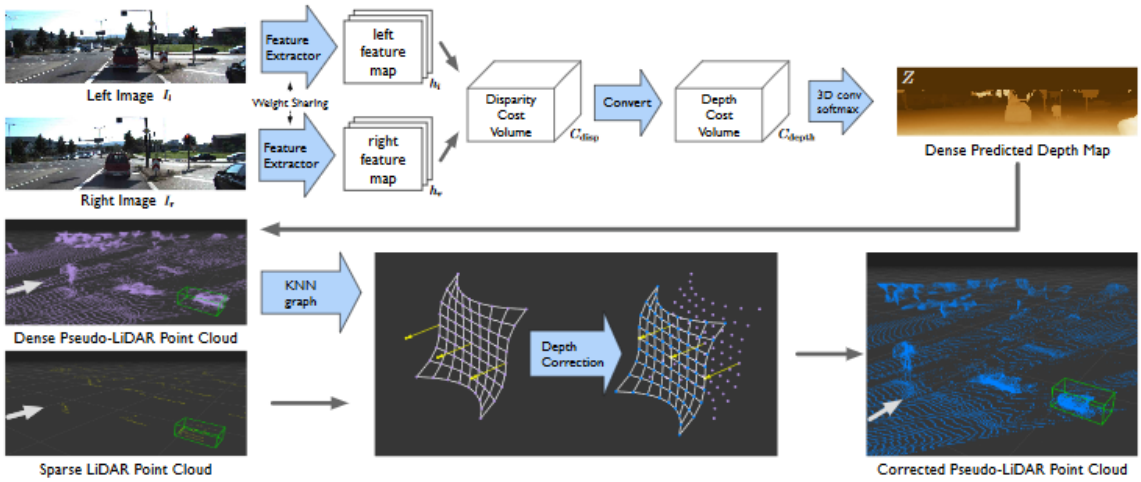


Figure 2.54: Pseudo-LiDAR++ pipeline [75].

However, even after the significant improvement on depth estimation, there is a limitation in stereo disparity estimation imposed by the discrete nature of the pixels. Being the disparity the difference between horizontal coordinates of corresponding pixels, the result is a discretized

value dependent on the image resolution, while the depth is continuous. As higher resolution images require higher computational requirements, the pipeline uses a hybrid approach, named Graph-based Depth Correction (GDC), to combine an inexpensive 4-line LiDAR sensor, that produces highly sparse point clouds, with the generated dense stereo point cloud to correct the bias imposed by the pixel discretization. The entire pipeline is shown in the Figure 2.54.

Performance Analysis

All the entries in blue are from the Pseudo-LiDAR++ with enhanced depth estimation by the SDN and GDC. The results of 3D object detection in both Figures 2.55 and 2.56 show that:

- Pseudo-LiDAR++ is able to improve upon the previous iteration in every category and difficulty.
- The depth correction step, performed by the GDC algorithm, significantly enhances the object detection performance compared to only using pseudo-lidar point clouds.

		IoU = 0.5			IoU = 0.7		
Detection algorithm	Input	Easy	Moderate	Hard	Easy	Moderate	Hard
3DOP [4]	S	55.0 / 46.0	41.3 / 34.6	34.6 / 30.1	12.6 / 6.6	9.5 / 5.1	7.6 / 4.1
MLF-STEREO [42]	S	-	53.7 / 47.4	-	-	19.5 / 9.8	-
S-RCNN [21]	S	87.1 / 85.8	74.1 / 66.3	58.9 / 57.2	68.5 / 54.1	48.3 / 36.7	41.5 / 31.1
PL: AVOD [36]	S	89.0 / 88.5	77.5 / 76.4	68.7 / 61.2	74.9 / 61.9	56.8 / 45.3	49.0 / 39.0
PL: PIXOR*	S	89.0 / -	75.2 / -	67.3 / -	73.9 / -	54.0 / -	46.9 / -
PL: P-RCNN	S	88.4 / 88.0	76.6 / 73.7	69.0 / 67.8	73.4 / 62.3	56.0 / 44.9	52.7 / 41.6
PL++: AVOD	S	89.4 / 89.0	79.0 / 77.8	70.1 / 69.1	77.0 / 63.2	63.7 / 46.8	56.0 / 39.8
PL++: PIXOR*	S	89.9 / -	78.4 / -	74.7 / -	79.7 / -	61.1 / -	54.5 / -
PL++: P-RCNN	S	89.8 / 89.7	83.8 / 78.6	77.5 / 75.1	82.0 / 67.9	64.0 / 50.1	57.3 / 45.3
PL++: AVOD	L# + S	90.2 / 90.1	87.7 / 86.9	79.8 / 79.2	86.8 / 70.7	76.6 / 56.2	68.7 / 53.4
PL++: PIXOR*	L# + S	95.1 / -	85.1 / -	78.3 / -	84.0 / -	71.0 / -	65.2 / -
PL++: P-RCNN	L# + S	90.3 / 90.3	87.7 / 86.9	84.6 / 84.2	88.2 / 75.1	76.9 / 63.8	73.4 / 57.4
AVOD [16]	L + M	90.5 / 90.5	89.4 / 89.2	88.5 / 88.2	89.4 / 82.8	86.5 / 73.5	79.3 / 67.1
PIXOR* [47, 22]	L + M	94.2 / -	86.7 / -	86.1 / -	85.2 / -	81.2 / -	76.1 / -
P-RCNN [35]	L	96.3 / 96.1	88.6 / 88.5	88.6 / 88.5	87.8 / 81.7	86.0 / 74.4	85.8 / 74.5

Figure 2.55: Pseudo-LiDAR++ object detection results on KITTI validation set of the car category, reported as AP_{BEV} / AP_{3D} (%). The input of the different architectures are represented by the following symbols: M: monocular images, S: stereo images, $L\#$: sparse 4-beam LiDAR, L: 64-beam LiDAR and S: stereo images [75].

Stereo depth	Easy	Moderate	Hard
PSMNET	41.3 / 33.8	34.9 / 27.4	30.1 / 24.0
SDN	48.7 / 40.9	40.4 / 32.9	34.9 / 28.8
SDN + GDC	**63.7 / 53.6**	**53.8 / 44.4**	**46.8 / 38.1**
PSMNET	47.6 / 41.3	29.9 / 25.2	27.0 / 24.9
SDN	49.3 / 44.6	30.4 / 28.7	28.6 / 26.4
SDN + GDC	**65.7 / 60.8**	**45.8 / 40.8**	**42.8 / 38.0**

Figure 2.56: Pseudo-LiDAR++ object detection results of pedestrians (top) and cyclists (bottom) on the KITTI validation set, reported as AP_{BEV} / AP_{3D} (%), at IoU of 0.5 [75].

The performance of the architecture on 3D object detection of the car category, on the test set is shown in Figure 2.57.

Input signal	Easy	Moderate	Hard
PL++ (SDN)	75.5 / 60.4	57.2 / 44.6	53.4 / 38.5
PL++ (SDN + GDC)	83.8 / 68.5	73.5 / 54.7	66.5 / 51.2
LiDAR	89.5 / 85.9	85.7 / 75.8	79.1 / 68.3

Figure 2.57: Pseudo-LiDAR++ object detection results of the car category, on the KITTI test set. Comparison between the models PL++ (blue) and 64-beam LiDAR (gray), using P-RCNN, and report AP_{BEV} / AP_{3D} (%), IoU = 0.7 [75].

Comments

Whereas Pseudo-LiDAR tries to detect objects solely from monocular or stereo vision, resulting in disparity estimation limitations due to the error increasing quadratically with the distance and to pixel discretization, the Pseudo-Lidar++ makes use of a low resolution LiDAR to implement a hybrid approach to achieve better results.

Comparing the Figures 2.52 and 2.57, it is possible to conclude that the performed point cloud correction resulted in significantly more precise Pseudo-LiDAR point clouds, as the results approach the ones from LiDAR generated point clouds.

In contrast to the original Pseudo-LiDAR, by relying on a LiDAR sensor, the inference rate of the Pseudo-LiDAR++ approach becomes limited by the sampling rate of the LiDAR and no redundancy is achieved in case of camera failure, obstruction or poor visibility conditions.

2.4.12 Point Voxel-RCNN (PV-RCNN)

In general, 3D object detection approaches can be classified as:

- **grid-based methods**: that transform the irregular point cloud representation into a regular structure using 3D voxels or 2D bird's eye views. These data structures can then be efficiently processed, but suffer from information loss due to the discretization of the data.

- **point-based methods**: that directly capture point features from raw point clouds. They can achieve larger receptive fields through the point set abstraction, capable of encoding point-features from a neighbourhood of any size, but are computationally expensive.

The published PV-RCNN framework, by Shaoshuai Shi et al, [60] incorporates both grid and point-based methods to improve feature learning in order to increase the 3D object detection performance.

PV-RCNN Architecture

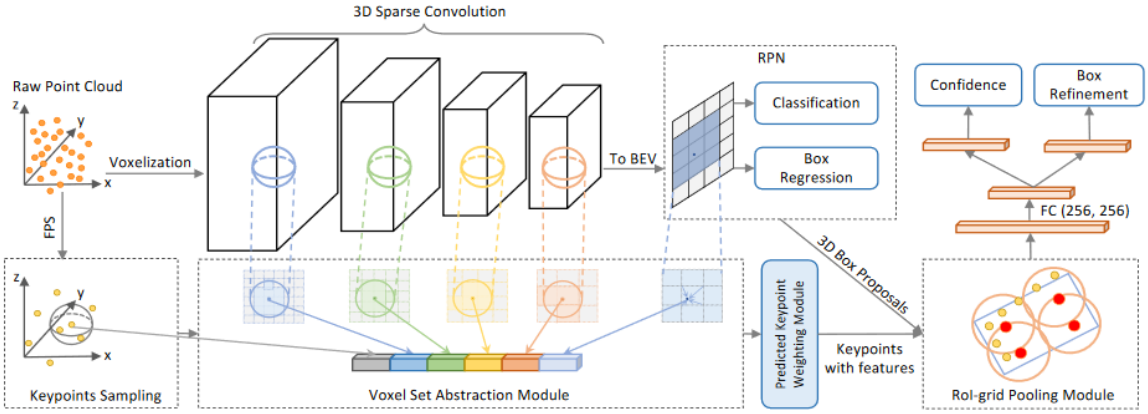


Figure 2.58: PV-RCNN architecture [60].

The steps performed by the PV-RCNN, illustrated in Figure 2.58, are as follow:

1. **Voxelization**: The PV-RCNN pipeline starts by voxelizing the input point cloud into a 3D grid, where the features of the non-empty voxels are calculated by averaging the point-wise features of the points within each voxel. The point-wise features are usually the 3D coordinates and the reflectance intensities.

2. **3D sparse convolutions**: The resulting volumetric representation is passed through four 3D sparse convolution layers, generating feature volumes with 1x, 2x, 4x and 8x downsampled sizes.

3. **RPN**: the 8x downsampled feature volume is converted into multiple 2D bird's eye view feature maps and are used to generate 3D proposals following the anchor-based approach.

4. **Keypoints Sampling**: the Farthest Point Sampling (FPS) algorithm is used to sample uniformly distributed points for representing the entire scene.

5. **Voxel Set Abstraction Module**: this module encodes the previously generated multi-scale features to the sampled keypoints. The "Set Abstraction" layer proposed by PointNet++ is used to aggregate the voxel-wise feature volumes.

6. **Predicted Keypoint Weighting module**: in order to better refine the box proposals, it is important to give priority to the keypoints belonging to foreground objects. As such, this module re-weighs the keypoint features during training, using the point cloud segmentation created from ground truth 3D boxes. So the entire scene is being summarized into the set of keypoints with multi-scale semantic features.

7. **RoI-grid Pooling via the Set Abstraction layer**: this layer aggregates the keypoint features into RoI grids for the box proposal refinement step.

8. **3D Proposal Refinement and Confidence Prediction**: the proposal refinement network uses the RoI features of each box proposal to learn how to predict the size and location residuals. It uses a 2-layer MLP adn two branches for the confidence prediction and box refinement.

Performance Analysis

The PV-RCNN pipeline outperforms the state of the art approaches to 3D object detection of cars. It achieves better performance in the easy and moderate difficulties of bird's eye view detection, with a slight mAP decrease in the hard difficulty, as shown in Figure 2.59.

In relation to the cyclist 3D and bird's eye view detection, the mAP surpasses the results from previous approaches in the moderate and hard difficulties, by approximately 2%.

Method	Reference	Modality	Car - 3D Detection			Car - BEV Detection			Cyclist - 3D Detection			Cyclist - BEV Detection		
			Easy	Mod.	Hard	Easy	Mod.	Hard	Easy	Mod.	Hard	Easy	Mod.	Hard
MV3D [1]	CVPR 2017	RGB + LiDAR	74.97	63.63	54.00	86.62	78.93	69.80	-	-	-	-	-	-
ContFuse [17]	ECCV 2018	RGB + LiDAR	83.68	68.78	61.67	94.07	85.35	75.88	-	-	-	-	-	-
AVOD-FPN [11]	IROS 2018	RGB + LiDAR	83.07	71.76	65.73	90.99	84.82	79.62	63.76	50.55	44.93	69.39	57.12	51.09
F-PointNet [22]	CVPR 2018	RGB + LiDAR	82.19	69.79	60.59	91.17	84.67	74.77	72.27	56.12	49.01	77.26	61.37	53.78
UberATG-MMF [16]	CVPR 2019	RGB + LiDAR	88.40	77.43	70.22	93.67	88.21	81.99	-	-	-	-	-	-
SECOND [34]	Sensors 2018	LiDAR only	83.34	72.55	65.82	89.39	83.77	78.59	71.33	52.08	45.83	76.50	56.05	49.45
PointPillars [12]	CVPR 2019	LiDAR only	82.58	74.31	68.99	90.07	86.56	82.81	77.10	58.65	51.92	79.90	62.73	55.58
PointRCNN [25]	CVPR 2019	LiDAR only	86.96	75.64	70.70	92.13	87.39	82.72	74.96	58.82	52.53	**82.56**	67.24	60.28
3D IoU Loss [39]	3DV 2019	LiDAR only	86.16	76.50	71.39	91.36	86.22	81.20	-	-	-	-	-	-
Fast Point R-CNN [2]	ICCV 2019	LiDAR only	85.29	77.40	70.24	90.87	87.84	80.52	-	-	-	-	-	-
STD [37]	ICCV 2019	LiDAR only	87.95	79.71	75.09	94.74	89.19	**86.42**	**78.69**	61.59	55.30	81.36	67.23	59.35
Patches [13]	Arxiv 2019	LiDAR only	88.67	77.20	71.82	92.72	88.39	83.19	-	-	-	-	-	-
Part-A^2 [26]	Arxiv 2019	LiDAR only	87.81	78.49	73.51	91.70	87.79	84.61	-	-	-	-	-	-
PV-RCNN (Ours)	-	LiDAR only	**90.25**	**81.43**	**76.82**	**94.98**	**90.65**	86.14	78.60	**63.71**	**57.65**	82.49	**68.89**	**62.41**
Improvement	-	-	*+1.58*	*+1.72*	*+1.73*	*+0.24*	*+1.46*	*-0.28*	*-0.06*	*+2.12*	*+2.35*	*-0.07*	*+1.65*	*+2.13*

Figure 2.59: Performance comparison on the KITTI test set, using mAP with 40 recall positions [60].

Comments

PV-RCNN was able to outperform the state of the art in several object categories and difficulties, with a score surpassing the 90% score in some of the difficulties associated with the car 3D and bird's eye view detection. However, the paper does not mention the inference time, which is needed in order to better compare it with Complex-YOLO and PointPillars, which are focused on this goal.

2.4.13 Summary

3D object identification is a relatively new field of study that has been explored, in the last few years, with a variety of methods, from architectures interpreting point clouds as discretized voxel grids and images, to directly extracting features from the raw point clouds, many in the autonomous driving context. The field has been progressively evolving, with the PV-RCNN architecture, the latest reviewed in this thesis, being capable of reaching the 90% mark in several categories and difficulties of the KITTI dataset.

We studied the PointNet and PointNet++ architectures, that are the foundation of many subsequent approaches to 3D object identification. Their usage in current state of the art algorithms is motivated by the fact that these architectures directly extract features from the point cloud representation, without pre-processing steps that would discretize the input point cloud and remove potentially important features.

There are also architectures that use a combination of the camera and LiDAR sensors for different purposes: Frustum PointNet uses 2D object detection on the front view image in order to increase the speed and accuracy of object proposal generation, and Pseudo-LiDAR++ for generating a more accurate point cloud from the front-view stereo or monocular images.

However, these approaches are dependent on the camera sensors, which makes them incapable to be applied as a redundancy measure in case of camera failure, obstruction or poor visibility conditions.

It should be noted that all proposed architectures are purely feedforward, which means that no knowledge of the past (frames) is used to their advantage. This is certainly not the case for Human driving.

On the other hand, the performance of 3D object identification architectures has yet to reach a high enough accuracy to be deployed in consumer-level autonomous vehicles, as the models still perform rather poorly on the more difficult levels of the KITTI dataset.

In addition, in more recent architectures, the limitations of the KITTI dataset became more apparent. First, the usage of an unbalanced evaluation metric, the 11-point interpolation average precision metric, until properly addressed in 2019, and the incompleteness of the KITTI dataset annotations, which only consider the points of the point cloud that are projected onto the front view image, do not provide a solid ground for evaluating datasets for real-world scenarios. For example, even though the Complex-YOLO and PointPillars architectures are capable of achieving an inference rate of 50fps and 105fps, respectively, in the KITTI dataset, the same results are not guaranteed when taking into account 3D object detection on the entire point cloud, which is the real world scenario in autonomous driving. As mentioned by the authors of the PointPillars architecture [35], the portion of the point cloud considered by KITTI is only about 10% of its total size.

It should also be noted that the performance of an architecture, that is measured with metrics such as average precision, should not be the only aspect to consider when evaluating a model destined to perform critical tasks such as autonomous driving. Other network aspects like explainability [40] and robustness [1], should also be evaluated. For example, as an attempt to better understand what the network has learnt, the exact sequence of steps used by the network to arrive to a certain decision and what situations it is able to effectively address or not.

The quantification of the trustworthiness aspect is to be highlighted for determining if a model is ready to be deployed in autonomous vehicles. This area of interest is very recent and tries to answer the question of how much can a deep neural network be trusted. Recent publications by Alexander Wong et al [73] and Andrew Hryniowski et al [27] explore this field of study by proposing a set of metrics to quantify the trustworthiness of a model.

2.5 KITTI Objection Detection Dataset in Detail

The KITTI object detection dataset [19], published by Andreas Geiger et al, was the first publicly available annotated dataset focused on autonomous vehicles. Due to its availability early on, it has become the standard dataset for benchmarking the 3D object identification architectures, as to easily compare new approaches to the previous state of the art performance results.

This section provides a detailed review of this dataset due to its selection as the targeted format for the output of the developed dataset generation software, described in Chapter 3. The importance of reproducing the system setup originally used by KITTI and generating datasets with the appropriately formatted data is due to how sensitive trained models are to the original dataset used for training. A single departure from the original system setup or the values, range and format comprising the dataset information, may lead to a decrease in performance of the model. This is also important when considering the use of transfer learning for training the models with the newly generated datasets.

2.5.1 Data acquisition sensors

The sensor setup used to acquire the KITTI dataset is illustrated in Figure 2.60.

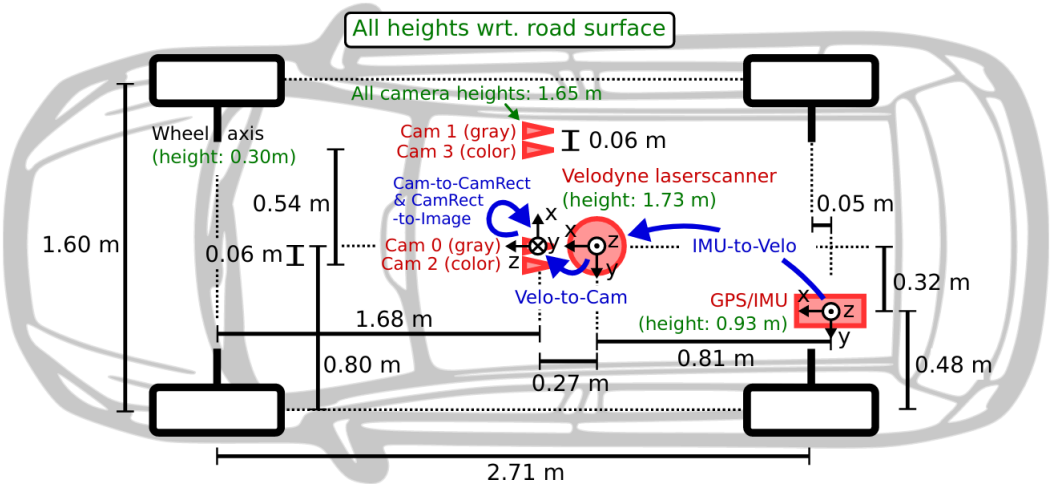


Figure 2.60: Vehicle sensor setup [19].

In summary, the system is comprised of 4 front view 1.4 MP cameras, 2 greyscale cameras and the 2 color cameras, a Velodyne HDL-64E with 0.09 degrees angular resolution, vertical FOV of 26.8 degrees, 120 m range and sampling rate of 10 Hz, and a GPS/IMU unit.

The 4 cameras are identified by numbers, as follows:

- Cam 0: left greyscale camera. It is the reference camera, i.e. it is the chosen camera for the projection of a point in the Velodyne coordinate system onto an image. This point is then transformed from Cam0 the coordinate systems of the other cameras.
- Cam 1: right greyscale camera
- Cam 2: left color camera

- Cam 3: right color camera

For the purpose of object identification, the majority of the studied Deep Learning architectures only use data from the LiDAR sensor and the left color camera (Cam 2), the exception being the Pseudo-LiDAR versions, which use both color cameras as sources for stereo disparity estimation.

The coordinate systems of the three types of sensors are presented in Table 2.2 and illustrated in Figure 2.61. Every coordinate system is right-handed.

Coordinate System	x	y	z
Camera	right	down	forward
Velodyne	forward	left	up
GPS/IMU	forward	left	up

Table 2.2: Sensors coordinate systems comparison

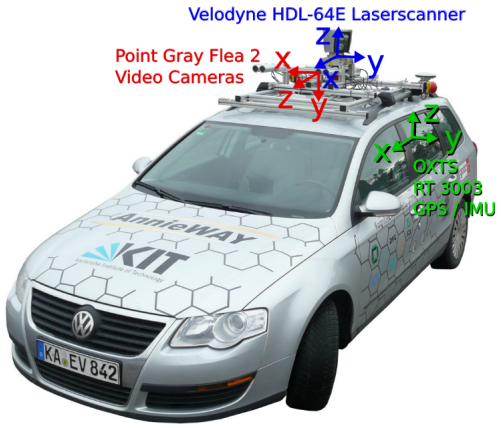
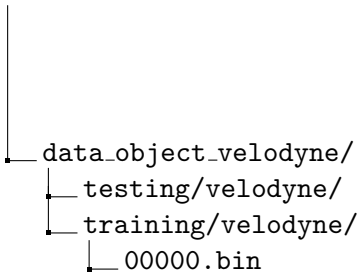


Figure 2.61: Sensors coordinate systems [19].

2.5.2 KITTI dataset file structure

The KITTI object detection dataset, downloaded from [20], uses 0-based indexing to identify all the files corresponding to a given sample, and has the following structure:

```
Dataset/
├── data_object_calib/
│   ├── testing/calib/
│   └── training/calib/
│       └── 00000.txt
├── data_object_label_2/
│   └── training/label_2/
│       └── 00000.txt
└── data_object_image_2/
    ├── testing/image_2/
    └── training/image_2/
        └── 00000.png
```



2.5.3 Evaluation on KITTI Object Detection Dataset

KITTI supplies two portions of the dataset: one for training and validation, which is comprised of 7481 samples, and one for testing, constituting 7518 samples. The labels of the testing dataset are not supplied as it is only used by KITTI to perform online evaluation of the submitted models. The standard for training and validating a model using the training dataset, is to perform a 50/50 dataset split. It is also crucial the split must avoid sharing samples between the training and validation samples that are from the same sample sequence.

2.5.4 Sample data analyses

Camera data: every image in the KITTI dataset is given without any compression loss and in the PNG format. The original images are subjected to two post-processing steps, where they are cropped at the top and bottom to remove the sky and car hood, resulting in images with 1382×512 pixel resolution, and are then rectified in order to remove the distortions applied by the lenses. At the end, the images present in the KITTI dataset have a resolution of 1224×470 pixels.

Label data: The label file contains all the information about the detectable objects in a scene, where each line corresponds to one. It is comprised of 15 properties described in Table 2.3:

# Values	Name	Description
1	type	Describes the type of object: 'Car', 'Van', 'Truck', 'Pedestrian', 'Person_sitting', 'Cyclist', 'Tram', 'Misc' or 'DontCare'
1	truncated	Float from 0 (non-truncated) to 1 (truncated), where truncated refers to the object leaving image boundaries
1	occluded	Integer (0,1,2,3) indicating occlusion state: 0 = fully visible, 1 = partly occluded, 2 = largely occluded, 3 = unknown
1	alpha	Observation angle of object, ranging [-pi..pi]
4	bbox	2D bounding box of object in the image: contains minX, minY, maxX, maxY pixel coordinates
3	dimensions	3D object dimensions: height, width, length (in meters)
3	location	3D object location x,y,z in the camera coordinate system (in meters). It is located at the center of the base of the object.
1	rotation_y	Rotation ry around Y-axis in camera coordinates [-pi..pi]
1	score	Only for results: Float, indicating confidence in detection, needed for p/r curves, higher is better.

Table 2.3: Values for each object present in the KITTI label file [47].

Not every architecture requires a dataset that provides all the properties presented in the Table 2.3, and default values should be used when encountering a 'DontCare' entry. The standard default values are listed in Table 2.4.

Name	Default Value
type	'DontCare'
truncated	-1
occluded	-1
alpha	-10
bbox	The only mandatory values
dimensions	(-1, -1, -1)
location	(-1000, -1000, -1000)
rotation_y	-10
score	Skipped when generating or labeling datasets

Table 2.4: KITTI label properties default values.

The 'DontCare' type identifies regions with non-labeled objects, and instructs the Deep Learning architectures to ignore them.

LiDAR data: each Velodyne pointcloud is stored in a binary file, as a float matrix. Each line corresponds to a point, represented by its 3D coordinates, relative to the LiDAR origin, and reflectance (which is the intensity at which the reflected light beam is received). The used Velodyne has a stack of 64 lasers, capturing at each rotation step around its z axis a maximum of 64 vertically aligned points.

Calibration data: A calibration file contains the projection matrices, in row-major order, used to convert from one coordinate system to another. This information is essential because the each sensor has its own coordinate system. This file has the following information:

- **P0, P1, P2, P3**: 3×4 projection matrices, used to project a point from the reference rectified coordinate system (Cam0) onto a rectified image of Cam0 (P0 matrix), Cam1 (P1 matrix), Cam2 (P2 matrix) and Cam3 (P3 matrix).
- **R0_rect**: 3×3 rectifying rotation matrix of the reference camera (Cam0) to make image planes coplanar. It is the transformation matrix used to rectify a non-rectified image of Cam0.
- **Tr_velo_to_cam**: 3×4 transformation matrix that maps a point in the LiDAR coordinate system to a point in a non-rectified image of Cam0.
- **Tr_imu_to_velo**: 3×4 transformation matrix that transforms a point in the IMU coordinate system onto a point in the LiDAR coordinate system.

Chapter 3

Dataset Generation for 3D Object Identification

The previous chapters highlight the predominant use of the KITTI object detection dataset for benchmarking the 3D object identification architectures over the years. It is a way to compare different models proposed to the same end. For this reason, we developed a software tool to generate samples according to the KITTI dataset format.

Currently, there are mainly two free alternatives that provide virtual environments required to generate fully synthetic datasets for autonomous vehicles, which are the CARLA open source simulator [13] and the GTA V game. At the moment of writing this thesis, GTA V is the most attractive option due to greater map variety, dynamic character behaviours, population diversity and a more dynamic environment due to its physics engine. As such, GTA V was selected as the target environment for the implementation of synthetic dataset generation. This project is possible through the use of ScriptHookV library and SDK [4], developed by Alexander Blade, that enables the implementation of mods in GTA V.

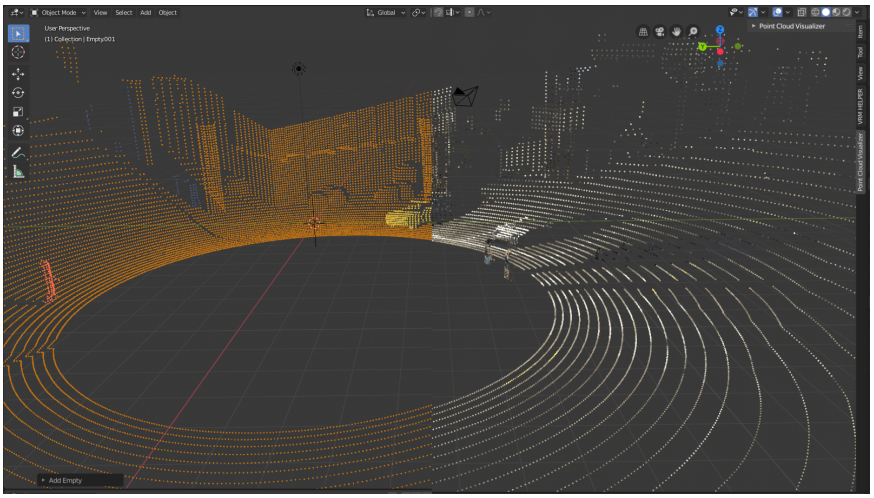


Figure 3.1: Comparison between colorless point cloud, with added color for visualization of the mod's object classes, and the colored point cloud.

This project is based on a previous work, developed in the 3rd year course of Informatics Engineering Project [7], which aimed at capturing and generating colored point clouds using the color of the pixels associated with the projected points of the point cloud onto the image views, as illustrated in Figure 3.1, with the click of a button.

3.1 GTA V Mod Project

The goal of the developed GTA V mod is to enable and facilitate the generation of annotated datasets, compatible with Deep Learning architectures in the field of autonomous driving.

The generation of a KITTI formatted dataset based on GTA V requires a two-step process, as to not overload the dataset generation in GTA V and to take advantage of Python libraries for the required calculations to obtain the data for the KITTI formatted dataset.

The data generated from the GTA V mod, for each dataset sample, enables several options for the type of dataset that it is possible to generate. For example, it is possible to generate KITTI formatted datasets, datasets comprised of point cloud samples with the background points removed, and so on. To enable this ability to produce different types of datasets, the GTA V mod outputs the following files for each sample:

- `LiDAR_PointCloud_Camera_Print_Day_0.bmp`: each sample has associated with it 3 images taken at 0^{o} (front view of the vehicle), 120^{o} and 240^{o}, for the day, night and cloudy weather, illustrated by the Figure 3.2.

Figure 3.2: The three weather conditions captured for each sample.

- `LiDAR_PointCloud.ply`: file containing the points position (x, y, z) of the sampled point cloud, without noise. It is denoted as the "ideal point cloud".
- `LiDAR_PointCloud_error.ply`: file containing the points position in euclidean space (x, y, z) of the sampled pointcloud, with noise applied to each point. It is represented in Figure 3.3 as the point cloud with noise.

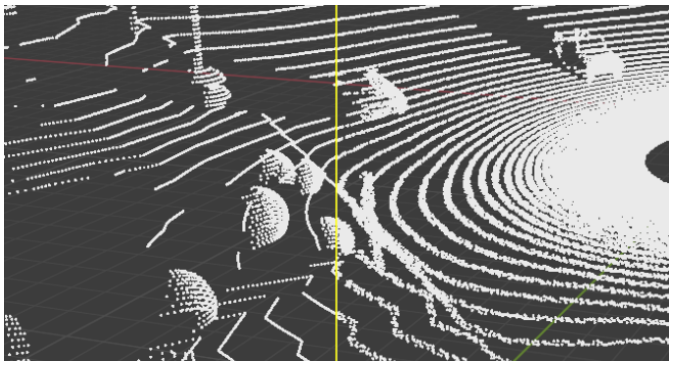


Figure 3.3: Comparison between the ideal point cloud and the point cloud with noise.

- `LiDAR_PointCloud_labels.txt`: file containg a label for each point of the point cloud. The detectable classes are background (0), humans and animals (1), vehicles (2) and game props (3). Figure 3.4 illustrates these classes by coloring the points belonging to objects of each class.

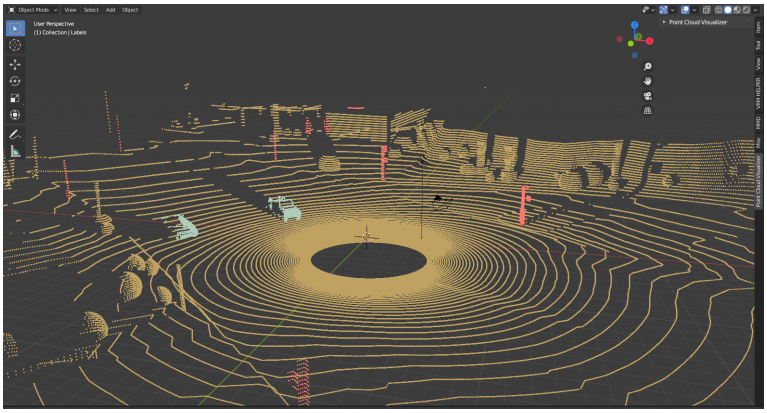


Figure 3.4: Illustration of the object classes.

- `LiDAR_PointCloud_labelsDetailed.txt`: file containing an id for each point, matching each of them to their respective game objects. Allows the differentiation between objects within the same class, as illustrated in Figure 3.5.

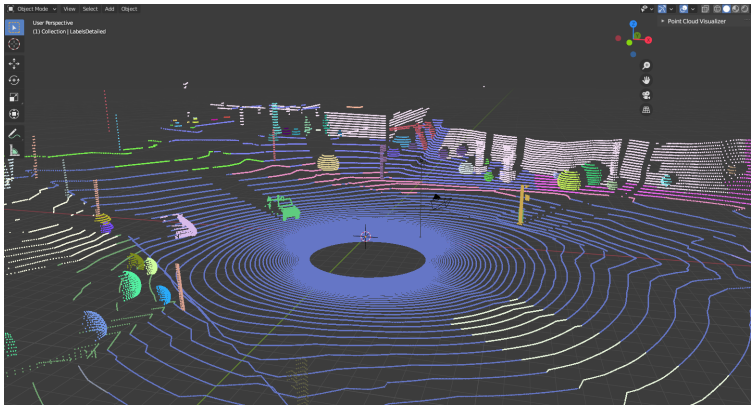


Figure 3.5: Illustration of individual identifiers for objects within the same class.

- `LiDAR_PointCloud_points.txt`: file containing the points position in euclidean space (x, y, z), point projected coordinates (x, y) pixels, and the index of the view that each point is projected onto (0 - front view, 1, or 2).
- `LiDAR_PointCloud_rotation.txt`: file containing the character rotation and forward direction, which are used by the post-processing scripts to remove the point cloud rotation around the y axis.
- `LiDAR_PointCloud_vehicles_dims.txt`: file containing all the information required to create the labels for the objects detected in the scene.

3.2 Implemented Features

Being the goal of the project to enable the generation of synthetic datasets using GTA V, it is important to establish the usability scenarios. The user should be able to capture samples manually, as to select the scenarios to be included in the dataset, and should be able to automatically generate datasets, if the target dataset size is too large and time consuming to generate by hand. The users should also be able to control the population density according to their needs. To meet these requirements, the project implements the following features:

- Ability to manually capture snapshots of the environment at the player's current position and rotation, containing all the information required for the conversion to the format of the KITTI dataset.
- Ability to record the player's traced path, for later use in automatic generation of samples.
- Ability to increase or decrease the population and traffic densities.
- Ability to generate highly detailed point clouds in less powerful systems.
- Ability to annotate entries for individual vehicles and pedestrians within a scene.
- Ability to create samples with the player inside or outside a vehicle.

3.3 LiDAR Configuration

In order for the GTA V mod to capture a point cloud for each sample, a LiDAR simulation is performed. As there are multiple LiDARs on the market with different specifications, such as angular resolution, range, vertical and horizontal FOVs, it is possible to specify the appropriate characteristics in the GTA V mod. Because the LiDAR used by KITTI was the Velodyne HDL-64E, the following configurations were given:

- `Horizontal_FOV_Min = 0`
- `Horizontal_FOV_Max = 360`
- `Vertical_FOV_Min = -21`
- `Vertical_FOV_Max = 5.8`
- `Horizontal_Step = 0.1`

- `Vertical_Step = 0.418`
- `Range(m) = 120`

A comparison between a generated point cloud and a KITTI point cloud is shown in Figure 3.6, and the properties previously configured are described as follows:

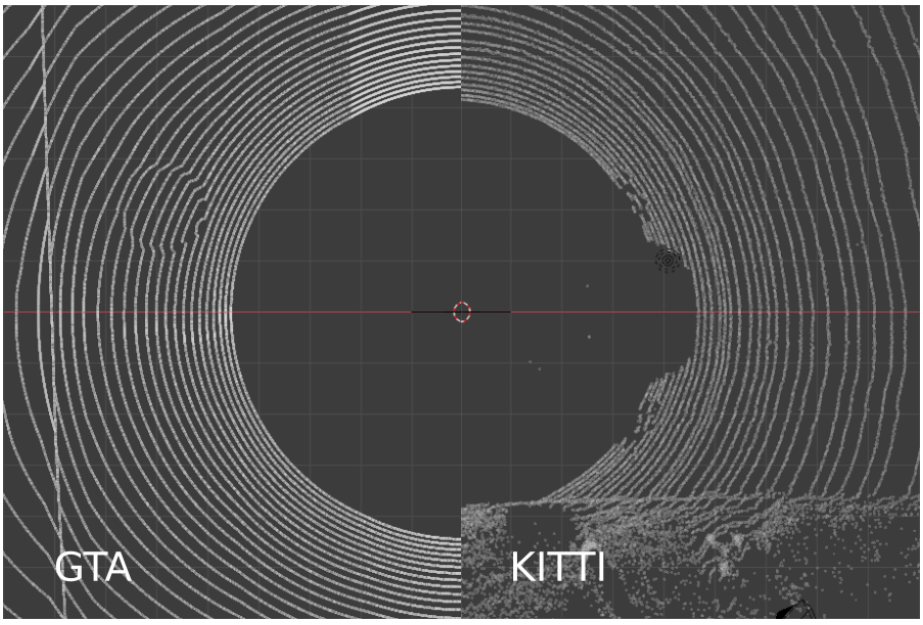


Figure 3.6: Top-down view of the point cloud generated from the GTA V mod and a point cloud from the KITTI dataset.

Horizontal Step: this parameter specifies the angular resolution, or azimuth, of the LiDAR, this case being 0.1^{o}. With all the horizontal properties, it is possible to calculate the maximum number of points that can be captured horizontally:

$$\#\ \text{Horizontal Points} = (\texttt{Horiz\_FOV\_Max} - \texttt{Horiz\_FOV\_Min})/\texttt{Horiz\_Step} \tag{3.1}$$

Meaning that, at every vertical step, there will be a maximum of 3600 points.

Vertical Step: this parameter specifies the vertical resolution of the LiDAR, in this case being 0.5^{o}. The maximum number of points that can be captured vertically is given by the equation:

$$\#\ \text{Vertical Channels} = (\texttt{Vert\_FOV\_Max} - \texttt{Vert\_FOV\_Min})/\texttt{Vert\_Step} \tag{3.2}$$

Meaning that, at every horizontal step, there will be a maximum of 64 points.

Range: specifies the maximum distance detectable by the LiDAR.

The configuration file also provides two ways of specifying the error for the generation of the point cloud with noise:

- **Error:** this parameter specifies how much noise to apply to every point in the point cloud, in order to create a point cloud with measurement errors.

- **ErrorDist:** this parameter is used to specify the method for generating the error that is added to each point:
  - 0: generate a random depth error
  - 1: generate an error relative to the distance
  - 2: generate an error using the dist_error.csv file

3.4 Dataset generation using GTA V

3.4.1 Setup of GTA V Mod

This project was developed with the version 1.0.1868.0 of ScriptHookV and the following files were added at the root of the GTA V game directory in order to activate the mod:

```
Grand Theft Auto V/
├── LiDAR GTA V/
│   ├── dist_error.csv
│   └── LiDAR GTA V.cfg
├── LiDAR GTA V.asi
├── NativeTrainer.asi
└── ScriptHookV.dll
```

The files `ScriptHookV.dll` and `NativeTrainer.asi` are part of ScriptHookV, and the file `LiDAR GTA V.asi` is the output of the GTA V mod project.

3.4.2 Project development

The mod requires input handling for the functionalities that it provides, and because part of the operations used in the automatic generation of samples are the same as the ones used in a single manual snapshot, the use of the state-based programming paradigm was chosen.

The **manual snapshot**, which is used to create a sample of the environment, at the players current position and rotation, involves the following three phases:

- **Phase 1:** In the preparation phase, the LiDAR settings are read and the player, vehicle, LiDAR and camera are modified in order to properly create the sample. It is comprised of the following steps:
  1. Read the LiDAR configuration parameters from the `LiDAR GTA V.cfg` file.
  2. Get the player's current position.
  3. Set the LiDAR origin at 1.7 m above ground.
  4. Turn the player invisible in order to not obstruct the view.
  5. Get player current rotation in order to apply it to the camera.
  6. Modify the camera and LiDAR origins in case the player is inside a vehicle. The LiDAR and camera are positioned at the same location.

7. Initialization of all the buffers that hold the sample information.
8. The time scale is set close to zero to freeze every animation and objects, and all the HUD elements are hidden from view.

- **Phase 2:** This phase handles the LiDAR simulation for the creation of the point cloud. It also has the task of stopping the movement of any vehicle intercepted by the raycasts of the LiDAR, and gathering their information for the automatic annotation.
- **Phase 3:** The last phase is when the nine views are captured, three views at 0^{o}, 120^{o} and 240^{o} for the day, night and cloudy weather conditions. It is at this stage that all the output files of the sample are created and the character is turned visible. The time scale is also reset to 1.

The **automatic dataset generation** is performed in two steps:

1. At first, the user needs to record the positions and rotations of the character while traveling through the map, by clicking a key to start the recording and to stop when finished. The positions and rotations are stored in a file every two seconds, and the interval can be adjusted within the mod.
2. After recording a list of positions and correspondent rotations, the user needs to click another key to start the automatic dataset generation, where the character is sequentially teleported to each position and rotated accordingly to replicate the initial character conditions.

3.4.3 Details

Details regarding the generation of a single sample:

Phase 1 - Preparation

- **Camera FOV:** the camera that is used to create the image views is set to have an FOV of 50^{o}, despite the use of 90^{o} horizontal FOV cameras in the KITTI dataset. This is done in order to decrease the distortions produced by the lenses.
- **Raycasting from within the player's collider:** in step 6, the character is turned invisible in order to hide the character's mesh from the screenshots, as the camera is positioned inside the head.

  Raycasting is the act of drawing a ray, with an origin and destination positions, that returns the intersection position with a collider and the properties associated to the corresponding object. Turning the player invisible does not remove the attached colliders, but the fact that the origin of the raycasts is from within the player's collider, the rays ignore any intersections with it and proceed to collide with the colliders associated to other objects in the scene.
- **Determining the LiDAR target position:** the LiDAR has to be at 1.7 m above ground in order to match the height used by KITTI. The camera is also positioned at the same location for simplicity, as it does not require any translations when transforming a point from the LiDAR coordinate system to the camera coordinate system.

The target positions for both sensors depends on whether the player is inside or outside a vehicle:

- if inside a vehicle, the target x and y coordinates are the same as the origin of the vehicle (which is located at its center).
- if outside a vehicle, the target x and y coordinates are the same as the origin of the player (also located at its center).

The height of the used vehicle or character is variable, so in order to position the sensors at 1.7 m above ground, a ray is castted from the player's origin position in the down direction ($-z$), and the position returned by the intersection with the floor is used as the ground level height for the calculation of the target height.

Phase 2 - LiDAR Simulation

- **Handling simulations of high resolution LiDAR's:** when the vertical and horizontal steps target high resolution LiDAR point clouds, if the computer is not capable of simulating all the raycasts required (one for every potential point in the point cloud) fast enough in one frame of the game, Windows will deem the process unresponsive and crashes the game. To overcome this problem for lower end CPUs, the implemented solution involved the division of the calculation of the point cloud into several game frames instead of only one. This way, the number of frames required to complete a point cloud can be adjusted accordingly.

  An effect of this approach is that vehicles going at a given speed in the moment that a snapshot was initiated, could move in between frames. To overcome this problem, when a ray intersect a vehicle, it's velocity is set to 0.

- **Identifying individual objects within the same class:** initially, the mod was only capable of detecting the classes "Vehicle", "HumansAnimals", "GameProp" and "RoadsBuildings" for each point of the point cloud, and did not differentiate between objects within the same class. To implement automatic annotations for the detected objects within a scene, for example to obtain their dimensions, position and rotation, it is necessary to individually identify objects. When a ray intersects an object, part of the data that it returns is an object handle. This handle allows for the discovery of the object's hash, which is the identifier used to derive all the available information about the object. It is stored in the file `LiDAR_PointCloud_vehicles_dims.txt` and consists of the following fields:

  - entity (unique identifier that is generated when a game object is instantiated)
  - hash (identifier of the 3D model used by the instantiated game object, allowing for the derivation of the object's properties, such as its dimensions. Two game objects based of the same 3D model have the same hash)
  - min. coordinates (x, y, z)
  - projected min. coordinates (x, y)
  - max. coordinates (x, y, z)

- projected max. coordinates (x, y)
- position (x, y, z)
- rotation (x, y, z)
- projected center (x, y)
- dimensions (x, y, z)
- object type ("Car" or "Pedestrian", which are the object types currently annotated by the mod)
- truncated (currently always set to zero)
- forward vector (x, y, z)

Each target object has a "box" defining the volume that they occupy, making it possible to derive its dimensions, the minimum and maximum coordinates in 3D space. The information about the dimensions of the vehicle is provided by an inner function belonging to the GTA V SDK.

There are specific cases where the box does not perfectly fit the object, for example, when it has adornments. However, during development and testing, sometimes the obtained box dimensions did not perfectly match the form of the vehicle, diverging a few centimeters on each side. The cause for these spontaneous error was not fully grasped, it could be a bug in the mod, an inaccuracy of the returned dimensions by the GTA V SDK function and it could also be specific to certain vehicle 3D models. By visualizing the ground truth 3D bounding boxes from the generated dataset, the problem can be found and is illustrated in Figure 3.7.

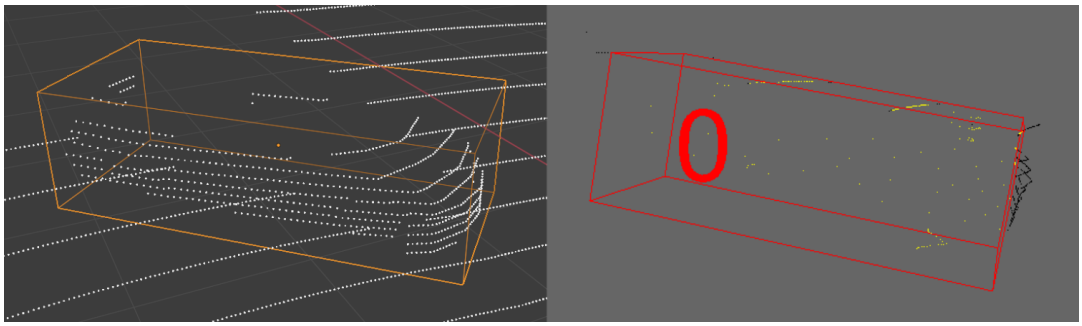


Figure 3.7: On the left, a sample with a 3D bounding box that perfectly fits the vehicle. On the right, a sample with a 3D bounding box that does not perfectly fit the vehicle.

3.5 Conversion of GTA Dataset to KITTI Format

After generating a dataset using the GTA V mod, the next step is to convert the samples into KITTI formatted ones. The conversion is made using Python because it is a language that allows for fast prototyping, it is the most popular in the Machine Learning field, having available a vast number of related tools and it is cross-platform.

Prior to converting the dataset, there are several options that can be changed in order to adjust the conversion to the desired goals. These options are implemented in the form of command line arguments in the Python project:

- Toggle between the the selection of the ideal point cloud or the point cloud with noise for the converted dataset.
- Toggle to include a specified intensity value for every point of the point cloud. This is a relevant option because several architectures were trained using intensity as one of the points' attributes.
- Toggle sample filtering based on the specified maximum object distance to the LiDAR origin. When turned on, all the samples containing detectable objects that are beyond the maximum distance ignored.
- Toggles to select which objects should have labels in the converted dataset. The only two detectable objects are vehicles and pedestrians.
- Selection of the target architecture, as each architecture can use just a part or all the data available for each sample.
- Toggle to generate an additional directory with point clouds that only contain individual objects obtained from the dataset. The entities are centered at the origin and the points normalized into a unit sphere.
- Toggle to generate an additional directory to store binary point clouds without background points.

3.5.1 Project development

After the specification of all the desired properties, the project proceeds to convert the GTA V dataset samples into KITTI formatted samples. The program main loop iterates through all the samples in the GTA V dataset directory, and the performed process in each iteration can be summarized into three general steps:

1. Load the data from the sample's files into memory.
2. Process the data in order to format and generate the information present in the KITTI dataset samples.
3. Store the generated sample's data into the correspondent files:
    - `0xxxx.bin`: point cloud stored in binary format
    - `0xxxx.png`: front view image with dimensions of 1224×470 pixels
    - `0xxxx.txt`: calibration file with the 7 transformation matrices
    - `0xxxx.txt`: label file with 15 values, as shown in Table 2.3

3.5.2 Details

The data provided by the GTA V mod cannot be directly used by the Deep Learning algorithms as it only forms a base for deriving the required information.

Point cloud

The point cloud obtained from the GTA V mod is rotated around the up axis by the character's/vehicle's rotation in that sample. By loading a point cloud from the KITTI dataset, it is possible to conclude that the point cloud should always be facing the x axis, as shown in Figure 3.8.

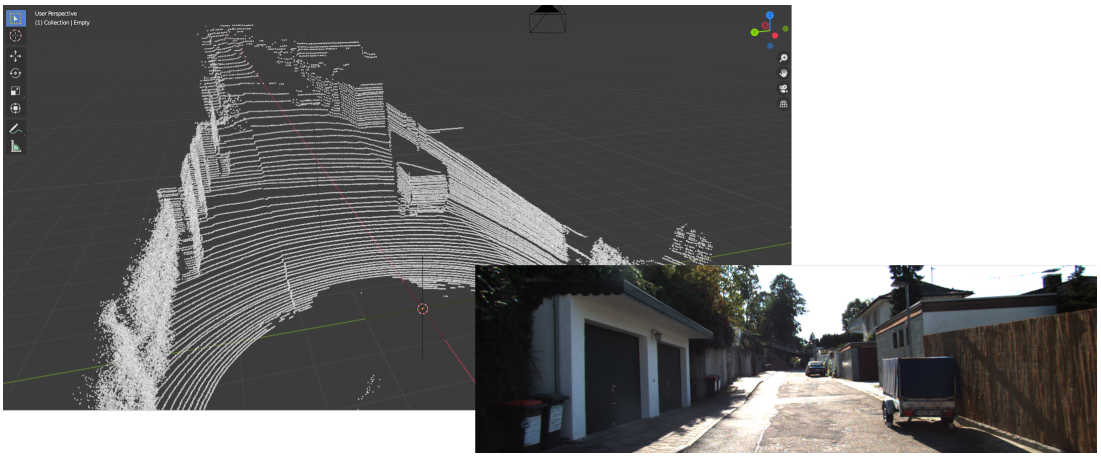


Figure 3.8: KITTI sample illustrating the relation between the direction of the point cloud and the direction that the vehicle is facing.

As such, the original point cloud rotation is negated to obtain a point cloud facing the y direction, which is then subtracted by 90^{o} to make the point cloud face the x direction. If the original rotation is negative, 360^{o} are first added in order to make it positive.

$$\text{target\_rot} = -\text{original\_rot} - 90. \tag{3.3}$$

Calibration matrices

- **P0, P1, P2 and P3 matrices:** even though there is only one front view image of the vehicle in the generated dataset, it is necessary to specify these four matrices. P0 is the matrix used to project a point in 3D camera coordinates to the front view image pixel coordinates, and the rest of the matrices are equal to P0. The architectures that only use one front view image (commonly the left image), only use the P0 matrix. This matrix is also denoted as "K Matrix", containing the intrinsic camera parameters. Because there is no information about the camera parameters used in the GTA V mod, it is calculated using the following matrix parameters [6]:

$$P0 = \begin{bmatrix} f & 0 & Cu & 0 \\ 0 & f & Cv & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \tag{3.4}$$

  where f is the focal length, Cu is the half screen width, Cv is the half screen height.

The use of 75^{o} for the FOV parameter, used to calculate the focal length, resulted in the best projection results, as shown in Figure 3.9. It is obtained by using the P0_mat matrix to project the LiDAR points to the front view image and coloring the projected point cloud points with a depth-based gradient. As a point approaches one of the sides of the images, the distortion of the image becomes more prominent.

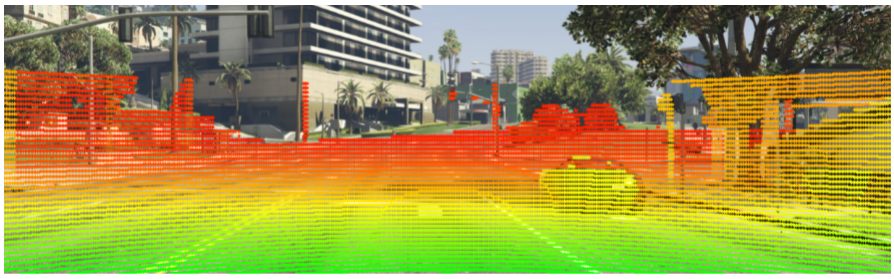


Figure 3.9: Output of the sample validation scripts available in the Frustum PointNet project, on a sample produced by the GTA V mod.

- **R0_rect matrix:** when capturing the front view image in the GTA V mod, the FOV of the camera was set to 50^{o} in order to reduce the lenses distortions. As the effects of the distortion are not very pronounced at the central part of the image, the identity matrix was used as the rectifying matrix for simplicity.
- **Tr_velo_to_cam:** this matrix is used to transform a point from the GTA point cloud coordinate system to the reference camera (image) coordinate system. Both GTA V point cloud and camera are represented in the right-handed coordinate system, and are shown in Table 3.1.

Coordinate System	x	y	z
Camera KITTI	right	down	forward
Velodyne KITTI	forward	left	up
GPS/IMU KITTI	forward	left	up
Image GTA V	right	down	forward
GTA V LiDAR pointcloud	right	forward	up

Table 3.1: Coordinate system of the KITTI sensors and the GTA V generated point cloud and front view image.

It is relevant to note that, by using Direct X as the graphics API, GTA V uses the left-handed coordinate system. However, the generated point cloud, by the GTA V mod, has into consideration the right-handed coordinate system.

The GTA V image uses the same coordinate system as the KITTI camera and, to perform the transformation from the GTA V point cloud coordinate system to the camera coordinate system, a 90^{o} rotation around the x axis is required,

$$\mathrm{R}_x(90) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos 90 & -\sin 90 \\ 0 & \sin 90 & \cos 90 \end{bmatrix}. \tag{3.5}$$

Because the camera's origin is the same as the LiDAR's, the transformation does not need to include any translation. So, the Tr_velo_to_cam matrix in use is:

$$\text{Tr\_velo\_to\_cam} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \end{bmatrix}. \tag{3.6}$$

- **Tr_imu_to_velo matrix:** this matrix is not used in any of the studied architectures, so it is set as an identity matrix.

Labels

The generation of the fields for each of the objects in a label file is performed in the following way:

- **Type:** the types currently detected by the GTA V mod are 'Car' and 'Pedestrian', covering two thirds of the types used to measure the performance of the state of the art.
- **Truncated:** as a first step, the mod focuses on the labeling of fully visible objects. If the object is occluded, this field remains as not truncated.
- **Occluded:** for the same reason as in the "truncated" field, if the object is occluded, this field remains as not occluded.
- **Alpha:** as the alpha field is rarely used by object detection architectures, it is set with the default value of -10.
- **Location:** several steps are required in order to obtain the correct object's location:

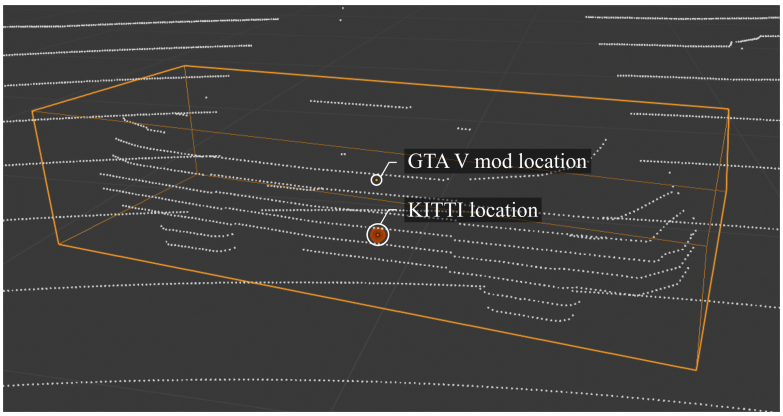


Figure 3.10: GTA V mod location vs the location used in KITTI.

1. First, the location outputted by the GTA V mod concerns the location of the center of the object. In the case of the KITTI dataset, the location of an object is located at the center of the its base, as illustrated in Figure 3.10. So, the z coordinate needs to be subtracted by half of the height of the object.
2. The location is now subjected to two coordinate system conversions, as illustrated in Figure 3.11, from the GTA V point cloud coordinate system to the camera's

coordinate system. These conversions are required because the object's location is relative the camera's coordinate system, as shown in the Table 2.3.

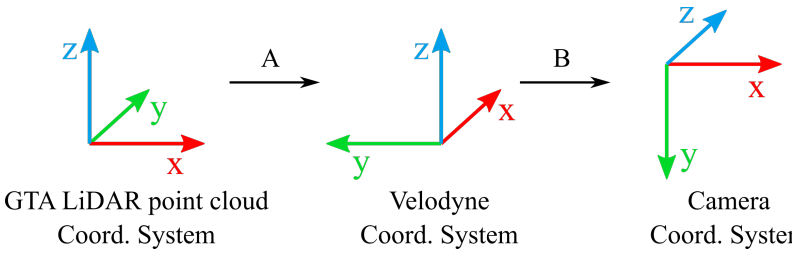


Figure 3.11: Coordinate system conversions from the GTA V point cloud object location to the correspondent location in the camera's coordinate system.

A: First, the position is rotated around the z axis by the same amount as the rotation that the point cloud was subjected to in order to be facing the x direction.

B: In order to convert from the Velodyne coordinate system to the camera coordinate system, a 90^{o} rotation around the z axis, followed by a 90^{o} rotation around the x axis is performed.

- **Bounding Box 2D:** the bounding box 2D coordinates are obtained by first calculating the 3D bounding box corners and projecting them onto the image using the P0_mat matrix. From the resulting corners, four coordinates are selected as the minx, miny, maxx, and maxy.
- **Dimensions:** the vehicle dimensions are given by the samples generated by the GTA V mod.
- **Rotation_y:** the rotation around the z axis of an object outputted by the GTA V mod is represented in Figure 3.12.

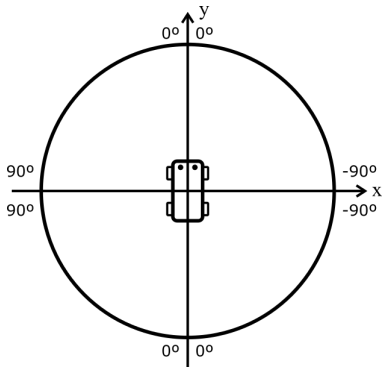


Figure 3.12: Object rotation angles.

This diagram was derived by taking a series of samples of a vehicle with different rotations, as shown in Figures 3.13, 3.14 and 3.15.

Figure 3.13: Vehicle rotations a).

Figure 3.14: Vehicle rotations b).

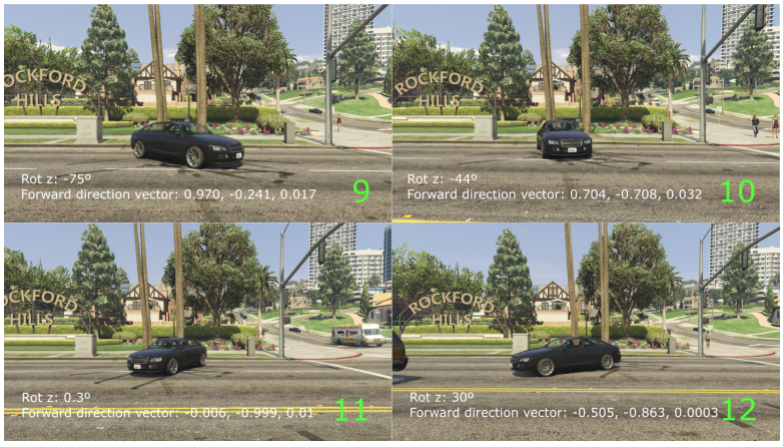


Figure 3.15: Vehicle rotations c).

The forward vector of the object enabled the derivation of the rotation in the range of [0^{o}, 360^{o}], from the object's rotation returned by the GTA V mod. After computing it, the rotation is converted to radians and set in the range [$-\pi$, π].

Front view image: the conversion from the GTA V mod front view to the dimensions of the KITTI front view images is performed in two steps:

1. Resize the image from 1920 × 1080 pixels, the resolution of the screen, to 1382 × 512 pixels, the resolution of the original KITTI images.
2. Crop the sides of the 1382 × 512 pixels image in order to obtain a 1224 × 470 pixels image 3.16.

Figure 3.16: Transformation of the 1382 × 512 pixels to a 1224 × 470 pixels image.

3.6 Summary

The developed GTA V mod is able to provide synthetic dataset generation and enables both manual and automatic generation of datasets. The output of the mod contains raw information about the scene and is not in the KITTI dataset format. A post-processing step was built to convert the mod's output to the targeted dataset format.

This mod has some limitations, involving:

- sometimes the 3D bounding boxes enveloping the vehicles do not perfectly fit their form.
- the dataset only captures the vehicle's bounding box yaw rotation, it does not take into consideration the vehicle's roll and pitch. However, the majority, if not all, of the current 3D object identification datasets for autonomous vehicles, and architectures, also do not consider inclined roads.
- the point cloud generation is only suitable to be used in relatively flat planes, as the raycasts are not adjusted according to the road slope (the pitch of the vehicle) where the vehicle is located.

As for the Python project, used to convert the GTA dataset to the KITTI format, some limitations are:

- the truncation, occlusion and alpha label fields are not calculated. This decision was

made because we targeted the generation of easy difficulty samples as a proof of concept, where the objects are not truncated and always totally visible in the front view image. The alpha is rarely used by the 3D object identifications architectures, so it was also not computed.

- the sample images are not rectified, presenting distortions at the extremes. The fact that the images are not being rectified, implied the usage of a 50^{o} FOV in order to reduce the distortions.
- the calibration matrix P0 values were obtained by assuming a 75^{o} horizontal FOV, instead of the 50^{o} FOV actually used to take the image views. The change to 75^{o} resulted in better point projection results, as shown in Figure 3.9. The problem encountered by using the value 50^{o}, and still present for the 75^{o} value, for the horizontal FOV is in part due to the unrectified front view image.
- during the conversion, a filtering of the vehicles belonging to the front view image is performed, in order to only include labels associated with these in the KITTI sample. Because each point of a vehicle's point cloud is associated to one of the image views of the sample, we can determine if a vehicle has any points in the front view sample and, if not, its label information is discarded. This approach is incomplete because an object can have points projected onto the front view image of the original GTA V size (1920×1080 pixels), but when the image is resized and cropped, to follow the KITTI format, the points of that were previously in the front view can be eliminated. This can result in situations where all, or part, of the points of the object that were associated with the original front view, after the image transformations, may no longer, or partially, appear in the KITTI-sized front view image. The current solution only relies on the mappings to the original front view due to the calibration matrix not being exact and the image not being rectified.

Chapter 4

Assessment of the Generated Datasets

In the previous chapter, the process of generating and converting a GTA dataset to a KITTI formatted dataset was described in detail. In this chapter, we proceed to evaluate the quality of the generated synthetic datasets, by experimenting with a generated dataset on several state of the art 3D object identification architectures, presented in Chapter 2.

We divided the experiments into two categories: the image-based object identification approach, comprised of the Pseudo-LiDAR and Pseudo-Lidar++, and the LiDAR-based approach.

The main criterion for the selection of Deep Learning architectures was to experiment with original techniques, thus avoiding fruitless repetition. Further criteria included public availability of the implementation and trained models, and satisfactory classification and detection results. Other factors such as inference time were not considered, as the objective is to evaluate the accuracy of the networks on the generated dataset. As a result, we chose the following architectures:

- Frustum PointNet
- PointPillars and SECOND, highlighted by Bosch©
- Pseudo-LiDAR and Pseudo-LiDAR++
- Point-RCNN
- PV-RCNN

4.1 Generated Dataset Criteria

In order to evaluate the quality of the generated samples, we decided to generate a small dataset, with a total of 50 samples. In general, it is sensible to start with a small dataset, in which samples are simple and well known to the user. This means that the generated samples must be within the Easy difficulty level defined by KITTI. We also imposed the following additional restrictions: a sample must only contain one vehicle, fully visible and without any

truncation, and the samples should be taken during the day. In order to accomplish these requirements, manual generation of samples is necessary. Figure 4.1 shows a subset of images belonging to the generated dataset.

Figure 4.1: Front view images of samples from the generated dataset.

4.2 LiDAR-based 3D Object Identification

4.2.1 Frustum PointNet Framework

The Frustum PointNet framework evaluates samples residing in the `dataset` directory, located in the root directory of the project. The samples must be organized in the following structure:

```
dataset/KITTI/object/
├── testing/
│   ├── calib/
│   ├── image_2/
│   └── velodyne/
└── training/
    ├── calib/
    ├── image_2/
    ├── label_2/
    └── velodyne/
```

Dataset samples validation

Prior to dataset validation, Frustum PointNet provides the `kitti_object.py` script to validate the samples. It allows for the visualization of the projected 2D and 3D bounding boxes, as shown in Figures 4.2 and 4.3;

Figure 4.2: Projected 2D bounding box.

Figure 4.3: Projected 3D bounding box.

and of the 3D bounding boxes of the vehicles residing within the portion of the point cloud that is projected onto the front view image, as show in Figure 4.4.

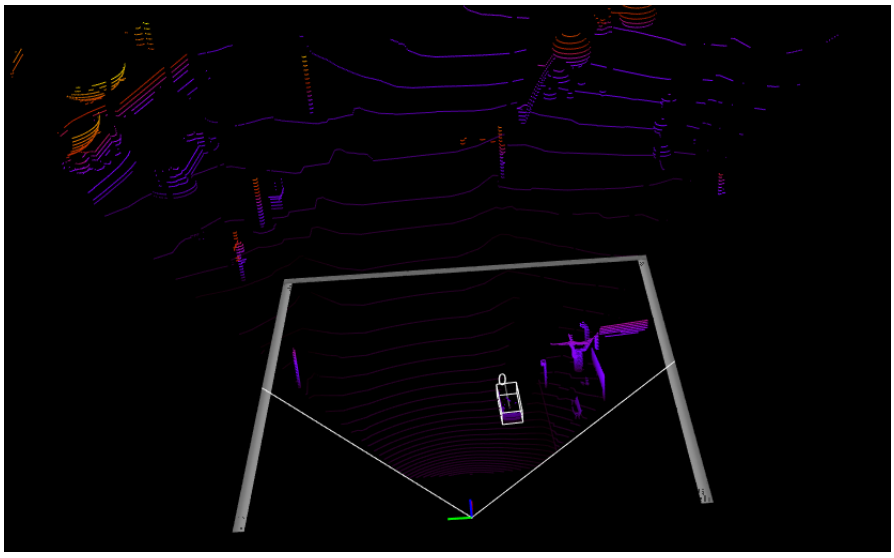


Figure 4.4: Projected 3D bounding box.

2D object detection and classification

Frustum PointNet relies on the usage of a 2D object detector and classifier to detect objects in the front view image. In addition to the KITTI formatted dataset, 2 types of files are required in order to test or train the model:

- `val.txt`: a file containing the list of the sample ids constituting the correspondent

dataset split.

- `rgb_detection_val.txt`: a file containing the output information of the 2D object detector. For each object detection, it specifies the origin sample, the inferred object class, the classification confidence and the coordinates of the correspondent 2D bounding box (xmin, ymin, xmax, ymax).

As the objective of this thesis focuses on 3D object detection, and in order to compare the Frustum PointNet results with the rest of the selected 3D object detection architectures without the influence of a 2D object detector's performance, we decided to supply the `rgb_detection_val.txt` file with ground truth 2D bounding boxes and classifications. A Python script was implemented to create this file based of the dataset labels. However, this approach gives an upper hand to this architecture because it leverages the 2D predictions to partition the input point cloud into several point clouds containing one vehicle each, which facilitates the 3D detection.

Dataset format conversion

After supplying the architecture with all the necessary input data, the first step towards testing or training a model consists in the conversion of the dataset's labels to the PICKLE format, which is the format processed by Frustum PointNet. This is performed by the `prepare_data.py` script, and the outputted pickled file will have the following information:

- `id_list`: list of the sample ids specified in the `val.txt` file, corresponding to the validation split.
- `box2d_list`: list of the minx, miny, maxx, maxy coordinates of the detected objects in the front view image.
- `box3d_list`: list of the 3D bounding boxes corner coordinates, in the camera coordinate system.
- `input_list`: list of the x, y, z and intensity of each point, in the camera coordinate system, that is projected within the 2D bounding box boundary of the detected object. It consists of the points comprising the frustum point cloud associated to the vehicle.
- `label_list`: list that identifies if an object is to be detected or ignored, based on properties such as number of points and distance.
- `type_list`: list specifying the class of each detected object (car, pedestrian, cyclist).
- `heading_list`: ry (rotation around the y axis of the object, in the camera coordinate system.
- `box3d_size_list`: list with the dimensions (h, w, l) of each detected object.
- `frustum_angle_list`: the frustum angle is the angle of the 2D bounding box center coordinates relative to the x axis.

The Frustum PointNet project also provides the `provider.py` script for the visualization of of each sample's frustum point clouds in the pickled file, which is illustrated in the Figure 4.5.

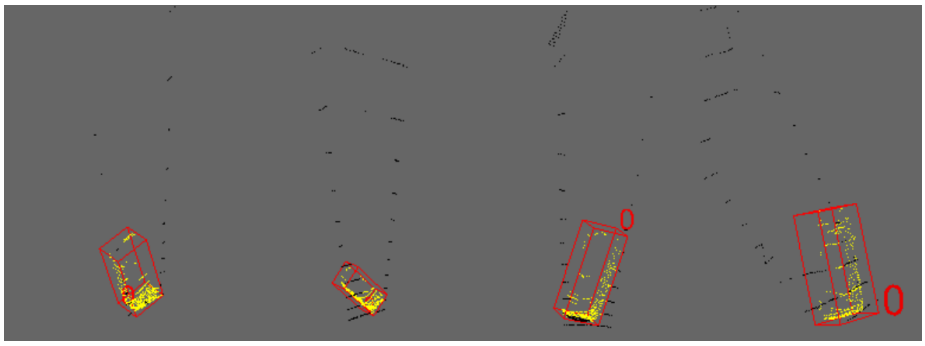


Figure 4.5: Visualization of frustum point clouds from the samples of the generated dataset, listed in the pickled file.

The `prepare_data.py` script also provides an option to visualize the computations performed during the conversion. Figure 3.9 is one of the outputs.

After performing these required steps for the sample's validation and dataset conversion to the file formats processed by Frustum PointNet, the dataset is now ready to be evaluated.

Importance of point intensity

The point clouds provided by the KITTI object detection dataset, provide 2 attributes for each point: the point coordinates in 3D space and its intensity value, ranging from 0 to 1.

As the intensity value cannot be easily simulated in GTA V, dummy intensities values were associated to the point cloud points of the generated sample. Initially, we assigned the value 1, maximum intensity, to all the points and evaluated on the Frustum PointNet architecture. Because the 3D object detection results were close or equal to zero, we decided to implement a visualization script to show the intensity values of the points of a real point cloud from one of the KITTI samples. Figure 4.6 represents the output of this script, where the gradient represents the interval of the possible intensity values, from red (intensity = 0) to green (intensity = 1).

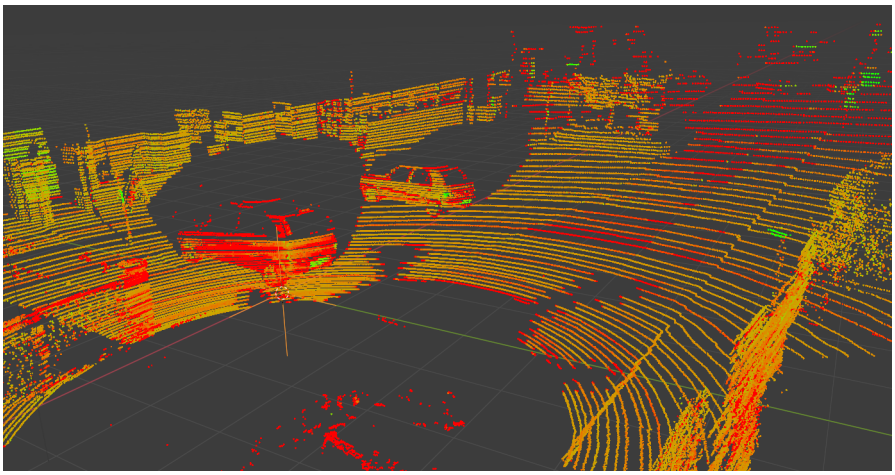


Figure 4.6: Visualization of the intensity of each point in a KITTI point cloud.

As in Figure 4.6, the majority of the points actually have intensities closer to 0. The points

with intensity closer to 1 are from metallic surfaces perpendicular to the LiDAR's beam direction, or from reflective materials, such as vehicle's registration plates and traffic signs, which are associated with high intensity values even when not perpendicular to the LiDAR's beam direction.

Because the majority of the point intensity values are closer to 0, we decided to use the intensity value of 0 for all the points of the synthetically generated point clouds. After testing with this new modification, the results were more satisfactory, as shown in the tables presented in Section 4.2.3.

From the observed improvements when only changing the intensity value from 1 to 0, we also conclude that networks trained with point intensity values are indeed dependent on them to achieve the demonstrated performance results. If we wanted to use point clouds without intensity values, we would need to retrain the network from the start, without the intensity in the point cloud samples.

4.2.2 OpenPCDet Toolbox

The OpenPCDet toolbox is a LiDAR-based open source project for 3D object detection [69]. This project aims to facilitate the training and testing of multiple state of the art 3D object detection architectures, by providing both simple terminal commands and already trained datasets for each of the provided models. It performs evaluations using 11 and 40 recall positions, and currently supports the following architectures:

- PointPillar
- SECOND
- PointRCNN
- Part-A^2-Free
- Part-A^2-Anchor
- PV-RCNN

Because this work was first released in March 2020, with its latest update released at the end of November, even though it proved to be very useful during our experiments, we were not able to fully experiment with it in this thesis.

4.2.3 Results

Using the pre-trained LiDAR-based 3D object detection models, provided by Frustum PointNet and OpenPCDet, the generated dataset comprised of 50 samples was evaluated and compared to the performance of the same models on the KITTI validation and test sets. The metrics used are the Average Precision for 2D, bird's eye view and 3D object detection, and the Average Orientation Similarity. It should be noted that, Frustum PointNet does not provide Average Precision results for 40 recall positions, in contrast to the OpenPCDet.

All the results presented in the following tables related to the evaluation on the KITTI dataset are obtained from the reviewed papers, and the table rows are ordered according to the results on the GTA dataset.

2D object detection results

Table 4.1 presents the 2D detection results obtained from the GTA V generated dataset on the selected architectures. As these networks do not perform 2D object detection, the correspondent papers do not present their results on this evaluation metric.

Apart from Frustum PointNet, the selected architectures do not rely on the front view image to perform object identification. Thus, the Average Precision for 2D object identification, presented in the following tables, was obtained from the projection of the inferred 3D bounding boxes onto the front view image.

Architecture	GTA AP R11	GTA AP R40
PointPillars	73.14	74.17
Point-RCNN	76.38	76.89
PV-RCNN	78.13	81.96
SECOND	82.43	83.64
Frustum PointNet	100	100

Table 4.1: Average Precision (%) of 2D object detection using 11 and 40 recall positions.

In both tables, the score of the Frustum PointNet is 100% because the ground truth 2D bounding boxes and correspondent classifications are supplied to this network, as explained in Section 4.2.

Once 3D object identification starts approaching maturity-level performance, a comparison could also be made between their results and the ones from the state of the art 2D object detection architectures. To do so, the 2D object detection architectures should be trained on the KITTI dataset front view images, and evaluated on the same test set.

Bird's eye view detection results

The bird's eye view, and in part 3D, object detection results, rely on the calculation of the IoU of 2D bounding boxes from the top-down perspective. In contrast to the IoU calculation used to evaluate 2D object detection in images, the bird's eye view bounding boxes are influenced by the rotation around the up axis. This makes the calculation of the overlapping area for rotated bounding boxes more complex than calculating for axis-aligned bounding boxes. Figure 4.7 illustrates the challenges, where a rotated bounding box can produce polygonal overlapping areas with variable number of edges and shapes.

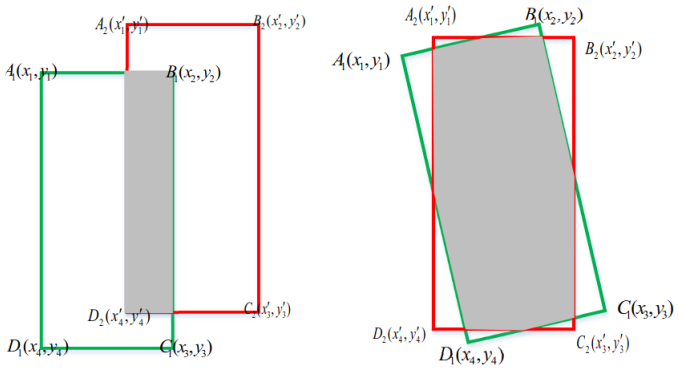


Figure 4.7: Comparison between the overlap area of axis-aligned bounding boxes and rotated bounding boxes [78].

Because the 3D object detection field of study is not as widespread as 2D object detection, a project specifically for the calculation and visualization of bird's eye view results could not be found. As such, with the objective to better understand the results obtained from the performed experiments, we implemented an IoU calculation tool, with bounding box visualization capabilities, following the steps described in the paper [78]. The calculation of the IoU between the ground truth and predicted bounding boxes is performed as follows:

1. Determine the bird's eye view area of the ground truth and inferred bounding boxes.
2. Determine the corners of the polygon that defines the overlapping area, using the Equations 4.1 and 4.2.

$$\text{Area}_{\text{gt}} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}. \tag{4.1}$$

$$\text{Area}_{\text{pred}} = \sqrt{(x'_2 - x'_1)^2 + (y'_2 - y'_1)^2}. \tag{4.2}$$

3. Determine the vertices of the overlapping area.
4. Order the points in the clock-wise order. We used the bubble sort algorithm and provided the center of the overlapping area and its vertices.
5. Calculate the area of overlap. We calculated the area of the polygon using the Shoelace formula, where the input points must be sorted clockwise or anti-clockwise.
6. Calculate the IoU of the bird's eye view bounding boxes using Equation 4.3.

$$\frac{\text{Area}_{\text{overlap}}}{\text{Area}_{\text{gt}} + \text{Area}_{\text{pred}} - \text{Area}_{\text{overlap}}}. \tag{4.3}$$

After the implementation of the IoU calculations for rotated bounding boxes, we proceed with an analyses of the bird's eye view object detection results, presented in Table 4.2.

Architecture	GTA AP R11	GTA AP R40	KITTI AP test R11	KITTI AP test R40	KITTI mAP val R11	KITTI mAP test R11
Frustum PointNet	7.63	—	88.70	91.17	—	65.39
Point-RCNN	40.72	38.51	—	92.13	—	—
PointPillars	40.97	38.61	88.35	90.07	—	66.19
SECOND	57.23	54.96	88.07	89.39	—	60.56
PV-RCNN	60.72	58.17	—	94.98	95.76	—

Table 4.2: Average Precision (%) of BEV object detection using 11 and 40 recall positions.

As the result of Frustum PointNet on the 11 recall average precision was unreasonably low compared to the rest, we visualized the overlapping bird's eye view areas, shown in Figure 4.8, and calculate the IoU of the bird's eye view and 3D detections.

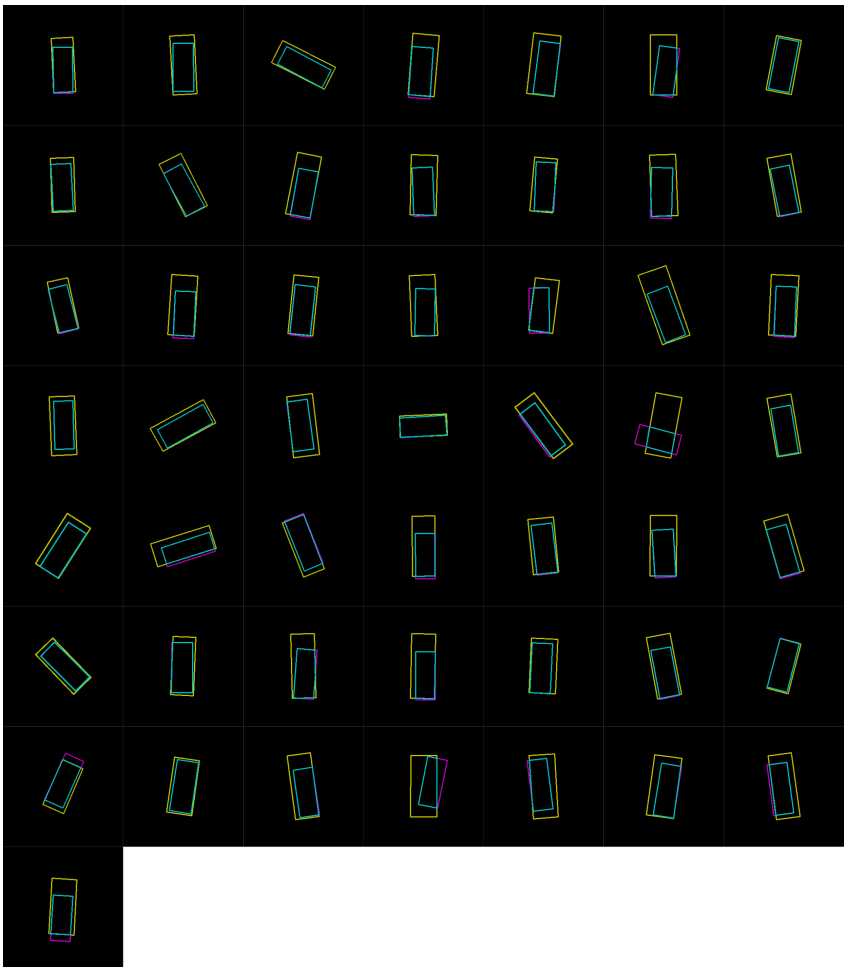


Figure 4.8: Visualization of the IoU of the predictions made by Frustum PointNet. Yellow represents the ground truth BEV bounding box, magenta the inferred BEV bounding box and cyan the overlapped area.

With the additional visualizations, several conclusions were drawn:

- Out of the 50 samples of the generated dataset, major errors produced by Frustum PointNet happened in only 2 of them, as illustrated in Figure 4.10.
- Frustum PointNet has difficulty in predicting the dimensions and location of the vehicles while taking into account the occluded parts to the LiDAR sensor. Figure 4.9 illustrates this by highlighting in red the zones without points.

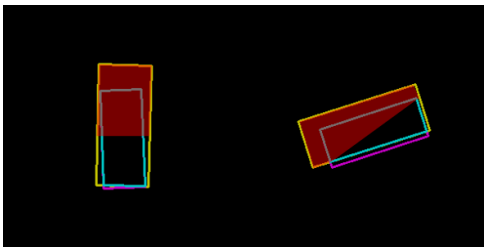


Figure 4.9: Visualization of the errors produced by Frustum PointNet on selected GTA predictions.

- Even though the Frustum PointNet predictions resulted in a very low average precision compared to the rest of the architectures, visually the predicted bounding boxes were inline with the ground truth.
- The observed discrepancy in the result of the Frustum PointNet, compared the the other models, might also be due to the its own implementation of the algorithm used for the evaluation, which is already 3 years old, as the rest of the architectures leverage the evaluation algorithm provided by OpenPCDet.
- The models can be severally affected not only from the predicted bounding box dimensions and position, but also from the inferred orientation. Figure 4.10 illustrates this point.

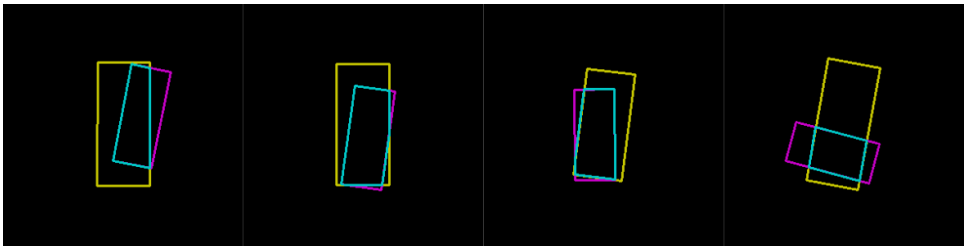


Figure 4.10: Visualization of erroneous orientation predictions.

- Apart from the possible errors induced by the nature and imperfections of the generated dataset, another possible contributor for the difference between the Frustum PointNet and the other architectures' results could be due to the model's inability to generalize as well to the synthetic data obtained from GTA V virtual environments, for example, to the generated ideal point clouds.

The majority of the tested architectures were able to obtain an average precision greater than 40%, which is about half the performance obtained from the KITTI dataset. It is important to note that the performance results are hurt by the not perfectly sized ground truth bounding boxes obtained by the GTA mod and from the synthetic nature of the dataset. Even with

these aspects in mind, it can be concluded that the models are able to generalize well to input data of different origins than the original dataset.

3D object detection results

The 3D object detection results are shown in Tables 4.3 and 4.3.

Architecture	GTA AP	KITTI AP val	KITTI AP test
Frustum PointNet	1.81	83.76	81.20
Point-RCNN	17.80	88.88	85.94
PointPillars	26.62	—	79.05
SECOND	28.20	87.43	83.13
PV-RCNN	42.15	—	—

Table 4.3: Average Precision (%) of 3D object detection using 11 recall positions.

Architecture	GTA AP	KITTI AP test
Frustum PointNet	—	82.19
PointPillars	15.79	82.58
Point-RCNN	22.98	86.96
SECOND	23.82	83.34
PV-RCNN	39.12	90.25

Table 4.4: Average Precision (%) of 3D object detection using 40 recall positions.

The prediction of the 3D bounding boxes is more difficult than the bird's eye view counterpart due to the added dimension, so relatively lower performance was to be expected. It is affected by not only the inferred vertical position of the vehicle, but also with the predicted vehicle's height. It is also important to note that the order of the architectures' results is the same as in the BEV results. This leads to the conclusion that the inference of the vehicles' heights performed by the SECOND architecture has inferior performance compared to the PV-RCNN network, as it greatly impacted its 3D object detection results. From the bird's eye view to the 3D object detection, the performance of PV-RCNN passed from 60% to 42%, while for SECOND it passed from 57% to 29%.

Average Orientation Similarity

According to the Table 4.5, the performance difference between the architectures on the GTA and KITTI datasets are similar, with PV-RCNN obtaining the best performance overall and the PointPillars the worst, not taking into account Frustum PointNet.

Architecture	GTA AOS	KITTI AOS test
PointPillars	35.64	90.19
PV-RCNN	35.87	—
Point-RCNN	39.04	—
SECOND	40.02	90.61

Table 4.5: Average Orientation Similarity (%) results. The evaluation scripts used by Frustum PointNet do not include a measurement for the AOS.

4.2.4 Final Remarks

Even though the results obtained from the generated GTA dataset were akin to the results obtained from the KITTI dataset, the fact that some of the tested architectures were able to achieve results above the 40% mark in several of the evaluation metrics means that the generated dataset is well structured and conforms to the format stipulated by KITTI.

To further improve on the quality of the generated datasets, the following limitations should be addressed:

- The ground truth bounding boxes returned by GTA do not perfectly fit the objects of interest.
- The point clouds used for the evaluation were ideal point clouds without noise, which does not properly represent point clouds produced from real LiDAR sensors.
- The constant intensity through all the points of the point clouds. We saw that by using a value of 1, it drastically changed the results for the worst. Associating intensity with slight variations, greater intensity to reflective objects and points belonging to surfaces perpendicular to the LiDAR beams might result in better performance.
- The image view contains distortions, while the images provided in the KITTI dataset are rectified. The K matrix, used to convert camera coordinates to pixel coordinates could also be improved after addressing this issue.
- The values for the occlusion and truncation levels, and for the alpha, are not calculated. At least the occlusion and truncation levels should be addressed in order for the evaluation metrics to be able to separate samples according to their difficulty level.

Yet another reason for the lower performance results is that the produced dataset is visually different from the KITTI dataset, on top of being obtained with ideal (noiseless) sensors. Further work in making the synthetic dataset to look real must thus be undertaken.

Given the results obtained from the generated synthetic dataset, and the promising bird's eye view IoU visualizations for an architecture that resulted in the overall worst performance, we can conclude that the generated dataset can be used to evaluate and compare 3D object identification algorithms.

Also, if we compare the results of the architectures on the KITTI and the generated GTA V datasets, we can observe that they match in the architectures with the lowest performance – Frustum PointNet – and with the highest performance – PV-RCNN. This observation is

another sign of a correctly generated synthetic dataset and of the iterative improvements on the published models generalization capabilities to input data of different origins.

Bear in mind that such conclusions must be deemed as preliminary, as these are based on a first, small and simple dataset. A bigger, more representative dataset would be required to confirm such conclusions.

4.3 Image-based 3D Object Identification

Image-based 3D object identification is an alternative approach to the standard 3D object identification approaches, that relies on monocular or stereo vision camers, much cheaper than LiDAR and yet with much higher resolution, in order to create a point cloud representation through depth estimation techniques.

This study was motivated by Bosch Car Multimedia Portugal and the main objective was to determine whether the image-based architectures were capable of producing relevant results when fed by synthetic samples obtained from the GTA V mod.

More specifically, the study aimed at:

- Capturing stereo front view images within the GTA V mod.
- Generating point clouds using pseudo-LiDAR(++) from those views.
- Evaluating single objects within the generated point clouds using object classification algorithms, namely the PointNet architecture.

The proposed study fit within the scope of the 4th year Machine Learning course, lectured by professor Pétia Georgieva, and was performed by cooperating and supervising two groups of students, consisting of Marco Silva and Rafael Maio, and Tomás Costa and João Marques.

4.3.1 Stereo-image capture in GTA V mod

The first task consisted in the capture of stereo images using the GTA V mod. The performed process is illustrated in Figure 4.11 and is comprised of the following steps:

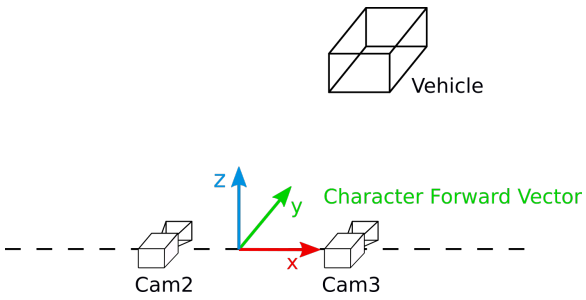


Figure 4.11: Capturing stereo-image of a vehicle in GTA V.

1. Obtain the character's/camera's forward vector.
2. Rotate the forward vector 90° around the z axis to obtain the local x axis vector of the camera.
3. Multiply the camera's x axis vector by a certain amount both ways, in order to replicate

the distance between the color cameras used by KITTI, of 54 cm.

An example output is the stereo image shown in Figures 4.12 and 4.13.

Figure 4.12: Left camera image view.

Figure 4.13: Right camera image view.

The following subsections 4.3.2 and 4.3.3 describe the process used by Pseudo-LiDAR and Pseudo-LiDAR++to generate point clouds from the input stereo images, given the sample show in Figures 4.12 and 4.13.

4.3.2 Pseudo-LiDAR

Depth estimation using PSMNet

The authors of Pseudo-LiDAR used the PSMNet architecture for stereo depth estimation, and made available an improved version with added optimizations and bug fixes, which is the one used in our experiments. Based on the stereo image represented in Figures 4.12 and 4.13, the PSMNet model, trained on the KITTI dataset, generated the depth map in Figure 4.14.

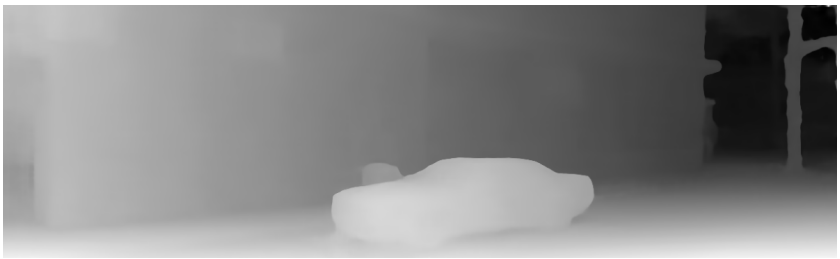


Figure 4.14: Result of the PSMNet trained on the KITTI dataset.

The visualization of the depth map represents farther away objects from the camera as black and closer objects as white. As it can be seen in Figure 4.14, PSMNet was able to discern the vehicle from the background object and building.

Depth map to point cloud conversion

After obtaining the depth map correspondent to the stereo image, the next step of the Pseudo-LiDAR architecture consists in the generation of a dense point cloud through the backprojection of all the pixels into 3D coordinates. The resulting dense point cloud is shown in Figure 4.15.

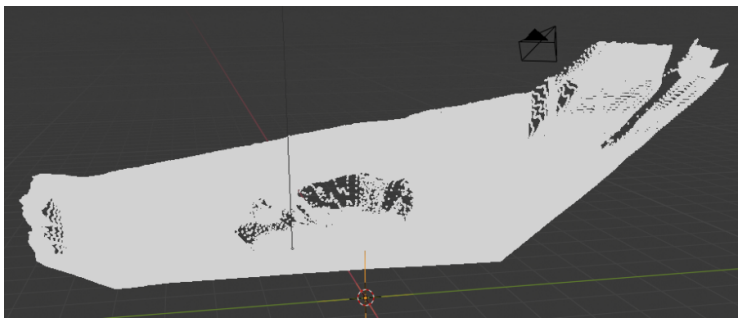


Figure 4.15: Dense point cloud generated from the depth map.

After the generation of the dense point cloud, we need to isolate the vehicle's points in order to perform object classification using the PointNet architecture. However, the highly dense point cloud constitutes a performance problem to current state of the art approaches and the majority of the architectures that process point cloud data are not trained with highly dense point clouds, which would imply the sub-sampling of the points in order to meet the targeted network's input point cloud size requirements.

Instead of spending an effort in overcoming such an obstacle, we decided to push forward and try the most recent version of Pseudo-LiDAR - Pseudo-LiDAR++, which already addresses this issue by combining the dense Pseudo-LiDAR point cloud with a sparse LiDAR point cloud.

4.3.3 Pseudo-LiDAR++

The Pseudo-LiDAR++ improves upon the previous work by leveraging a new depth estimation algorithm called SDNet [48] and a low cost LiDAR sensor.

The authors of Pseudo-LiDAR++ modified the original SDNet architecture, that only supported monocular images, for use with stereo images. The SDNet first creates a depth map from the input stereo image and uses it to generate a dense Pseudo-LiDAR point cloud, as shown in Figures 4.16 and 4.17, respectively.

Figure 4.16: Predicted depth map from the stereo image.

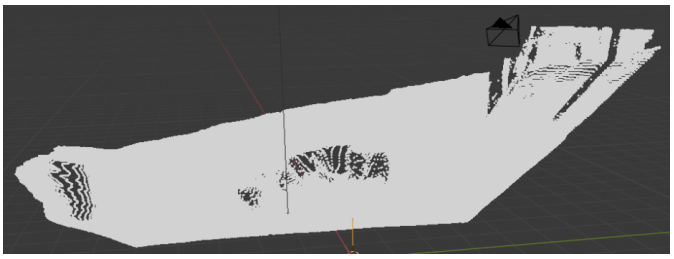


Figure 4.17: Generated dense Pseudo-LiDAR point loud from the created depth map.

After obtaining the refined dense point cloud, by using a lower resolution LiDAR point cloud to perform depth correction, a sparse point cloud is then generated by calculating the elevation angles according to the resolution of the targeted LiDAR specifications and slicing the dense point cloud accordingly.

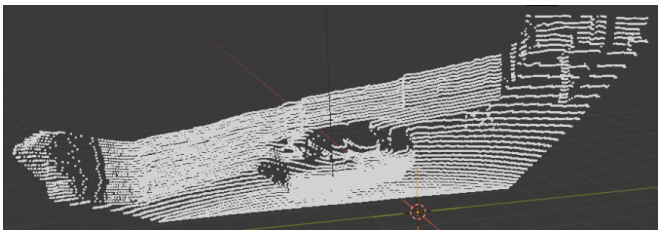


Figure 4.18: Final result of the Pseudo-LiDAR architecture.

Compared to the first iteration of Pseudo-LiDAR, Pseudo-LiDAR++ produces sparser and more accurate point clouds through the use of a low resolution and cost effective LiDAR sensor. A LiDAR sensor with low resolution does not capture a significant number of points for effective detection of objects of interest in the scene but, in conjunction with the generated dense Pseudo-LiDAR point cloud, it is possible to obtain a similar point cloud as the ones produced by a higher resolution LiDAR sensor, as shown in Figure 4.18.

Classification of the vehicle using the PointNet architecture

From the sparser point cloud obtained from Pseudo-LiDAR++, it is now possible to manually isolate the points belonging to the vehicle, shown in Figure 4.19, and perform 3D object classification using the PointNet architecture.

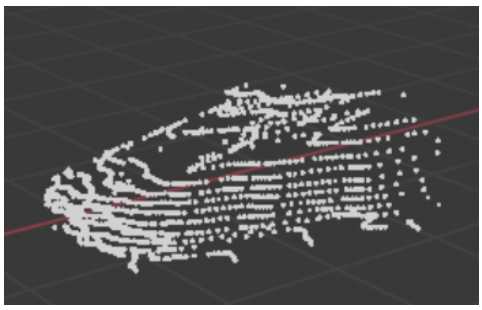


Figure 4.19: Isolated points of the target vehicle.

A visualization tool is provided by the PointNet project in order to see the shape of the input samples in several orientations.

The vehicle point cloud obtained from the sparser point cloud was classified by PointNet as belonging to the "chair" category. By analysing the Figure 4.20, and taking into account the results obtained on the Modelnet40 dataset, we can conclude that the incomplete nature of point clouds captured from LiDAR or Pseudo-LiDAR contributes to the wrong prediction made by the model, as it was trained on point clouds comprised of full representations of the correspondent objects.

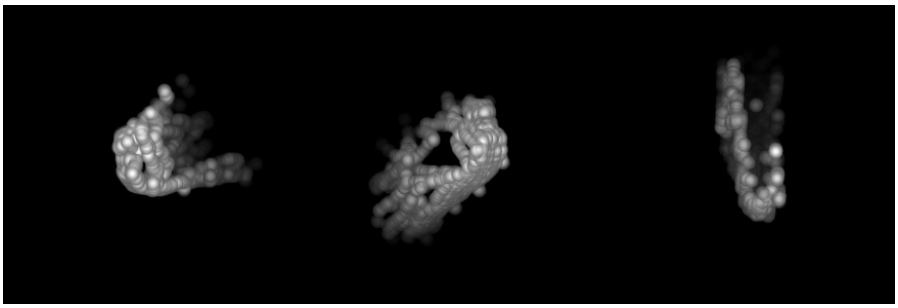


Figure 4.20: Visualization of the previous sample on the PointNet architecture.

Final Remarks

The main outcome produced by this collaborative work between author and four fourth year students is that a GTA V mod for generating stereo views has been successfully implemented. While the number of generated samples was clearly insufficient to assess the quality of the point clouds produced from the stereo depth maps, we believe that this mod enables a more in depth study.

From the results obtained through experiments on both image-based 3D object identification architectures, we can conclude that these are capable of adapting to less realistic datasets, as the ones produced within GTA V. The visualization of the results match the expected outcomes of the architectures, with Pseudo-LiDAR++ being the better approach to produce point clouds compatible with established 3D object identification algorithms, such as the ones based on PointNet, with similar point distribution and sparsity.

The experiment with the PointNet architecture reveals the necessity of training with incomplete object point clouds in order to obtain the desired classification results. Also, PointNet was used by many of the previously tested architectures in Section 4.2.3 with high effectiveness, which must be due to the training with the KITTI dataset and also from the fact that only 3 categories are being considered in the context of autonomous vehicles (car, pedestrian and cyclist). In contrast, the original PointNet model, used in this section, was trained to classify 40 different categories of point clouds with points uniformly sampled from the objects' surfaces.

Chapter 5

Discussion and Future Work

We started this Master's thesis with no prior knowledge about the Deep Learning approaches for 3D object identification, how these perform in the context of autonomous vehicles, and what kind of datasets are publicly available and commonly used for training and evaluating these architectures.

The main objective of this work was the develop software for generating synthetic datasets, in order to help with the training and testing of 3D object identification architectures in the context of autonomous vehicles.

5.1 Summary of Developed Work

We start our analysis of 3D object identification architectures by reviewing the first approaches to 3D object classification and segmentation on point clouds. These constitute the basis of future state of the art architectures in this field of study.

From the several representations used to extract features from the point cloud input data – multi-view, volumetric, mesh and point cloud – the latter became the most attractive representation with the introduction of the PointNet and PointNet++ architectures. These architectures no longer required input discretization, which could remove important features from the original data.

Given that the PointNet architectures only perform object classification and segmentation, we searched for methods specifically oriented towards the context of autonomous vehicles, being able to process an entire point cloud scene and to detect the three main object categories in such a context: pedestrian, car and cyclist. Guided by the goals of this thesis, we select a subset of the analysed architectures for evaluation.

After a thorough review of the state of the art, we conclude that the KITTI object detection dataset had become the standardized dataset for evaluating and comparing the performance between previous and newly released architectures. Based on this knowledge, we decide to follow this format for the generation of synthetic datasets, in order to be supported by the 3D object identification architectures and to easily compare the results with the state of the art, on the real-life KITTI dataset.

Following this decision, we implement the GTA V mod for generating synthetic datasets, using a state-based approach. It enables the manual or automatic creation of synthetic datasets. For each sample, every relevant information about the vehicles and pedestrians is collected, along with point-wise information for segmentation and mappings to the three image views.

Given that the dataset produced by the GTA V mod is not in the KITTI format, we proceed with the implementation of a Python project to convert the GTA V dataset format to KITTI's.

Having completed both projects, we generate a simple dataset of 50 samples, with no more than a car per scene and following the Easy difficulty level defined by KITTI. The output KITTI-formatted dataset could then be visualized using multiple scripts, in order to assess its quality.

With the synthetic dataset generated, we perform its evaluation on the selected architectures, using four different metrics for 2D, bird's eye view and 3D object detection, and orientation similarity. The performance results are finally summarized and analysed in order to draw conclusions about the quality of the generated dataset and the ability of the models, trained on real-life datasets, to adapt to fully synthetic ones.

All conclusions drawn in the entire work regarding performance must be deemed as preliminary, as these are based on a first, small and simple dataset. A bigger, more representative dataset would be required to confirm such conclusions.

5.2 Main Outcomes

We can divide the outcomes of this project into two types: the developed software solution and the results.

In regards to the developed software solution, the main projects consist of:

1. **GTA V mod for the synthetic dataset generation:** facilitates the generation of synthetic datasets, based on the GTA V environment, with automatic labeling and is capable of detecting cars and pedestrians. From the cooperation with the students of the Machine Learning course, it also became possible to capture stereo images for use in Pseudo-LiDAR architectures.
2. **Python project for the conversion of a GTA V dataset to the KITTI format:** given the output of the dataset generated by the GTA V mod, this project converts it into the KITTI format, following the options specified by the user. These options include the addition of noise, addition or removal of the intensity values, filtering of objects of interest by distance to the LiDAR, filtering by object category (vehicle or pedestrian) and generation of additional point clouds consisting of single vehicles or without background points.
3. **Python project for calculating the IoU of predicted bounding boxes:** this is used to calculate the IoU of the predicted 2D (bird's eye view) and 3D bounding boxes, and allows for the visualization of the overlapping region of each detected bounding box to draw conclusions from the Average Precision results.

In regards to the results, the outcomes are the following:

- The generated synthetic dataset is correctly converted into the KITTI format and is compatible with the state of the art 3D object identification architectures.
- The selected state of the art architectures are able to extract relevant features from the synthetic dataset, even though these were trained using a real-life dataset.
- The fact that the majority of the models published in the context of autonomous vehicles being trained on point clouds with intensity values may always contribute to lower performance results when evaluating on synthetic datasets, as the simulation of this point attribute is not a trivial process.
- The Pseudo-LiDAR++ architecture was able to generate a satisfactory point cloud, even with synthetic stereo images without rectification.
- Classification architectures, such as PointNet and PointNet++, need to be trained with partial point clouds to be able to correctly classify objects captured by a LiDAR sensor.

5.3 Limitations of this Work

There are two types of limitations when considering the developed work for generating synthetic datasets:

- Limitations due to the nature of using synthetic datasets on Deep Learning models trained on real-life datasets:
  - The environments provided by GTA V have less complexity than real-world environments, and the output does not reproduce the measurement errors and noise produced by real sensors.
  - Difficulty in reproducing certain data attributes, such as point intensity values.
  - Difficult to find the appropriate calibration matrix of the virtual camera to rectify the image views and properly project the points from the camera coordinate system to the pixel coordinate system.
- Limitations in the implementation of the software solution:
  - The 3D bounding boxes, returned internally by GTA V, are not perfectly fit to the objects of interest, resulting in lower performance results due to imperfect ground truth labels.
  - The generation of point clouds only supports relatively flat ground, resulting in an elliptic shaped point cloud if the character is on a slope.
  - The truncation, occlusion and alpha label fields are not calculated. These are important for filtering samples that are of moderate or hard difficulty level.

5.4 Discussion and Future Work

This thesis provides the foundations for studying 3D object identification in the context of autonomous driving, using synthetic datasets produced within GTA V.

Given the success in generating datasets based on GTA V, future work may focus on four directions:

1. Leverage the developed tool for generating bigger and more diverse datasets in order to obtain sounder conclusions.
2. Perform experiments for determining how small models can be such that 3D object identification is accurate, having in mind the reduction of hardware needs and associated cost.
3. Experiment with different LiDAR specifications, such as resolution, noise or intensity parameter, to determine its impact in the accuracy of 3D object identification.
4. Reflect on whether the now-standard metric, all based on the IoU, do make sense. It is interesting to note that Deep Learning architectures, which are known for its approximation purposes, are compared using rigid mathematical metrics.

Looking deeper into the future, there are still many interesting research directions that deserve being explored:

1. The introduction of time information, by processing past frames and not only the present frame. Further studies on the approaches used for 3D object identification can also to be conducted in order to address the fact that all of the reviewed state of the art architectures perform object identification solely based on the currently captured point cloud frame.
2. The consideration of prior knowledge. All investigated architectures have no prior knowledge except for being trained in training data from the same dataset. This means that there is no prior knowledge on the particular scene the LiDAR is observing.

   Provided that the vehicle has high-definition maps of its location, it should be possible to produce a point cloud from such a location. Such a point cloud, only containing background points can be compared with the point cloud captured by the LiDAR in order to differentiate foreground from background.

   This way, the performance of the region proposal algorithm, which is one of the bottlenecks of the current state of the art architectures that directly process point clouds, is significantly improved, as only foreground points from the point cloud are indeed processed.
3. Making datasets look real using filters, namely based on Generative Adversarial Networks. Such Deep Learning architectures are well known for producing realistic pictures, to a point that a Human cannot tell whether the produced images are real or synthetic. Such techniques could therefore be used to produced realistic synthetic datasets.

Bibliography

[1] Hossein Aboutalebi, Mohammad Javad Shafiee, Michelle Karg, Christian Scharfenberger, and Alexander Wong. Vulnerability under adversarial machine learning: Bias or variance? *arXiv preprint arXiv:2008.00138*, 2020.

[2] Anas Alhashimi. *Statistical Sensor Calibration Algorithms*. PhD thesis, Luleå University of Technology, 2018.

[3] Michael Barnard. Tesla and google disagree about lidar, which is right? `https://cleantechnica.com/2016/07/29/tesla-google-disagree-lidar-right/`.

[4] Alexander Blade. Script hook v. `http://www.dev-c.com/gtav/scripthookv/`.

[5] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016.

[6] Engin Bozkurt. Carla training data. `https://github.com/enginBozkurt/carla-training-data`.

[7] Xavier Santos Bruno Aguiar. Gta project mod. `https://github.com/gta-project/mod`.

[8] Jia-Ren Chang and Yong-Sheng Chen. Pyramid stereo matching network. *CoRR*, abs/1803.08669, 2018.

[9] X. Chen, K. Kundu, Y. Zhu, H. Ma, S. Fidler, and R. Urtasun. 3d object proposals for accurate object classdetection. *NIPS*, 2015.

[10] X. Chen, Z. Zhang, K. Kundu, H. Ma, S. Fidler, and R. Urtasun. Monocular 3d object detection for autonomous driving. *InCVPR*, 2016.

[11] Xiaozhi Chen, Kaustav Kundu, Yukun Zhu, Andrew G Berneshawi, Huimin Ma, Sanja Fidler, and Raquel Urtasun. 3d object proposals for accurate object class detection, 2015.

[12] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3d object detection network for autonomous driving. *CoRR*, abs/1611.07759, 2016.

[13] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. `https://carla.org`, 2017.

[14] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[15] Huan Fu, Mingming Gong, Chaohui Wang, Kayhan Batmanghelich, and Dacheng Tao. Deep ordinal regression network for monocular depth estimation. *CoRR*, abs/1806.02446, 2018.

[16] Ahmed Fawzy Gad. Evaluating deep learning models: The confusion matrix, accuracy, precision, and recall. `https://blog.paperspace.com/deep-learning-metrics-precision-recall-accuracy/`.

[17] Shanghua Gao, Ming-Ming Cheng, Kai Zhao, Xinyu Zhang, Ming-Hsuan Yang, and Philip H. S. Torr. Res2net: A new multi-scale backbone architecture. *CoRR*, abs/1904.01169, 2019.

[18] Alberto Garcia-Garcia, Sergio Orts-Escolano, Sergiu Oprea, Victor Villena-Martinez, and José García Rodríguez. A review on deep learning techniques applied to semantic segmentation. *CoRR*, abs/1704.06857, 2017.

[19] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.

[20] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Kitti object detection evaluation 2012.

[21] 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.

[22] Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015.

[23] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[24] Ian J. Goodfellow, Yoshua Bengio, and Aaron C. Courville. *Deep Learning*. Adaptive computation and machine learning. MIT Press, 2016.

[25] Kaiming He. Deep residual networks. `https://icml.cc/2016/tutorials/icml2016_tutorial_deep_residual_networks_kaiminghe.pdf`.

[26] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017.

[27] Andrew Hryniowski, Xiao Yu Wang, and Alexander Wong. Where does trust break down? a quantitative trust analysis of deep neural networks via trust matrix and conditional trust densities. *arXiv preprint arXiv:2009.14701*, 2020.

[28] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. *CoRR*, abs/1709.01507, 2017.

[29] ImageNet. Imagenet large scale visual recognition challenge. `http://www.image-net.`

org/.

[30] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[31] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks. *CoRR*, abs/1506.02025, 2015.

[32] Nikolaus Kriegeskorte. Deep neural networks: A new framework for modeling biological vision and brain information processing. *Annual Review of Vision Science*, 1(1):417–446, 2015. PMID: 28532370.

[33] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[34] Jason Ku, Melissa Mozifian, Jungwook Lee, Ali Harakeh, and Steven Lake Waslander. Joint 3d proposal generation and object detection from view aggregation. *CoRR*, abs/1712.02294, 2017.

[35] Alex H. Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. *CoRR*, abs/1812.05784, 2018.

[36] Scikit Learn. Scikit learn. `https://scikit-learn.org`.

[37] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[38] Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. Object recognition with gradient-based learning. In *Shape, contour and grouping in computer vision*, pages 319–345. Springer, 1999.

[39] Tsung-Yi Lin, Piotr Dollár, Ross B. Girshick, Kaiming He, Bharath Hariharan, and Serge J. Belongie. Feature pyramid networks for object detection. *CoRR*, abs/1612.03144, 2016.

[40] Zhong Qiu Lin, Mohammad Javad Shafiee, Stanislav Bochkarev, Michael St. Jules, Xiaoyu Wang, and Alexander Wong. Do explanations reflect decisions? A machine-centric strategy to quantify the performance of explainability algorithms. *CoRR*, abs/1910.07387, 2019.

[41] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015.

[42] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *CoRR*, abs/1411.4038, 2014.

[43] Wenjie Luo, Yujia Li, Raquel Urtasun, and Richard S. Zemel. Understanding the effective receptive field in deep convolutional neural networks. *CoRR*, abs/1701.04128, 2017.

[44] Sancho McCann. Average precision. `https://sanchom.wordpress.com/tag/`

average-precision/.

[45] Shervin Minaee, Yuri Boykov, Fatih Porikli, Antonio Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. Image segmentation using deep learning: A survey, 2020.

[46] NHTSA. 2018 fatal motor vehicle crashes: Overview. https://www.nhtsa.gov/press-releases/roadway-fatalities-2018-fars.

[47] NVIDIA. Object detection data extension.

[48] Matthias Ochs, Adrian Kretz, and Rudolf Mester. Sdnet: Semantically guided depth estimation network. *CoRR*, abs/1907.10659, 2019.

[49] Indian Institute of Technology Kharagpur. Construction of 3d orthogonal cover. http://cse.iitkgp.ac.in/~pb/research/3dpoly/3dpoly.html.

[50] University of Washington. University of washington repository. https://sites.stat.washington.edu/.

[51] Charles Ruizhongtai Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J. Guibas. Frustum pointnets for 3d object detection from RGB-D data. *CoRR*, abs/1711.08488, 2017.

[52] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *CoRR*, abs/1612.00593, 2016.

[53] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *CoRR*, abs/1706.02413, 2017.

[54] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[55] Joseph Redmon and Ali Farhadi. YOLO9000: better, faster, stronger. *CoRR*, abs/1612.08242, 2016.

[56] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[57] Seyed Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian D. Reid, and Silvio Savarese. Generalized intersection over union: A metric and A loss for bounding box regression. *CoRR*, abs/1902.09630, 2019.

[58] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li. Imagenet large scale visual recognition challenge. *CoRR*, abs/1409.0575, 2014.

[59] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[60] Shaoshuai Shi, Chaoxu Guo, Li Jiang, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. Pv-rcnn: Point-voxel feature set abstraction for 3d object detection. *CoRR*, abs/1912.13192, 2019.

[61] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. Pointrcnn: 3d object proposal generation and detection from point cloud. *CoRR*, abs/1812.04244, 2018.

[62] Martin Simon, Stefan Milz, Karl Amende, and Horst-Michael Gross. Complex-yolo: Real-time 3d object detection on point clouds. *CoRR*, abs/1803.06199, 2018.

[63] Andrea Simonelli, Samuel Rota Bulo, Lorenzo Porzi, Manuel López-Antequera, and Peter Kontschieder. Disentangling monocular 3d object detection, 2019.

[64] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[65] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[66] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik G. Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. *CoRR*, abs/1505.00880, 2015.

[67] Jorge Sánchez, Thomas Mensink, and Jakob Verbeek. Image classification with the fisher vector: Theory and practice. *International Journal of Computer Vision*, 105, 12 2013.

[68] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *CoRR*, abs/1905.11946, 2019.

[69] OpenPCDet Development Team. Openpcdet: An open-source toolbox for 3d object detection from point clouds. `https://github.com/open-mmlab/OpenPCDet`, 2020.

[70] Princeton University. Princeton modelnet. `https://modelnet.cs.princeton.edu/`.

[71] Stanford University. The stanford 3d scanning repository. `http://graphics.stanford.edu/data/3Dscanrep/`.

[72] Yan Wang, Wei-Lun Chao, Divyansh Garg, Bharath Hariharan, Mark Campbell, and Kilian Q. Weinberger. Pseudo-lidar from visual depth estimation: Bridging the gap in 3d object detection for autonomous driving. *CoRR*, abs/1812.07179, 2018.

[73] Alexander Wong, Xiao Yu Wang, and Andrew Hryniowski. How much can we really trust you? towards simple, interpretable trust quantification metrics for deep neural networks. *arXiv preprint arXiv:2009.05835*, 2020.

[74] Daniel E. Worrall, Stephan J. Garbin, and Gabriel J. Brostow. Harmonic networks: Deep translation and rotation equivariance. *CoRR*, abs/1612.04642, 2016.

[75] Yurong You, Yan Wang, Wei-Lun Chao, Divyansh Garg, Geoff Pleiss, Bharath Hariharan, Mark Campbell, and Kilian Q. Weinberger. Pseudo-lidar++: Accurate depth for 3d object detection in autonomous driving. *CoRR*, abs/1906.06310, 2019.

[76] Matthew D. Zeiler, Dilip Krishnan, Graham W. Taylor, and R. Fergus. Deconvolutional networks. *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2528–2535, 2010.

[77] Zhong-Qiu Zhao, Peng Zheng, Shou-tao Xu, and Xindong Wu. Object detection with deep learning: A review. *CoRR*, abs/1807.05511, 2018.

[78] Dingfu Zhou, Jin Fang, Xibin Song, Chenye Guan, Junbo Yin, Yuchao Dai, and Ruigang Yang. Iou loss for 2d/3d object detection. *CoRR*, abs/1908.03851, 2019.

[79] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. *CoRR*, abs/1711.06396, 2017.