Università degli Studi di Padova


University of Padova

Department of Information Engineering

Master Degree in
ICT for Internet & Multimedia

# Point cloud object detection and classification for railway applications

*Supervisor:*
Prof. Federica Battisti

*Candidate:*
Michael Neri
1232224

A.Y. 2020/2021

**Abstract**

Accurate detection and classification of objects in 3D point clouds is a central problem in autonomous navigation and augmented/virtual reality scenarios.

It consists in a combination of regression (center, dimensions and rotation of a 3D bounding box) and classification (positive and negative labels) tasks, fine-tuning predictions and overall accuracy of the detection system.

In railway applications, one of the most critical difficulty is the scarcity of annotated datasets in order to detect specific landmarks. **VOLIERA** and **RAILGAP** are the first projects that implement detection technologies based on artificial intelligence in the railway environment, combined with GNSS data.

We are going to accurately describe a Deep Learning strategy using VoxelNet model for 3D object detection. Such network is adaptable to different environments by modifying initial hyper parameters of the model and by learning different data features.

Experiments on automotive datasets, which can be used as training data in autonomous driving tasks, yield in poor performances for railway applications. A synthetic dataset is generated in order to train VoxelNet to *ad-hoc* environments.

Furthermore, analysis of metrics and results of the network show that our network learns an effective representation of railway landmarks, using only raw LiDAR point clouds, leading to encouraging results and possible future implementations in this research field.

## Abstract

Il rilevamento e la classificazione accurata di oggetti in nuvole di punti 3D è un problema centrale nell' ambito della guida autonoma e in scenari di realtà aumentata/virtuale.

Consiste nella combinazione di algoritmi di regressione (centro, dimensione e rotazione di un parallelepipedo che racchiude l'oggetto) e classificazione (discriminando se genuino o falso positivo), ottimizando le previsioni e l'accuratezza del sistema di rilevamento.

In applicazioni ferroviarie, una delle difficoltà più critiche è la scarsità di dataset dettagliati per riconoscere specifici punti di riferimento. **VOLIERA** e **RAILGAP** sono i primi progetti che implementano tecnologie di rilevamento basate su intelligenza artificiale nell'ambiente ferroviario, combinate con l'informazione prodotta dal GNSS.

Descriveremo accuratamente una strategia basata sul Deep Learning, utilizzando la rete neurale VoxelNet per il rilevamento di oggetti in ambienti tri-dimensionali. Tale rete è adattabile a differenti scenari, modificandone i parametri iniziali e, perciò, estraendo diverse caratteristiche dai dati.

Sono stati effettuati diversi esperimenti su possibili dataset, i quali possono essere usati come dataset di allenamento per la guida autonoma in ambito automobilistico, portando scarsi risultati per l'ambiente ferroviario. Per questo motivo, è stato generato un dataset sintetico in modo da allenare correttamente la rete VoxelNet per scenari *ad-hoc*.

Inoltre, analisi sulle metriche e risultati della rete mostrano che VoxelNet riesce ad imparare una propria rappresentazione dei punti di riferimento ferroviari, usando solo nuvole di punti 3D, portando a risultati incoraggianti e possibili future migliorie in questo campo di ricerca.

# Contents


**1 Introduction** **1**

1.1 Problem Statement . . . . . 1

1.1.1 Related Works . . . . . 1

1.1.2 List of Sensors . . . . . 4

1.1.3 LiDAR Further Insights . . . . . 5

1.1.4 RAILGAP Project . . . . . 7

1.1.5 VOLIERA Project . . . . . 9

1.1.6 Contributions . . . . . 12

**2 Dataset** **15**

2.1 KITTI360 . . . . . 15

2.2 RailSem19 . . . . . 17

2.3 KITTI . . . . . 20

2.4 Syntethic Dataset . . . . . 23

2.4.1 Matlab™ - LiDAR data simulation . . . . . 24

2.4.2 Unreal Engine 4™ - Simulating railway environment . . . . 25

2.4.3 Dataset Generation . . . . . 28

**3 VoxelNet - Deep Learning Architecture** **31**

3.1 Deep Learning Strategy Selection . . . . . 31

3.2 VoxelNet architecture . . . . . 32

3.2.1 Feature Learning Network . . . . . 33

3.2.2 Convolutional Middle Layers . . . . . 36

3.2.3 Region Proposal Network . . . . . 37

3.3 Loss Function . . . . . 37

**4 Training Details** **43**

4.1 Network Details . . . . . 43

4.1.1 KITTI . . . . . 43

4.1.2 Syntethic Dataset . . . . . 49
4.2 Data Augmentation . . . . . 50
4.3 Code Insights . . . . . 51

**5 Evaluation of VoxelNet Performances** **53**
5.1 Metrics . . . . . 53
5.2 Evaluation on KITTI Dataset . . . . . 54
5.3 Evaluation on Synthetic Dataset . . . . . 58

**6 Final Considerations** **61**
6.1 Future Work . . . . . 61
6.2 Conclusion . . . . . 64

**References** **71**

# Chapter 1

# Introduction

## 1.1 Problem Statement

An autonomous vehicle requires a precise perception of the surrounding environment in order to operate and select correct decision.

Perception is developed thanks to Machine Learning and Deep Learning algorithms by transforming sensor data into semantic information.

Object detection is one of the most critical tasks in an autonomous vehicle. It consist of a combination of regression and classification tasks with the aim to recognize and classify relevant objects from images, videos and point clouds. The problem is relevant for other various applications such as video surveillance monitoring for security purposes, automatic analysis of medical data, etc..

The main objective of this thesis is to identify major landmarks of the railway environment from raw point clouds in order to define their relative location.

### 1.1.1 Related Works

Working with images and videos, inspecting consecutive frames permits to extract 2D features in order to localize other agents and landmarks on the field of view of the autonomous vehicle.

Techniques of this approach predict and extrapolate 2D bounding box on the image plane to 3D through re-projection constraint or bounding box regression using neural networks [3] [4] [5].

In [3], **SVM** (*Support Vector Machine*) and **CNN** (*Convolutional Neural Networks*) are combined in order to place 3D bounding boxes on the ground-plane and scoring them via simple and efficiently computable image features.

**Figure 1.1:** 3D Object detection task in the automotive environment from CIA-SSD [15] paper

.

They exploited pre-trained VGG16 model - trained on ImageNet [6] dataset - and fed images of the KITTI [20] dataset for generating predictions in the 3D space. VGG16 [2] is a Convolutional Network for detection and classification for 2D images, displayed in Fig.1.2. A specific branch of the network is responsible for proposing most relevant regions for objects, placing pre-defined anchors.

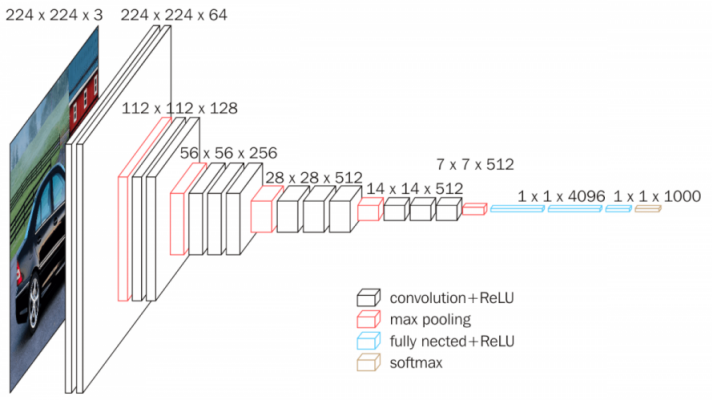



**Figure 1.2:** VGG16 deep architecture.

Finally, they used a multi-task loss to jointly predict category labels, bounding box offsets, and object orientation - obtaining high value of accuracy.

Instead, [4] uses images constraints for placing 3D bounding box on KITTI dataset images. The constraint that the 3D bounding box fits tightly into 2D detection window requires that each side of the 2D bounding box to be touched by the projection of at least one of the 3D box corners.

Without depth estimation, accuracy and localization performances are limited. A possible research topic can be the development of CNN which simulates

depth channel in order to use mono **RGB** (*Red-Green-Blue*) images.

With the third dimension, we can deploy different strategies that take into account point clouds for extracting semantic information for autonomous driving. There are three possible methodologies for point cloud processing:

- **Projection**: Projection of point clouds on image plane in order to fit 3D bounding box to objects. It results in loss of information during the projection process - re-projection error - but it can be improved by involving accurate and fine-tuned parameters of cameras and sensors [7] [8];
- **Voxelization**: Generation of a voxel structure from point clouds and then a **FCN** (*Fully Convolutional Network*) is developed for processing. Volumetric representation is still sparse but spatial information is preserved during encoding phase. At the end, 4-D feature tensors are fed to **RPN** (*Region Proposal Network*) which improves both processing time and accuracy of the network [18];
- **PointNet**: Raw point clouds are elaborated by the network using a feed-forward network without any sort of pre-encoding. Feeding entire point cloud as input requests high computational preconditions and difficulties to define region proposals [9] [10] [11].

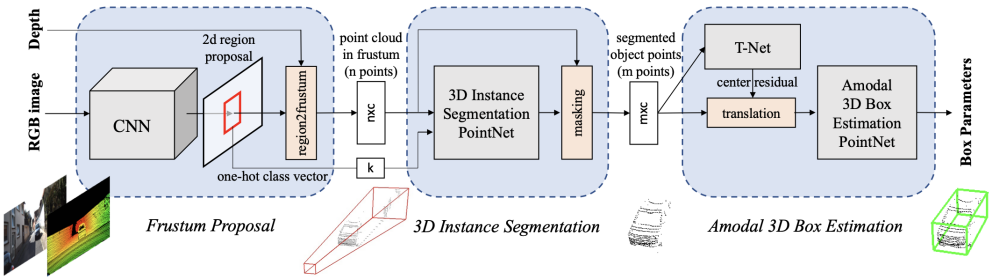



**Figure 1.3:** Frustum PointNet architecture.

State-of-the-art detectors combine the aforementioned solutions. Even if elaborating data from sensor fusion can be challenging because of calibration requirements, hybrid methods reach high level of accuracy and precision for 3D object detection. Frustum Pointnet [12] is an example of fusing RGB-D images with point clouds by generating region proposal w.r.t. mono RGB frames, obtaining remarkable results.

Moreover, existing 3D object detectors often treat object localization and category classification as separate tasks, so the localization accuracy and classifi-

cation confidence may not be coherent. For this reason, we can characterize two different types of detectors:

- **Single-Stage Detectors**: features are learned mostly from pre-defined anchors. Thus. a priori knowledge is required in order to obtain successful bounding box and classification predictions. Given their high efficiency, single-stage detectors have great potential for real-time applications;
- **Two-Stage Detectors**: features are extracted from the region proposal generated by the first-stage backbone and predict the **IoUs** (*Intersection Over Union*) between the regressed bounding boxes and ground truth boxes in the second stage to refine the confidence predictions.

An example of state-of-the-art single-stage detector is **CIA-SSD** [15] (*Confident IoU-Aware Single-Stage Object Detector*). First, the authors encode the point cloud using a sparse convolutional network called SPConvNet, followed by a spatial-semantic feature aggregation module for robust feature extraction. Then, thanks to an attentional fusion model, they are able to realize object classification and localization by applying a multi-task head.

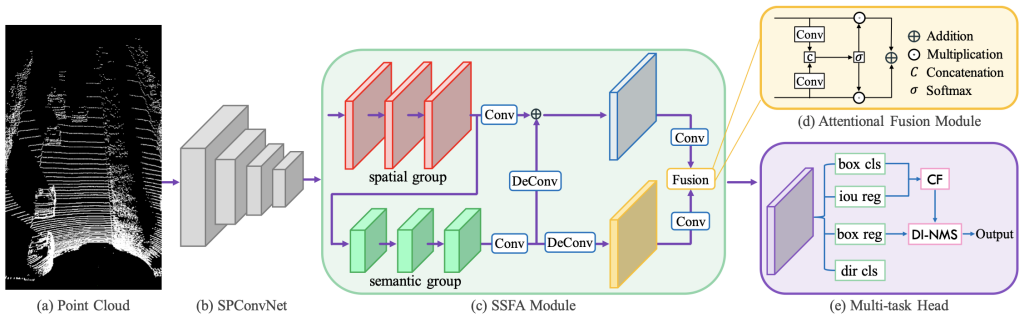



**Figure 1.4:** CIA-SSD architecture.

We can inspect the structure of the network in Fig.1.4.

### 1.1.2 List of Sensors

The main objective of the thesis is to detect relevant objects from the train surrounding by inspecting raw point clouds. Thanks to the landmarks detection, we are able to evaluate relative and absolute position of the train with high availability and reliability.

In order to record the railway environment, some sensors are mounted on the train in different locations and/or orientations. Thus, a registration process

is needed to process samples of each sensor in the same system reference. The algorithm is discussed with point clouds and depth maps in the next section (Sec. 1.1.3).

The available sensors are:

- **LiDAR** (*Light Detection and Ranging*) sensor which permits to obtain a point cloud representation of the surrounding environment of the train. Points are characterized as 4-D vectors $[x, y, z, w]$ which contain the 3D location and reflectivity of each point of the cloud. In our case, our sensor is the LIVOX™ Horizon which is shown in Fig.1.5;
- **Stereo camera** permits to obtain an **Intensity** map and a **Depth Map** (information relating to the distance of the surfaces of scene objects) of a specific **FOV** (*Field Of View*);
- GNSS combined with a **IMU** (*Inertial Measurement Unit*), which permits to have a priori knowledge of system evolution.

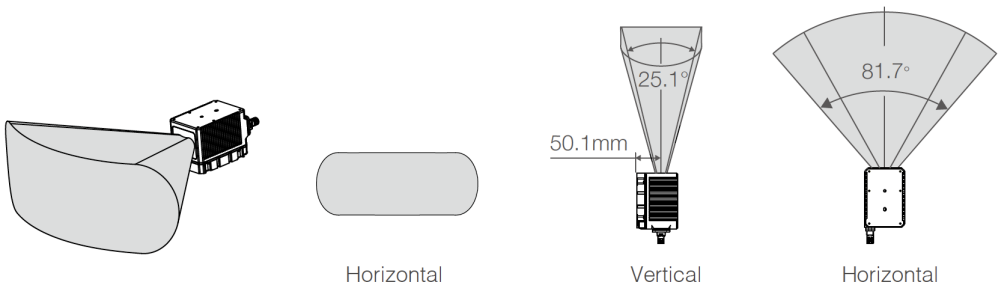



**Figure 1.5:** LIVOX™ Horizon sensor with its functional characteristics.

### 1.1.3 LiDAR Further Insights

One of the sensor that has captured the attention of the industry for several years now (automotive, agriculture etc..) is the LiDAR. It can be used in various field and application like photogrammetry or mapping.

The range measurement is equivalent to measuring the round-trip delay of light waves to the target. This can be achieved by modulating the intensity, phase, and/or frequency of the waveform of the transmitted light and measuring the time required for that modulation pattern to come back at the receiver.

In general, lasers are the preferred source of light because of their narrow spectral and beam properties; furthermore, phase and frequency-modulation (PM

| Pros | Cons |
|---|---|
| Can be used day and night | Very large dataset that can be difficult to register or to post-process |
| It is cheaper w.r.t. other sensors | Can be ineffective during heavy rain or low hanging clouds |
| Surface data can have higher sample density | No international protocol for data collection and analysis |
| Data can be collected quickly and with high accuracy | Can offer degraded performance on high sun angles and reflections |
| | Eye safety due to usage of laser source |

**Table 1.1:** Pros and cons of LiDAR sensor.

and FM) LiDARs require the coherence of the laser light. Lasers with wavelengths of 905, 1300, or 1550 nm, which are near the three established telecommunications windows, are commonly used in LiDAR applications.

In our case, LiDAR point clouds are fused with RGB-D images using depth maps which are image channels that contain information related to the distance of the surfaces of scene objects from a fixed viewpoint.

In order to mix point clouds and depth maps, we need to transform the latter into a point cloud thanks to a trivial linear transformation:

$$\begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix} = Q \cdot \begin{bmatrix} u \\ v \\ d \\ 1 \end{bmatrix} \tag{1.1}$$

where

- $Q$ is the mapping matrix provided by the camera itself which is a $4 \times 4$ matrix;
- $(x', y', z', w')$ are the final points in a point cloud structure;
- $(u, v, d, 1)$ are tuples obtained from the depth map where $(u, v)$ are the pixels of the image, $d$ is the disparity value and 1 is used for simplifying computational complexity (*homogeneous coordinates*).

Finally, 3D position of the point $(x, y, z)$ with image coordinates $(u, v)$ can be computed thanks to the following relation:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \frac{1}{w} \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} \tag{1.2}$$

In words, we divide the aforementioned vector with the reflectivity of each point.

Since **LiDAR** and **Stereo Camera** are in different position (but horizontally aligned), we need to adjust the orientation of a sensor using the other one as a reference.

After exploiting some analytical relations, we obtain:

$$tan(\gamma_1) = \frac{d_2 cos(\alpha_2) sin(\gamma_2) + \Delta y}{d_2 cos(\alpha_2) cos(\gamma_2) + \Delta x} \tag{1.3}$$

$$tan(\alpha_1) = \frac{(\Delta z + d_2 sin(\alpha_2)) cos(\gamma_1)}{d_2 cos(\alpha_2) cos(\gamma_2) + \Delta x} \tag{1.4}$$

where

- $(\Delta x, \Delta y, \Delta z)$ are the distances (or displacements) between the two sensors in all the directions;
- $(d_1, \alpha_1, \gamma_1)$ are, respectively, the measured distance, latitude and longitude w.r.t. sensor 1 to a specific object (landmark);
- $(d_2, \alpha_2, \gamma_2)$ are, respectively, the measured distance, latitude and longitude w.r.t. sensor 2 to a specific object (landmark);

Thanks to previous equations, we are able to merge the two point clouds in order to obtain an augmented point cloud which will be fed to different neural networks for next tasks.

We can inspect in Fig.1.6 and Fig.1.7 the lateral and top view of the sensors for the previous equations.

### 1.1.4 RAILGAP Project

The main objective of **RAILGAP** European funded project (*EU H2020 – GSA*) is to propose a high accuracy and integrity ground truth and track-side digital map methodologies and related tool-set development to provide an accurate and

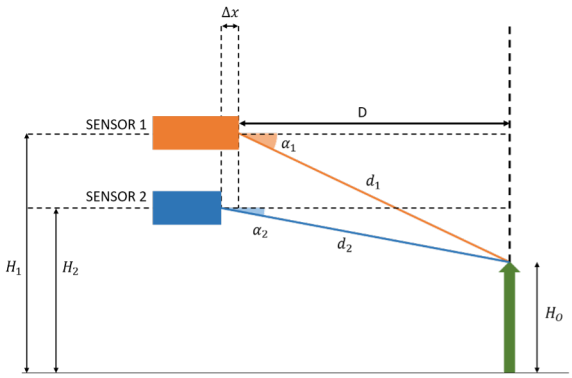



**Figure 1.6:** Sensor Geometry from lateral view.

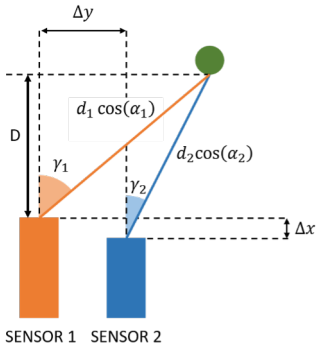



**Figure 1.7:** Sensor Geometry from top view.

reliable metric for evaluating the train satellite-based positioning accuracy and its confidence intervals.

The outcomes will address two show stoppers: lack of high-quality data with ground truth (needed for developing new navigation systems) and a modernized process for mapping existing train tracks cost-effectively, by deriving mapping information directly from trains in commercial operation.

Ground truth can be described as a set of georeferenced data with sufficient accuracy, availability and reliability to be considered a stable and true reference, suitable for the purpose of comparison and validation of other data sources in the railway domain according to established requirements.

More specifically, one of the key objectives of **RAILGAP** project is the development of **GTP** (*Ground Truth for the Train Position*) and a **GTO** (*Ground Truth for the Odometry*).

Each ground truth has to meet the following key requirements:

- It does not require installation of equipment on the signalling track side or any modification to the exiting signal track side;

- The track where the train runs is autonomously identified without considering any exchange of information from the track side subsystem to identify it;

- It relies on measured data collected by using trains in commercial service.

The missing piece is a methodology to collect and aggregate the data without operational overheads or labour, at minimal cost in hardware while removing any need for track side infrastructure.

RAILGAP addresses these challenges with a method based on commercial trains collecting massive amounts of data. This enables characterizing even the most challenging railway environments.

### 1.1.5 VOLIERA Project


**Figure 1.8: VOLIERA** logo.

**VOLIERA** is an **ESA** (*European Space Agency*) funded project (*ESA – NAVISP-EL2-009*) which aims to introduce detection technologies based on videos and 3D data in railway applications, combined with **GNSS** (*Global Navigation Satellite System*) data for the very first time in the state-of-the-art.

It consists in three main tasks:

- **Track Classification**: it aims to count the number of railtracks and then determine the one the train is following. First of all, the depth map is converted to the corresponding point cloud, then a registration step is performed to represent the depth map-based point cloud and the original point cloud in the same reference system.

  After registration, the two point clouds are merged to obtain an augmented point cloud which is used as input to the track discrimination algorithm.

This last exploits the information provided by GNSS and IMU (i.e. position and position estimate confidence interval) to speed up and simplify the track identification task.

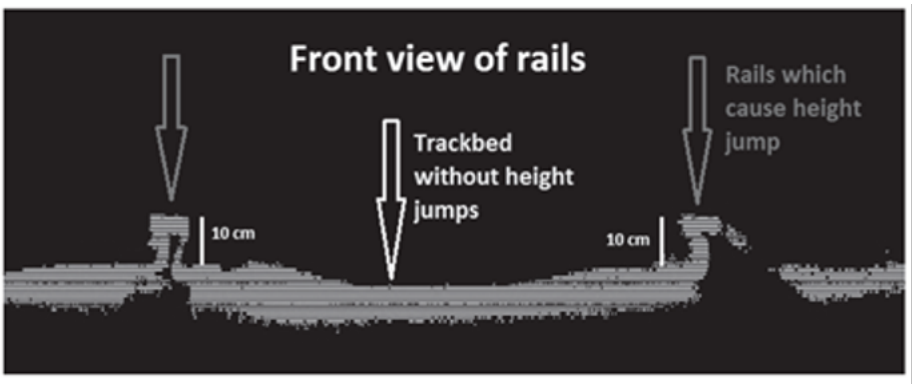



**Figure 1.9:** Detecting railways using only raw point clouds.

More in details, given the area in which the train is located, the information about how many tracks are present can be obtained, thus reducing the complexity of the track discrimination algorithm;

- **Relative and Absolute Positioning**: in order to compute the absolute position of the train, the available georeferenced landmarks are exploited. More in details, well established techniques for object detection and recognition are based on deep learning algorithms which take as input images of the surrounding scene. For this reason, when the intensity image is available, we use it to detect the visible landmarks.

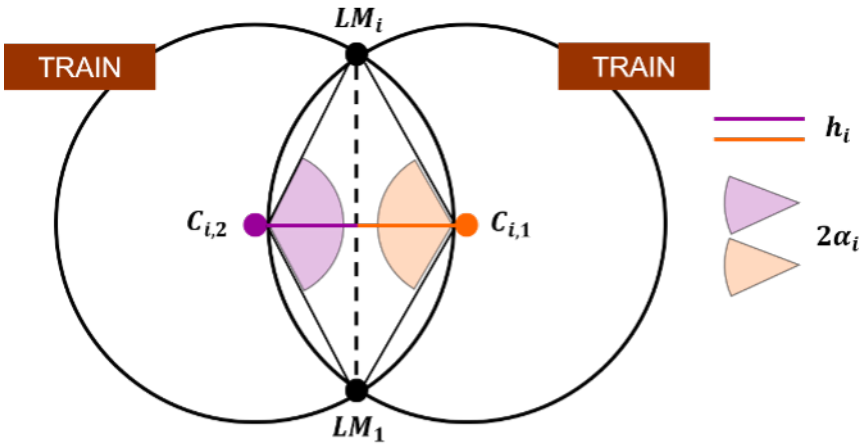



**Figure 1.10:** Computation of relative position with more than two detected landmarks.

The object detection algorithms output a bounding box which contains the desired objects, whose depth in relative coordinates can thus be extracted by

the corresponding depth map. Thanks to the registration process, moreover, the parameters needed for converting the 2D boxes identified in the intensity images to the corresponding 3D boxes in the point cloud reference system are available. Once the 3D box is obtained, the points which fall in it are extracted from the point cloud. The 2D and 3D bounding boxes, together with the corresponding depth map and point cloud portions are processed by the absolute positioning algorithm.

In order to exploit the landmark position, the absolute positioning algorithm has to match the landmarks detected in the acquired visual information (i.e. using their relative position) with the ones of the database. To do so, the information provided by IMU and GNSS (i.e. position and confidence interval) can be exploited to speed up the task. More specifically, the database search procedure is restricted to the landmarks falling in the area corresponding to the confidence interval provided by IMU and GNSS.

Differently, when the information provided by the stereo camera, i.e. the intensity image and the depth map, are lacking, the point cloud is used as input for a deep learning algorithm in order to detect the visible landmarks and their relative location. Once the 3D bounding box is obtained, the points which fall in it are extracted from the point cloud. The 3D bounding boxes, together with the corresponding point cloud portions are processed by the absolute positioning algorithm.

- **Pose Update**: to estimate the train pose update a late fusion is performed between the pose update provided by the visual odometry algorithm which exploits the depth map and the intensity image, and one obtained through the odometry algorithm based on the point cloud.

  The reason why a late fusion approach has been chosen is that the computational complexity of the point cloud-based odometry dramatically increases when a dense point cloud is provided as input. In addition, due to the different range of the sensors, the density of points in the augmented point cloud would not be uniform thus impairing the algorithm performances [14].

We can inspect project flow chart in Fig.1.12.

As we can clearly see from descriptions of aforementioned projects, they have similar structure and objectives, leading to a common solution for detecting and registering railway landmarks.

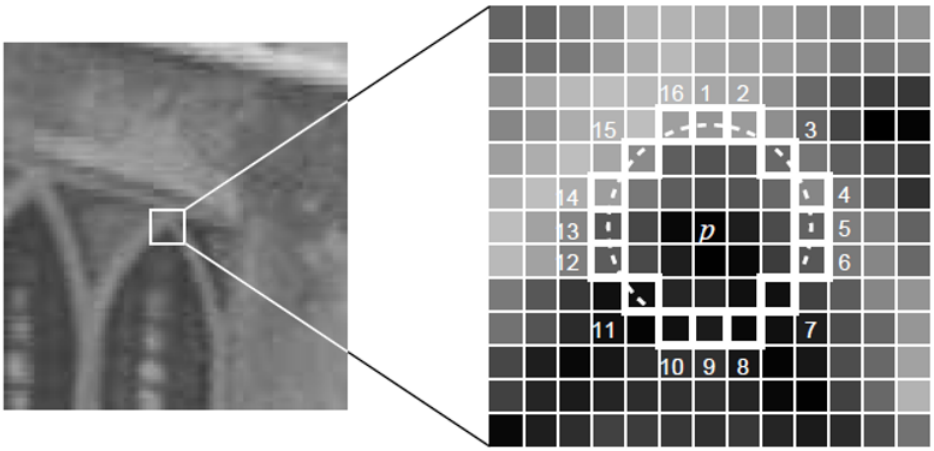



**Figure 1.11:** Rosten & Drummond [13] algorithm for fast feature detection on images.

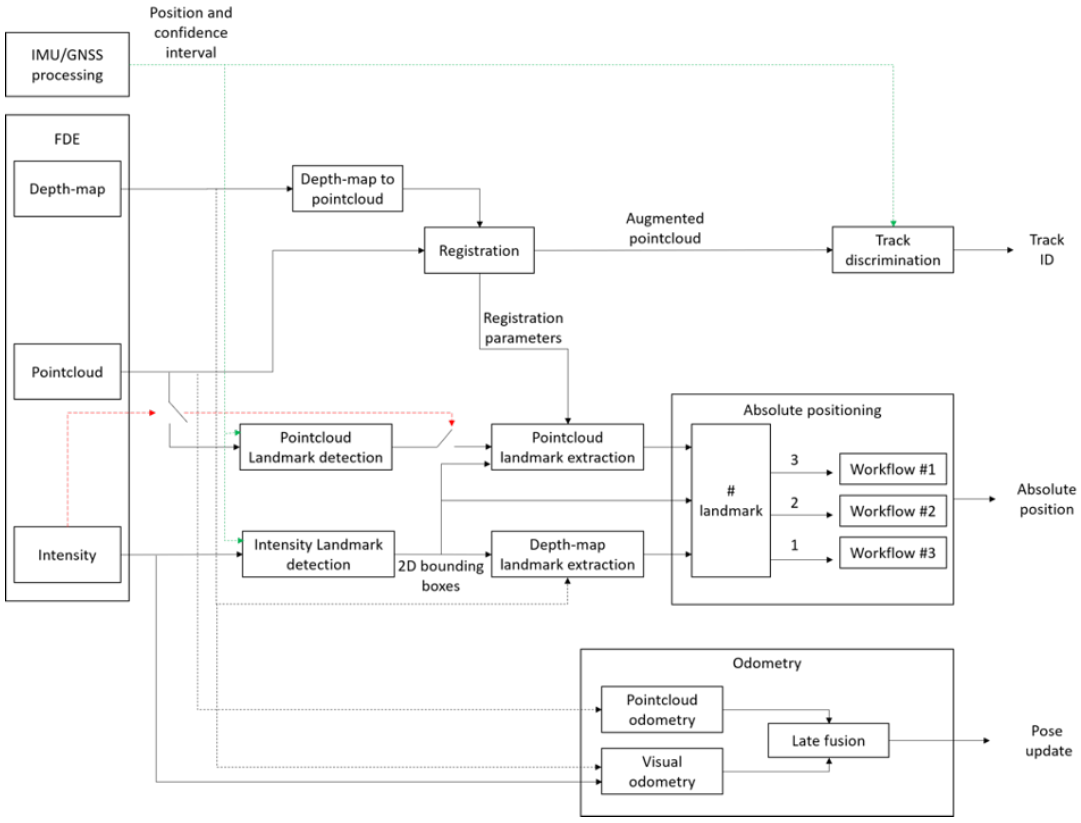



**Figure 1.12:** Flow chart of the **VOLIERA** project.

### 1.1.6 Contributions

In this thesis, we are going to illustrate how 3D object detection in **Relative and Absolute Positioning** task is performed using only LiDAR data thanks to Deep Learning strategies such as the single stage detector VoxelNet (Chapter 3).

Having a solution which works on LiDAR permits to take into account situ-

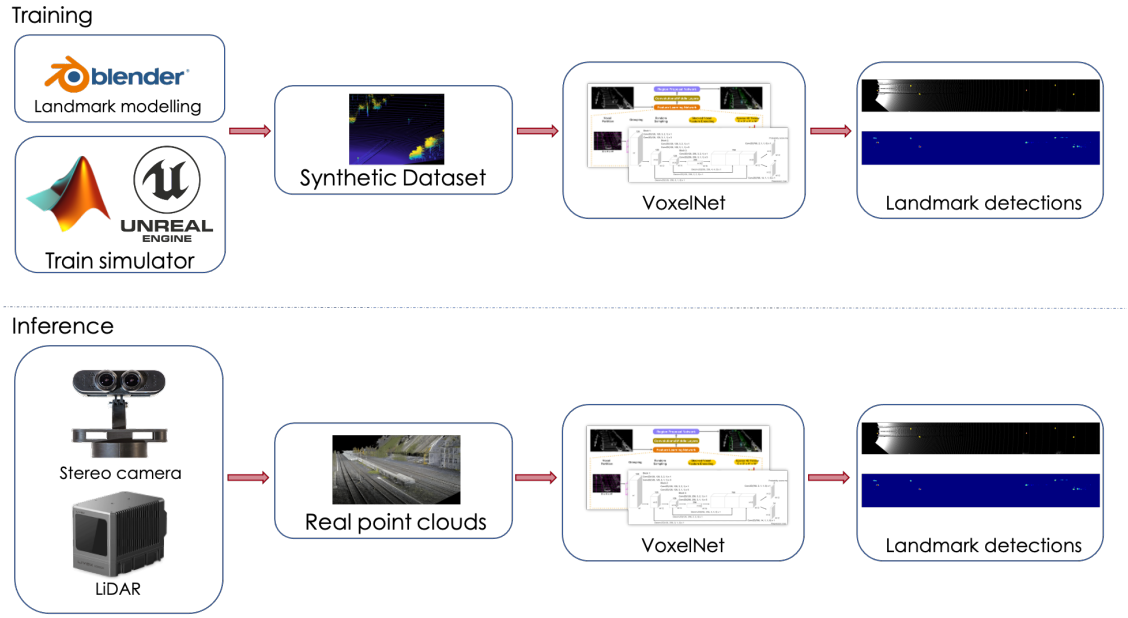



**Figure 1.13:** Proposed framework.

ations where images from RGB cameras are missing, increasing robustness and accuracy of the next workflows. Moreover, VoxelNet is able to:

- Handle point clouds with different number of points. Even if a more dense point cloud is fed to the network which was trained with different sized point clouds, functionalities and proprieties are not affected hereby;
- Preserve spatial information of data. Voxelization creates a grid structure of the point cloud and then encodes each voxel into a 4-D tensor called *feature tensor*;
- Reach high accuracy of detection with an inference time in the order of hundred of milliseconds;
- Classify different type of objects by adding pre-defined anchors.

The training of the chosen neural network requires a huge amount of data and time in order to generalize and reach successful percentage of detection accuracy and precision.

Therefore, a synthetic dataset is created by developing a virtual railway scene using 3D graphics engine Unreal Engine 4™ and simulating a LiDAR sensor using Matlab™ with the Automated Driving Toolbox (Chapter 2).

Since simulation models also railtracks and their structure in the Italian railways, it is possible to use the custom dataset also for track identification tasks.

An initial configuration is required by VoxelNet to start learning.

We deeply discuss all the possible arrangements in Chapter 4, describing also hyper parameters of training loop.

Network performances are evaluated on KITTI dataset in order to understand if the network is suitable for 3D object detection in wide open area.

Since automotive scenarios are different w.r.t. railway ones, a fine-tuning phase is mandatory in order to fit correctly the network.

We analyze evaluation results from manufactured dataset, reviewing output of the network and conducting some ablation studies in Chapter 5.

The proposed framework is in Fig.1.13.

In conclusion, we discuss some possible implementation for increasing accuracy and refining predictions in Chapter 6.

# Chapter 2

# Dataset

In this section, we are going to analyze some datasets for the autonomous driving task. **KITTI360** [21] and **RailSem19** [22] are the newest datasets for, respectively, automotive and railway applications. Unfortunately, they provide only RGB frames of the environment which permits only 2D object detection without an accurate depth estimation.

Nevertheless, **KITTI** [20] provides annotated automotive point clouds for deep learning strategies in the autonomous driving field. It is possible then to choose and validate a possible deep network to train for object detection in the rail context.

## 2.1 KITTI360

**KITTI360** [21] is a large-scale dataset that contains rich sensory information with full annotations. Jun Xie et al. recorded several suburbs of Karlsruhe, in Germany, obtaining over 320k images and 100k laser scans in a driving distance of 73.7km.

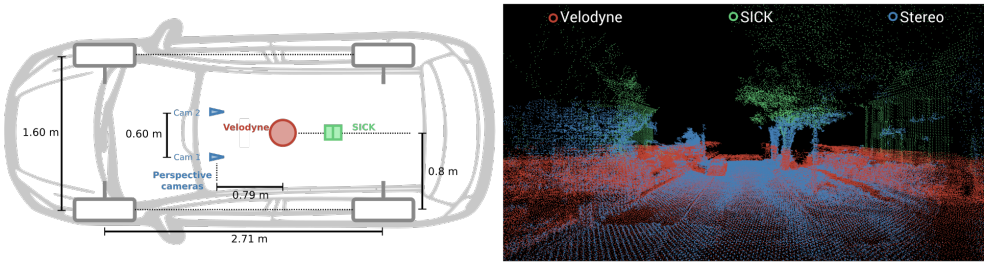



**Figure 2.1:** KITTI360 setup and example of obtained point cloud.

As we can see from Fig.2.1, a station wagon was equipped with:

- One 180° fish-eye camera to each side;
- A 90° perspective stereo camera;
- a Velodyne HDL-64E and a SICK LMS 200 laser scanning unit;
- an IMU/GPS localization system.

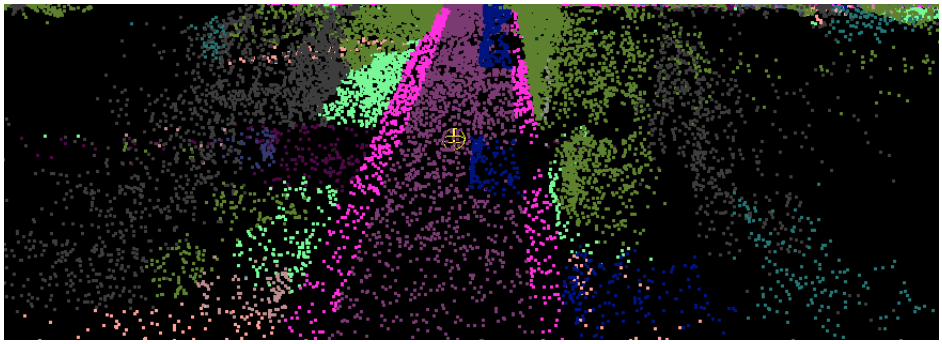


**Figure 2.2:** KITTI360 example of point cloud with semantic information.

For point cloud deep learning, they provide both raw 3D data and fused point clouds in order to perform either semantic segmentation and/or 3D object detection.


```
<?xml version="1.0"?>
<opencv_storage>
<object1>
  <index>4381</index>
  <label>car</label>
  <semanticId_orig>7</semanticId_orig>
  <semanticId>13</semanticId>
  <instanceId>1</instanceId>
  <category>instance</category>
  <timestamp>-1</timestamp>
  <dynamic>0</dynamic>
  <transform type_id="opencv-matrix">
    <rows>4</rows>
    <cols>4</cols>
    <dt>f</dt>
    <data>
      2.92481422e+00 -1.63844514e+00 2.43007307e-04 9.66490295e+02
      4.16620302e+00 1.15024292e+00 -7.32338813e-04 3.78474097e+03
      1.53910322e-03 8.15824198e-04 1.52057445e+00 1.14976791e+02 0. 0.
      0. 1.</data></transform>
  <vertices type_id="opencv-matrix">
    <rows>8</rows>
    <cols>3</cols>
    <dt>f</dt>
    <data>
      5.00000000e-01 5.00000000e-01 5.00000000e-01 5.00000000e-01
      5.00000000e-01 -5.00000000e-01 5.00000000e-01 -5.00000000e-01
      5.00000000e-01 5.00000000e-01 -5.00000000e-01 -5.00000000e-01
      -5.00000000e-01 5.00000000e-01 -5.00000000e-01 -5.00000000e-01
      5.00000000e-01 5.00000000e-01 -5.00000000e-01 -5.00000000e-01
      -5.00000000e-01 -5.00000000e-01 -5.00000000e-01 5.00000000e-01</data></vertices>
  <faces type_id="opencv-matrix">
    <rows>12</rows>
    <cols>3</cols>
    <dt>u</dt>
    <data>
      0 2 1 2 3 1 4 6 5 6 7 5 4 5 1 5 0 1 7 6 2 6 3 2 5 7 0 7 2 0 1 3 4
      3 6 4</data></faces>
  <start_frame>2</start_frame>
  <end_frame>385</end_frame></object1>
```


**Figure 2.3:** KITTI360 example of object ground truth.

They provide also stereo camera acquisitions (RGB, confidence maps), calibration matrices and system poses as well as raw (GPS/IMU) measurements.

KITTI360 can be a powerful dataset for computer vision tasks related to autonomous driving.

On the other hand, KITTI360 has not been involved in the training process of the network because:

- Raw data is not labeled. An .xml file contains all the 3D bounding box of the objects but only for fused point clouds (which are composed by approximately 3 milions points). Complex pre-processing algorithms are requested for selecting and sampling most valuable point clouds. We can inspect this behaviour in Fig.2.3;
- Low number of signals with poor quality. Since our main purpose is to detect props of the railway environment, we wanted to validate the model in an automotive surrounding, performing signal detections. An example can be seen in Fig.2.4;
- No benchmark is available at the moment which will be implemented in the future. Because of this, we are not able to compare results with state-of-the-art detectors.

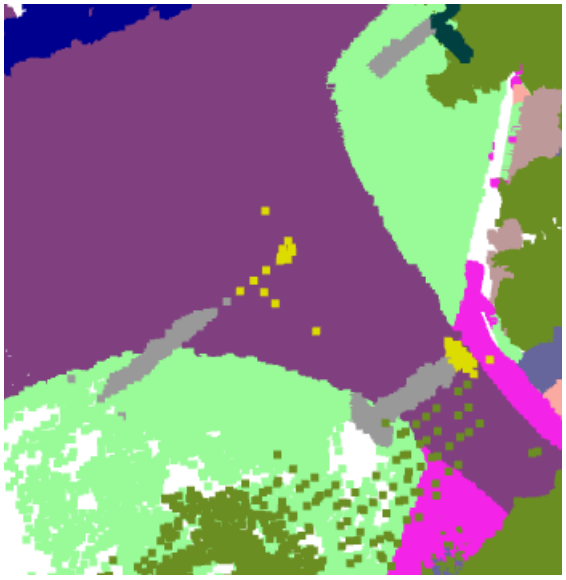


**Figure 2.4:** KITTI360 example of noisy signal.

## 2.2 RailSem19

**RailSem19** [22] dataset is a collection of mono RGB images which describes railway and inner-city tram scenes in order to be fed to an Artificial Neural

Network for semantic segmentation.

Together with KITTI360, RailSem19 does not provide a public benchmark so it is not clear which type of network suits better with railway environment.

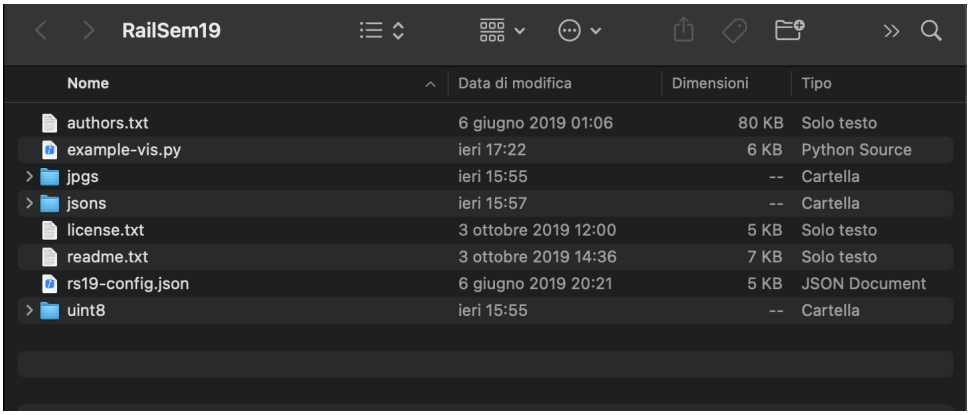



**Figure 2.5:** List of files inside the RailSem19 directory.

RailSem19 contains 8501 mono RGB images of railway environment with ground truth and its structure is defined as follow:

- *authors.txt* which contains all the links to YouTube videos where scenes are extracted in order to compose the dataset;
- *license.txt* which determines which license agreement is present for the dataset;
- *readme.txt* which decribes briefly the structure of the folder;
- **jpgs** is a folder which contains all the dataset images in *jpg* format (1920x1080 pixels mono RGB);
- **json** is a folder which contains a json for each jpg image and describes 2D bounding box with labels of detected objects in a scene. An example is in Fig.2.8;
- **uint8** is a folder which contains all dense semantic segmentation id maps of each scene using uint8 notation. An example is Fig.2.9;
- *rs19-config.json* is a configuration file where colors can be modified for each class described in the dataset;
- *example-vis.py* is a Python3 script which permits to visualize image in the collection. In Fig.2.6 we can inspect how sample code is working.


**Figure 2.6:** Example of image with semantic segmentation and object detection labels.

Inside the Python3 sample code and *json* files, we are able to identify dataset classes which are listed at the beginning of the script (Fig.2.7).

In order to be used for training in a deep learning strategy, we need to parse the json file for collecting the detected objects in the scene and then perform the loss between the prediction of the neural network and the ground truth of the specific scene. Since the number of detected objects is not fixed, a network which employs a **RPN** (Region Proposal Network) is mandatory in order to generalize as much as possible these types of scenarios.

```
rs19_label2bgr = {"buffer-stop": (70,70,70),
                  "crossing": (128,64,128),
                  "guard-rail": (0,255,0),
                  "train-car" :  (100,80,0),
                  "platform" : (232,35,244),
                  "rail": (255,255,0),
                  "switch-indicator": (127,255,0),
                  "switch-left": (255,255,0),
                  "switch-right": (127,127,0),
                  "switch-unknown": (191,191,0),
                  "switch-static": (0,255,127),
                  "track-sign-front" : (0,220,220),
                  "track-signal-front" : (30,170,250),
                  "track-signal-back" : (0,85,125),
                  #rail occluders
                  "person-group" : (60,20,220),
                  "car" : (142,0,0),
                  "fence" : (153,153,190),
                  "person" : (60,20,220),
                  "pole" : (153,153,153),
                  "rail-occluder" : (255,255,255),
                  "truck" : (70,0,0)
```

**Figure 2.7:** Map between bgr colors of uint8 images and labels.

**RailSem19** contains specific annotations collected from various scenario but images are extracted from Youtube videos and then no information about distances from train to landmarks is available. Furthemore, no point clouds are available.

Even if RailSem19 can be used for semantic segmentation and 2D object detection using, respectively, **FCN** (Fully Convolutional Networks) and *Mask RCNN*, **VOLIERA** and **RAILGAP** projects request to localize, respectively, landmarks and absolute position of the train during the run and, without distances and yaw angles, we cannot train properly the Neural Network for detecting railway signals and specific props of the environment and then place them correctly on the digital map.


```
{
    "frame": "rs00000",
    "imgHeight": 1080,
    "imgWidth": 1920,
    "objects": [
        {
            "boundingbox": [
                1066,
                571,
                1077,
                590
            ],
            "label": "track-sign-front"
        },
        {
            "boundingbox": [
                1024,
                599,
                1052,
                612
            ],
            "label": "switch-unknown"
        },
```


**Figure 2.8:** Structure of ground truth inside json files.

## 2.3 KITTI

Andreas Geiger et al. [20] developed novel challenging benchmarks for the tasks of stereo, optical flow, visual odometry / SLAM (*Simultaneous Localisation and Mapping*) and 3D object detection in 2012 called **KITTI**.

The 3D object detection KITTI benchmark consists of 7481 training images and 7518 test images as well as the corresponding point clouds, comprising a total of 80.256 labeled objects. For evaluation, KITTI adopts precision-recall curves and average precision for ranking models. Datasets are captured by driving around the mid-size city of Karlsruhe, in rural areas and on highways.

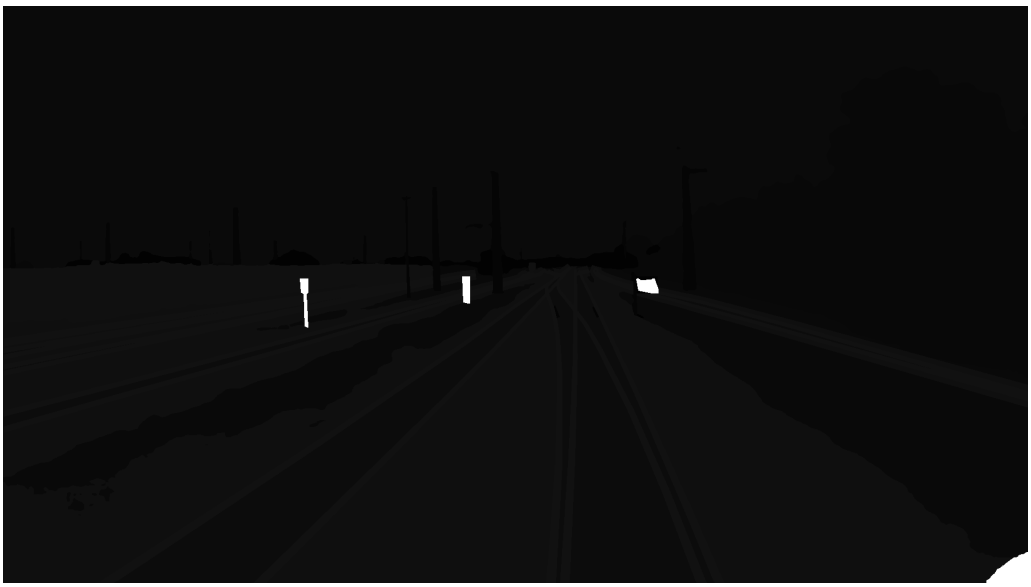


**Figure 2.9:** Example of semantic segmentation map of a sample image.

**Figure 2.10:** Example of left RGB image from KITTI Dataset.

KITTI has been used for network pre-validation in order to evaluate if the chosen neural network is suitable for the 3D Object detection task.

KITTI 3D Object Detection benchmark is structured in training and testing data. As we can see from Fig.2.11, each folder consists in:

- *calib* folder which contains camera matrix and transform matrix for each image and point cloud. Given a 3D point in camera coordinates with homogeneous coordinates $\mathbf{p_c} = [x, y, z, 1]^T$, we can convert into LiDAR coordinates by performing a matrix multiplication:

$$\mathbf{p_l}^{4\times1} = T_{c2l}^{4\times4}\mathbf{p_c}^{4\times1} \tag{2.1}$$

- *image_2* contains all mono RGB photos from the second camera. In our environment, we use them in order to plot predicted bounding box for visualization purposes;

- *label_2* contains the ground truth for each sample;
- *velodyne* contains all point cloud from the LiDAR sensor in .bin format.

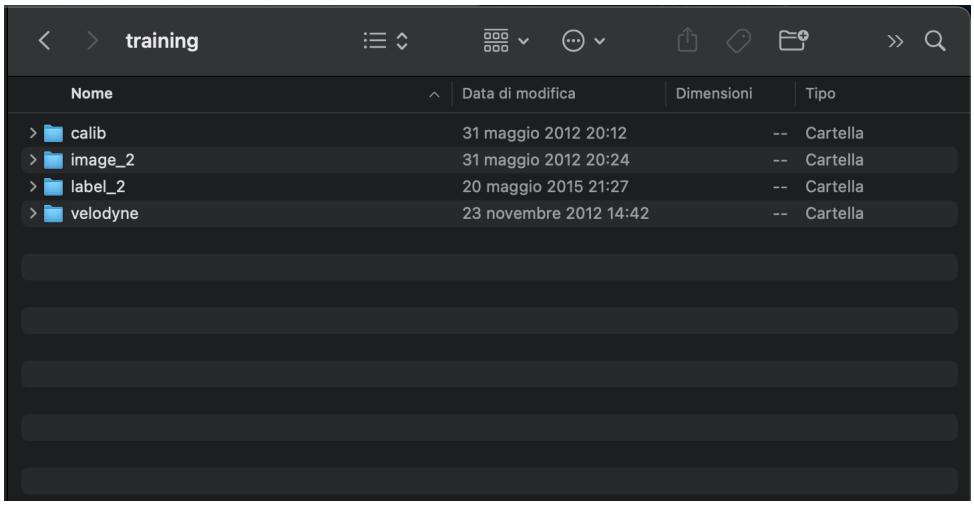



**Figure 2.11:** KITTI folder structure for 3D Object Detection suite.

Ground Truth is structured in 16 fields used as follow:

- 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. These values are used for classify decision difficulty;
- 1 **Theta** Observation angle of object (yaw), $\theta \in [-\frac{\pi}{2}, \frac{\pi}{2}]$;
- 4 **Bbox** 2D bounding box of object in the image (0-based index) contains left, top, right, bottom pixel coordinates $[u_{min}, v_{max}, u_{max}, v_{min}]$;
- 3 **Dimensions** 3D object dimensions: $[h, w, l]$ height, width, length in camera coordinates (in meters);
- 3 **Location** 3D object location $[x, y, z]$ in camera coordinates of the box center (in meters);
- 1 **Rotationy** Rotation $r_y$ around Y-axis in camera coordinates $[-\frac{\pi}{2}, \frac{\pi}{2}]$;

- 1 **Score** Only for results: Float, indicating confidence in detection, needed for precision/recall curves;

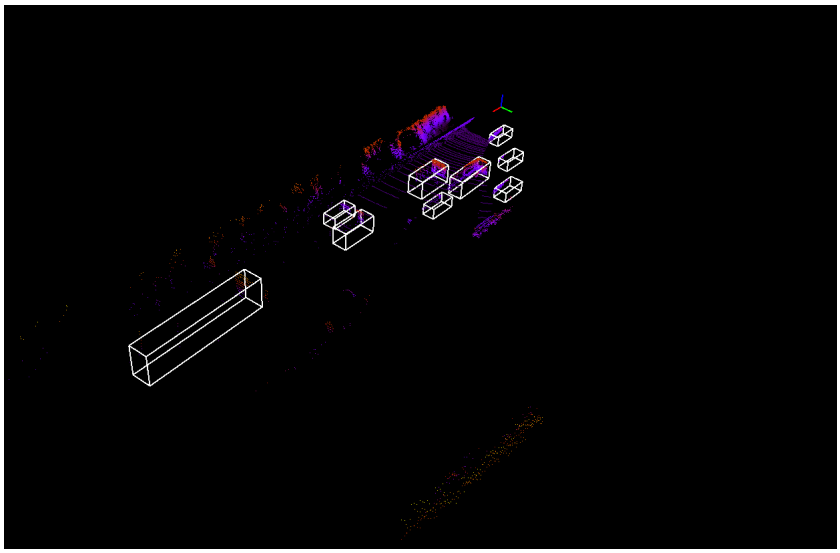


**Figure 2.12:** Example of KITTI point cloud with 3D bounding boxes.

As we can clearly see, labels are strictly correlated with provided images. Ground truth of an object is available if and only if it is on the field of view of the RGB camera. This issue could cause problems during training loop since correct detections of cars are labeled as false positive which are back-propagated through the network.

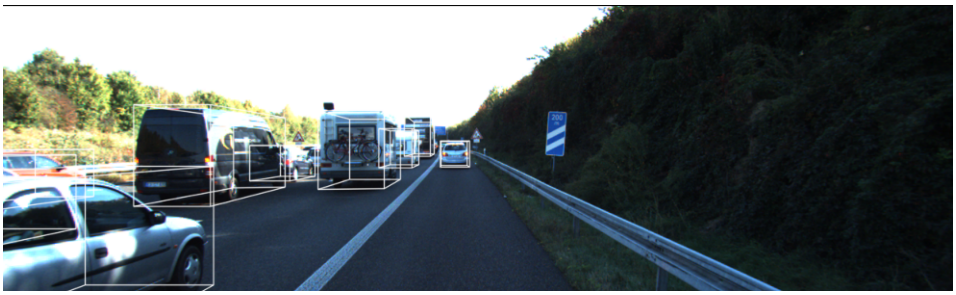


**Figure 2.13:** Example of left RGB with 3D bounding boxes.

## 2.4 Syntethic Dataset

Inspecting public datasets results in poor quality of labeled data to be used as training set. Even if KITTI has a great potential for automotive purposes, it is not suited for railways application. Deep learning strategies are based on observing a

huge amount of data of the same domain in order to learn relevant features and discover correlations.

Automotive environment is too different for using it as a training dataset for the railway context.

This strong data limitation forces us to choose between two possibilities:

- Resize **KITTI360** point clouds and mixing them with **KITTI** dataset, adjusting ground truth in order to perform object detection of traffic signs using only raw LiDAR data. Some scenes of the dataset are noisy and difficult to be handled. Moreover, there could be a problem of generalization and it could not be enough dense for a proper training of the model, risking overfitting;

- Thanks to the integration between Unreal Engine 4™ and Matlab™, we can code a train simulator which permits to simulate LiDAR scans in railway environment with a procedural generation of the railway with random spawning of signs for creating a synthetic point cloud dataset with known ground truth.

State-of-the-art shows multiple study cases where neural networks are trained using simulated data [26] with comparable results w.r.t. real ones, conducting us to start developing a train simulator.

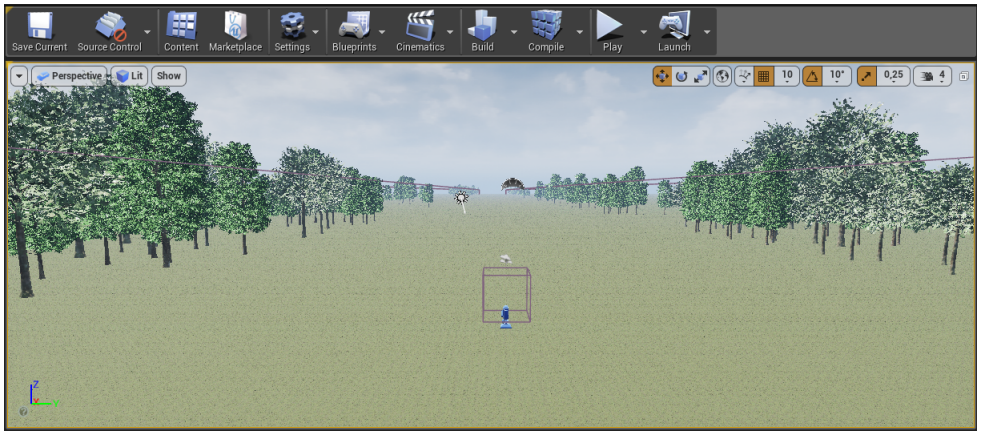



**Figure 2.14:** Main window of Unreal Engine™ software.

### 2.4.1 Matlab™ - LiDAR data simulation

The Automated Driving Toolbox simulation blocks provide the tools for testing and visualizing path planning, vehicle control, and perception algorithms. It supplies also an interface to the LiDAR sensor in a 3D simulation environment. This

environment is rendered using the Unreal Engine™ from Epic Games™. The block returns a point cloud with the specified field of view and angular resolution. We set up the LiDAR to be on top of the train (approximately 3.7 meters from ground level) with the same physical configurations of the LIVOX™ Horizon sensor.

The Simulation 3D Scene Configuration block requires an Unreal Engine™ executable or project in order to run simulations.

Simulation 3D Scene Configuration and LiDAR Simulation blocks from Automated Driving Toolbox compose a Simulink™ model which is capable of being executed iteratively, creating multiple point clouds consecutively.

The toolbox includes 9 built-in scenes where researchers can test their algorithms, such as lane keeping assist and adaptive cruise control.

### 2.4.2 Unreal Engine 4™ - Simulating railway environment

Unreal Engine™ is made up of several components that work together to drive the game. Its massive system of tools and editors allows to organize assets and manipulate them to create the game play for the game.

In Fig.2.14 we can inspect the main windows of the editor where assets and textures can be viewed during development.

Before starting creating the railway, we modeled some specific signals which are going to be spawned near railtracks. An example of a model of a railway sign support is illustrated in Fig.2.15.

In our setup, we respect distances and dimensions during the modeling of rails, signals and props of the railway environment. We used Blender™ which is a free and open-source 3D computer graphics software tool set used for creating animated films, visual effects, art and 3D printed models.

Models in the Unreal Engine™ environment are called Game Object which can be a static mesh, point light, cameras and others possible props of the game level.

Logic structure of Unreal Engine™ is provided by programming Blueprints.

The Blueprint Visual Scripting system in Unreal Engine is a complete game play scripting system based on the concept of using a node-based interface to create game play elements from within Unreal's Editor. As with many common scripting languages, it is used to define object-oriented (OO) classes or objects in the engine.

Each level of a game has a **Blueprint Level** where initial configuration are set up.

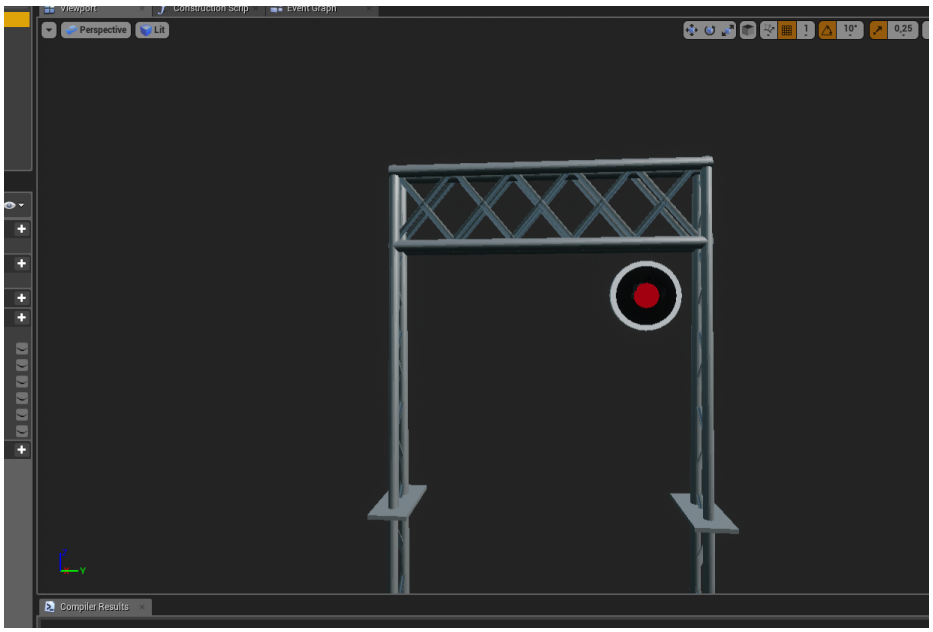



**Figure 2.15:** Example of 3D model of a Railway support sign with a traffic light.

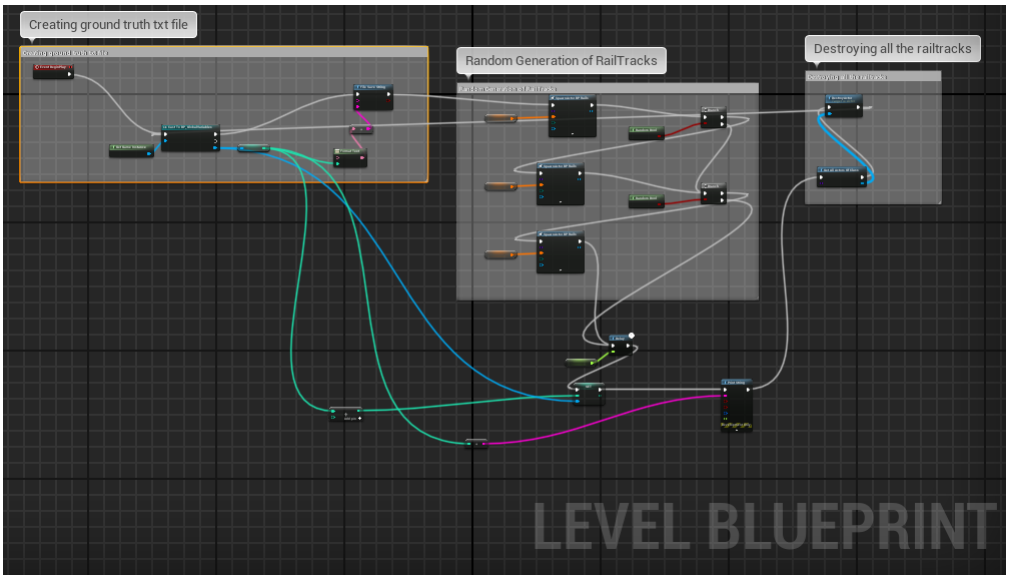



**Figure 2.16:** Blueprint scripting for generating track rails.

Fig.2.16 shows simulator blueprint which permits to:

- Generate a *.txt* file where ground truth is going to be written in next steps;
- Spawn a random number of railtracks, from 1 to 3. Since field of view of LiDAR is not wide, we restricted the number of railtracks for the purpose of reducing complexity of the simulation. At least one railtrack is spawned - the one where the train is following;

- Destroy all railtracks at the end of simulation for memory re-allocation intentions.

The initial idea was to generate multiple point clouds in a single simulation by changing the environment rapidly. The main issue was the difficulty to synchronize both Unreal Engine and Matlab™ time since the two programs interpret simulated time in different way.

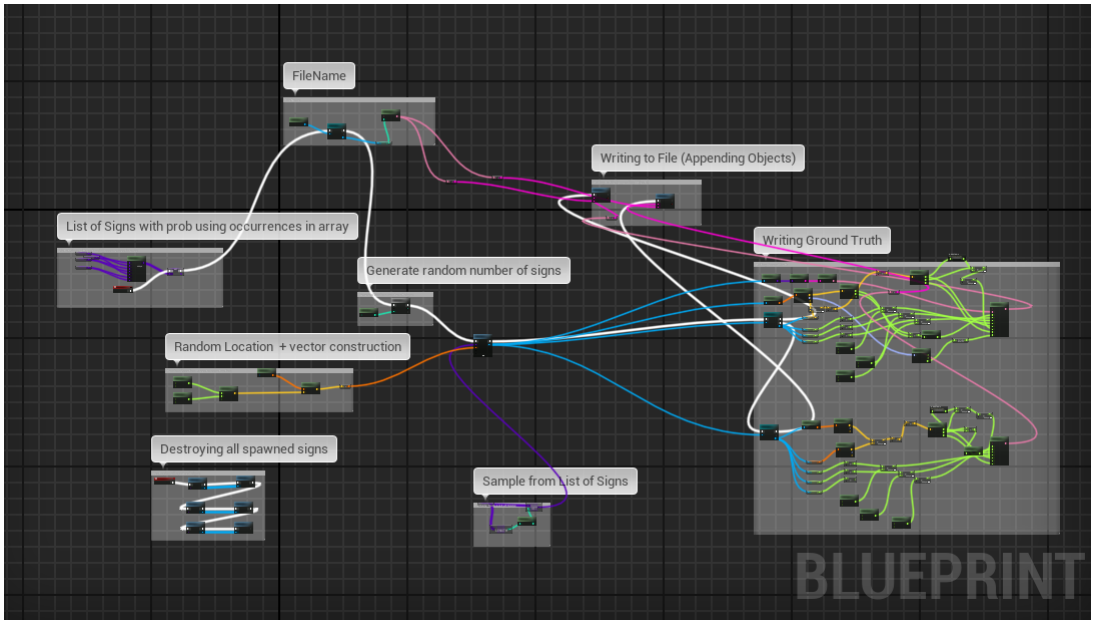



**Figure 2.17:** Blueprint scripting for generating landmark near track rails.

Spawning a railtrack calls its Blueprint which creates landmarks nearby. Process is displayed in Fig.2.17 which can be summarized as follows:

- We create an array of possible signals to be spawned. Adding multiple instances of the same landmark permits to model signal distribution;
- For each railway, we sample an integer from a uniform distribution which describes the number of signals near railtracks;
- For each signal, we sample $x$ and $y$ coordinates of the landmark from a uniform distribution over the left-side of the railtracks. In Italy, signals are installed on the left side of the road;
- We write ground truth using KITTI format in order to store $[x_c, y_c, z_c]$ of the 3D box with its dimensions $[l, w, h]$ and yaw angle $\theta$.
- In conclusion, we destroy all the Game Objects spawned for memory re-allocation purposes.

Final output of simulation is then a *.txt* file which contains all the characteristics of spawned landmarks.

We have to emphasize that every parameter of the Blueprint can be modified, according to the railway environment. Adding different signals is a straightforward procedure which takes into account applying object-oriented paradigm.

We develop a Blueprint interface which permits to customize parameters of 3D bounding box w.r.t. the type of signal. In this way, we inherit proprieties from the interface to the extended signal class and then fetch data, by accessing its public variables, and write to ground truth file of the simulation.

### 2.4.3 Dataset Generation

LiDAR simulator adds a displacement with respect to the Y axis. As we can see from Fig.2.18, distortion is strictly correlated with X value e.g. the more distant the object is from the sensor, the more distorted the Y axis is.

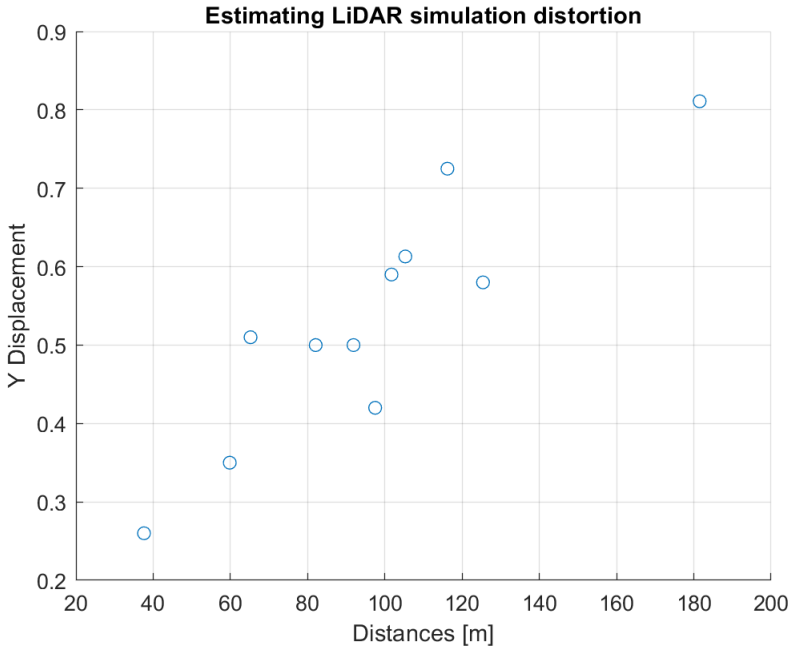



**Figure 2.18:** Estimation of LiDAR distortion on Matlab™ simulation.

In order to fix this, we regress the problem into a linear function which depends on X axis and add the predicted displacement to Y axis during ground truth generation.

For collecting multiple point cloud instances, we code a script which permits to:

- Create automatically a dataset which is splitted in training and validation set. Percentage of each subset is defined at the preamble of the code;
- Define the number of point clouds to generate;
- Plot one random (or fixed) point cloud and display bounding box ground truth for visualization and testing purposes (class and dimensions of the box).

For our aim, we decide to simulate KITTI settings so:

- Generation of 7500 point clouds which takes approximately one day and an half;
- Split syntethic dataset in 50% training and 50% validation sets.

With these settings, syntethic dataset has a size of approximately 50 GB to be fed to our VoxelNet.

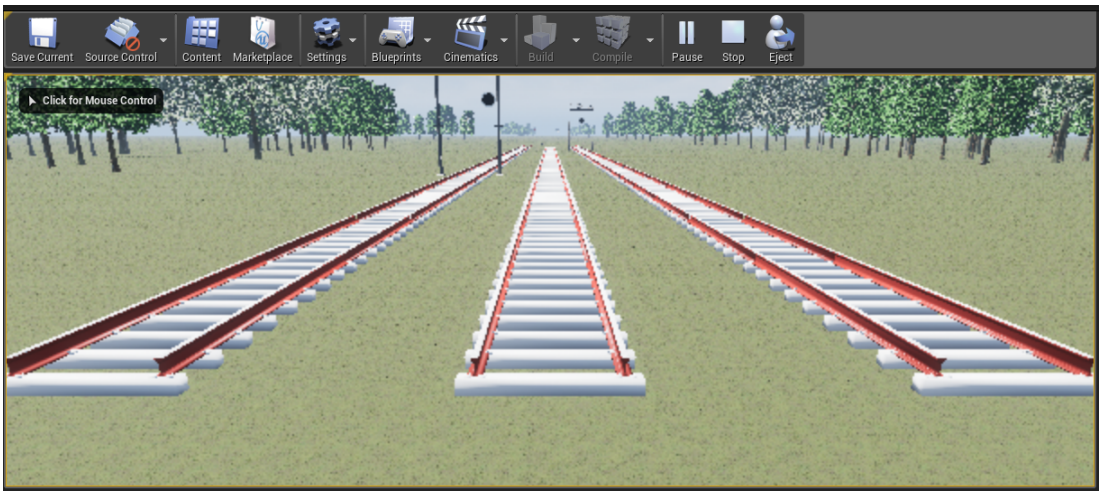



**Figure 2.19:** Snapshot of the Unreal Engine's viewport during a simulation run.

An example of running simulation is in Fig.2.19 where 3 tracks are spawned with support signs and traffic lights.

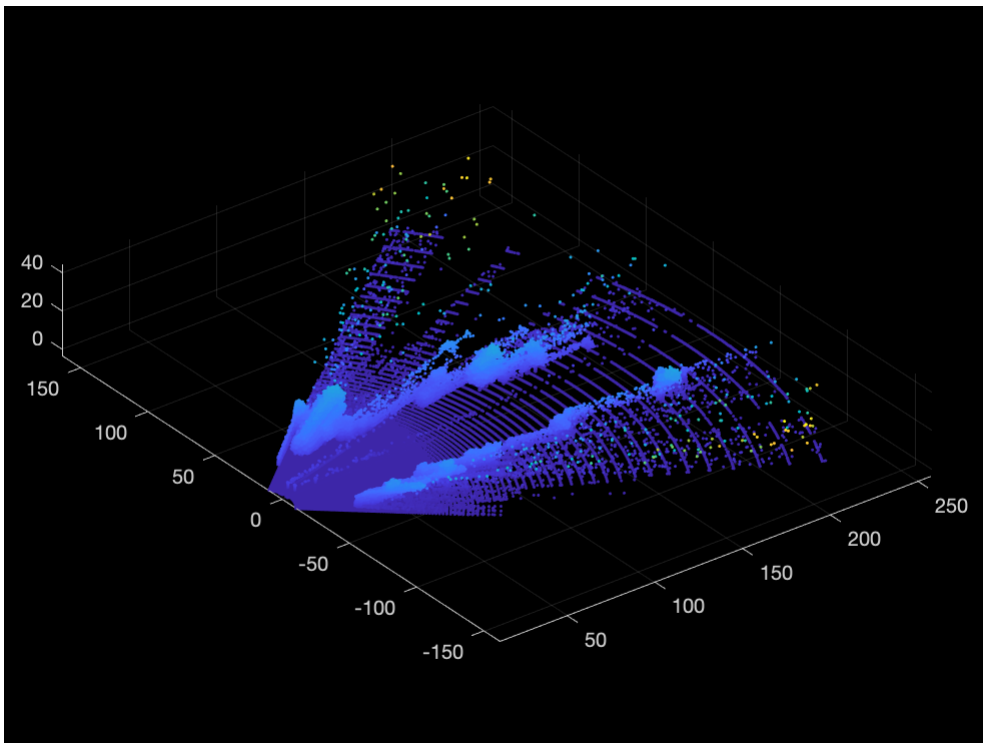



**Figure 2.20:** Example of synthetic point cloud.

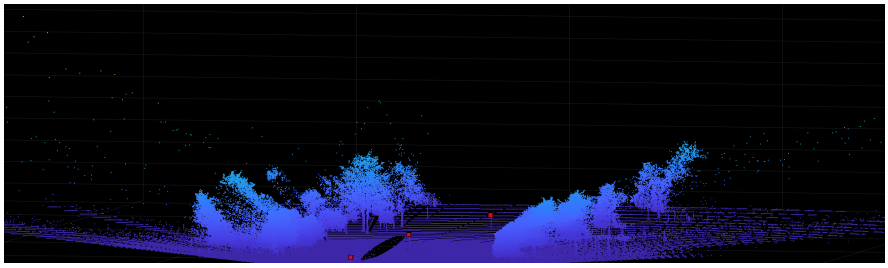


**Figure 2.21:** Syntethic point cloud with annotations.

# Chapter 3

# VoxelNet - Deep Learning Architecture

## 3.1 Deep Learning Strategy Selection

Unlike images and videos, LiDAR point clouds are sparse and have highly variable point density due to non-uniform sampling of 3D environment, occlusions and relative poses. To handle this challenge, many approaches with different types of representations are defined.

PointNet architecture, developed by Charles R.Li et al [9], permits to extract a feature vector $z \in \mathbb{R}^n$, where $z$ is called *latent vector* with its length $n$ and belongs to the *latent space*, from raw LiDAR data in order to perform classification and semantic segmentation tasks.

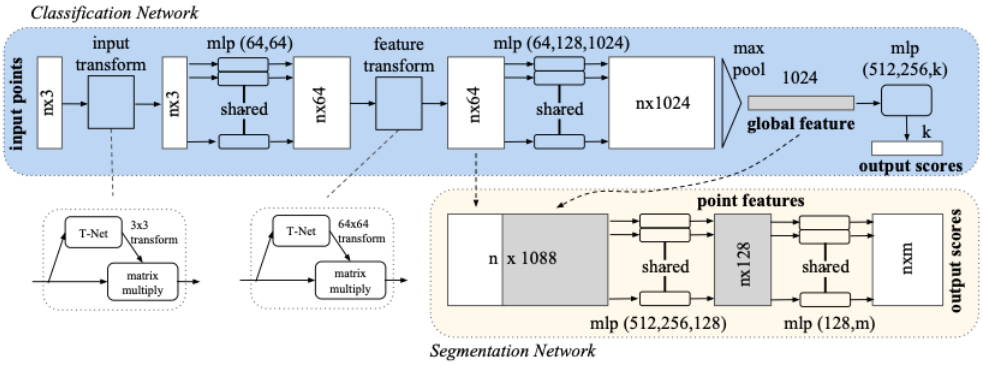



**Figure 3.1:** Structure of the point cloud encoder PointNet.

It can be also used for 3D object detection in small point clouds.

The main problem of this approach is that it requires to feed the whole point cloud, which is around one hundred thousands points, to the encoder causing

massive hardware and computation requirements.

From Fig.3.1, we can see that the whole point cloud is encoded in a fixed-length float vector, specifically a 1024-length vector.

In terms of compression, it is a powerful method for reducing the size of data. Unfortunately, features extracted are not representative for a 3D space. Moreover, spatial information is lost during the process so no generalization is possible.

Pointnet can be used as encoder-backbone for more advanced deep architecture, such as 3D-BoNet [16], but is not suitable for large scale point clouds because of encoder representation limits.

Global features, displayed in Fig.3.2 with the architecture of 3D-BoNeT, are obtained by feeding the whole point cloud to PointNet.

Experiments on 3D-BoNeT with KITTI data stood out difficulties on predicting bounding box in large scale environments.

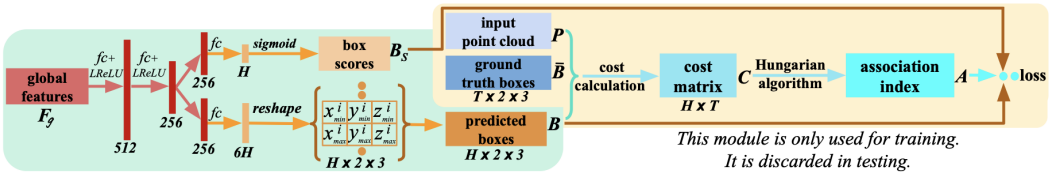



**Figure 3.2:** 3D-BoNet architecture.

Many other approaches, involving image processing and machine learning, implement sensor fusion between point clouds and images [17] and then provide improved performances compared to LiDAR-only 3D detection, particularly for small objects (pedestrians, cyclists) or when landmarks are far from sensor.

For our aim, we deploy the **VoxelNet** deep network [18].

It belongs to the class of networks which deploy voxelization pre-processing, creating a grid representation of the point cloud.

## 3.2 VoxelNet architecture

**VoxelNet** was proposed by Yin Zhou and Once Tuzel [18] in 2019 in order to perform 3D Object Detection on point clouds. The authors proposed a novel end-to-end (without jumps and/or context vectors) trainable deep architecture that operates on sparse 3D points, presenting an efficient method to implement it without information bottlenecks and developing efficient parallel processing of voxels.

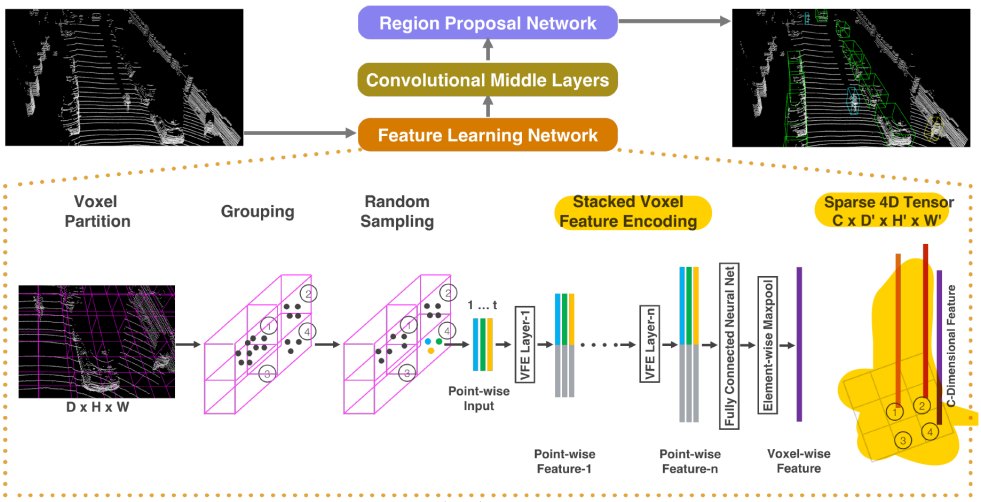



**Figure 3.3:** VoxelNet architecture.

They conducted experiments on KITTI dataset and showed that VoxelNet had produced state-of-the-art results by exploiting only raw data.

Neural network is structured in three main parts as we can see from Fig.3.3:

- **Feature Learning Network**;
- **Convolutional Middle Layers**;
- **Region Proposal Network**

The following sections will deeply focus on each of the aforementioned processes in order to demonstrate efficiency and accuracy of the deep network.

### 3.2.1 Feature Learning Network

Let $\mathbf{P} = \{\mathbf{p_i}\}, \quad \forall i = 1 \dots N$ be the input point cloud with $N$ points and $\mathbf{p_i} = [x_i, y_i, z_i, r_i]^T$ a 3D point which is characterized using 3D coordinate $(x_i, y_i, z_i)$ and reflectivity value $r_i$.

We subdivide the point cloud $\mathbf{P}$ into equally spaced voxels in order to be fed to Convolutional Middle Layers. Let's first assume that our point cloud is 3D bounded with range $D, H, W$ in meters along the $Z, Y, X$ coordinates, respectively.

We define voxel sizes as $v_D, v_H, v_W$ in order to obtain a voxel partition which has size $[D', H', W']$. They are computed as follows:

- $D' = \frac{D}{v_D}$

- $H' = \frac{H}{v_H}$
- $W' = \frac{W}{v_W}$

Without loss of generality, we assume that $[D', H', W']$ are integers. After the splitting phase, we assign to each point $\mathbf{p_i}$ to the corresponding voxel $\mathbf{V_i} = \{\mathbf{p_i}\}, \quad \forall i = 1 \dots N_i$.

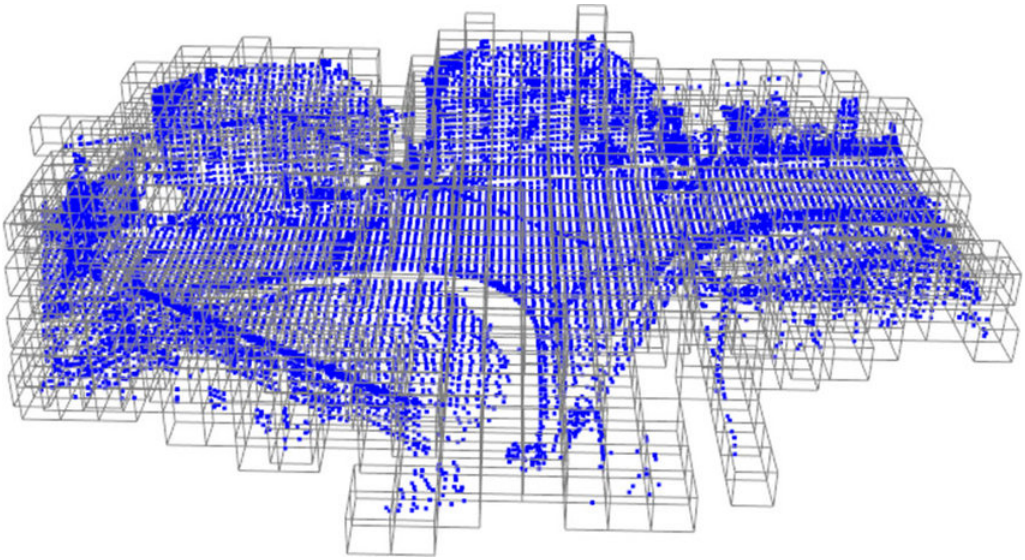


**Figure 3.4:** Voxelization of the point cloud.

We have to notice that each voxel $\mathbf{V_i}$ can group different amount of 3D points or also be empty. This is one of the most critical drawbacks of the network since it does not re-allocate memory from empty voxels.

We can affirm that space complexity of the network depends on voxels sizes and not on point cloud density.

For the purpose of extracting regular features from voxels, we perform random sampling in order to fix the number of contained 3D points for each voxel. At the end, we have a certain number of non-empty voxels which contain $T$ points each. Thanks to this process, we accomplish three important goals:

- Computational savings;
- Decreasing imbalance between voxels with different number of points;
- Reducing bias and adding some variance in order to increase training accuracy.

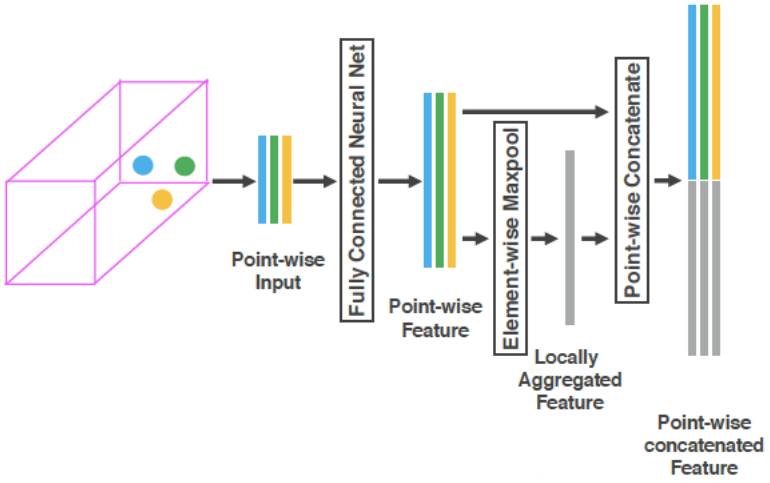



**Figure 3.5:** Single VFE encoding process.

Last but not least, we define a stacked **VFE** (*Voxel Feature Encoder*) which is the key algorithm for excerpting relevant features from voxels. Given a non-empty voxel $\mathbf{V_i} = \{\mathbf{p_i} \in \mathbb{R}^4, \quad \forall i = 1 \dots T\}$, we first compute the local mean $\mu_\mathbf{i} = [\mu_x, \mu_y, \mu_z]$ as the centroid of all the points inside the voxel $\mathbf{V_i}$, i.e:

$$\mu_\mathbf{i} = \begin{cases} \mu_x = \frac{\sum_{\mathbf{p_i} \in \mathbf{V_i}} x_i}{T} \\ \mu_y = \frac{\sum_{\mathbf{p_i} \in \mathbf{V_i}} y_i}{T} \\ \mu_z = \frac{\sum_{\mathbf{p_i} \in \mathbf{V_i}} z_i}{T} \end{cases} \tag{3.1}$$

Thanks to this piece of information, we are able to augment each point by removing its characteristic local mean and obtaining an augmented point $\hat{\mathbf{p}}_\mathbf{i}$. Therefore, we are able to define an input feature set $\mathbf{V_{in}}$ which is described as

$$\mathbf{V_{in}} = \{\hat{\mathbf{p}}_\mathbf{i} = [x_i, y_i, z_i, r_i, x_i - \mu_x, y_i - \mu_y, z_i - \mu_z]^T \in \mathbb{R}^7, \quad \forall i = 1 \dots T\} \tag{3.2}$$

Next step is to transform each augmented point through a **FCN** (*Fully Connected Network*) into a point features $\mathbf{f_i}$.

We connect information from point features $\mathbf{f_i} \in \mathbb{R}^m$, where $m$ is the length of the feature vector, to represent voxel shape.

FCN is composed of a linear layer, a Batch Normalization layer and a REctified Linear Unit (ReLU).

We apply element-wise MaxPooling across all $\mathbf{f_i}$ associated to $V_i$ to get the local aggregated feature vector $\hat{\mathbf{f}}_\mathbf{i} \in \mathbb{R}^m$.

A final augmentation of data is performed by concatenating point features $\mathbf{f_i}$ and local feature vector $\hat{\mathbf{f}}_\mathbf{i}$ into

$$\mathbf{f_i^{out}} = [\mathbf{f_i}\hat{\mathbf{f}}_\mathbf{i}] \in \mathbb{R}^{2m} \tag{3.3}$$

This process is iterated for extracting more peculiar features from spatial information of the voxel.

We use the notation VFE-i($c_{in}, c_{out}$) to represent the $i$-th layer that transforms input feature vectors of dimension $c_{in}$ to output feature vectors of dimension $c_{out}$.

The result of the whole feature learning process is a 4-D tensor of size $[C, D', H', W']$ where $C$ is the length of the feature vector $c_{out}$ obtained from the last VFE layer.

Since point clouds are sparse by definition, resulting tensor is typically sparse reducing memory usage and computation cost during back-propagation algorithm.

### 3.2.2 Convolutional Middle Layers

This intermediate section of the network consists in extracting relevant features from the 4-D tensor. Let *ConvMD*($c_{in}, c_{out}, \mathbf{k}, \mathbf{s}, \mathbf{p}$) be the M-dimensional convolution operator where:

- $c_{in}$ and $c_{out}$ are the numbers for input and output channels, respectively;
- $\mathbf{k}$ is the kernel size;
- $\mathbf{s}$ is the stride size;
- $\mathbf{p}$ is the padding operator;

If the size is the same for each dimension, we define it as a scalar e.g. $s$ for $\mathbf{s} = (s, s, s)$.

For each convolutional network, we attach a Batch Normalization layer and a ReLU layer sequentially.

Thanks to this configuration, we are able to extend receptive fields of the network.

Since some parameters are different w.r.t. the dataset used, further insights can be inspected in Chapter 4.

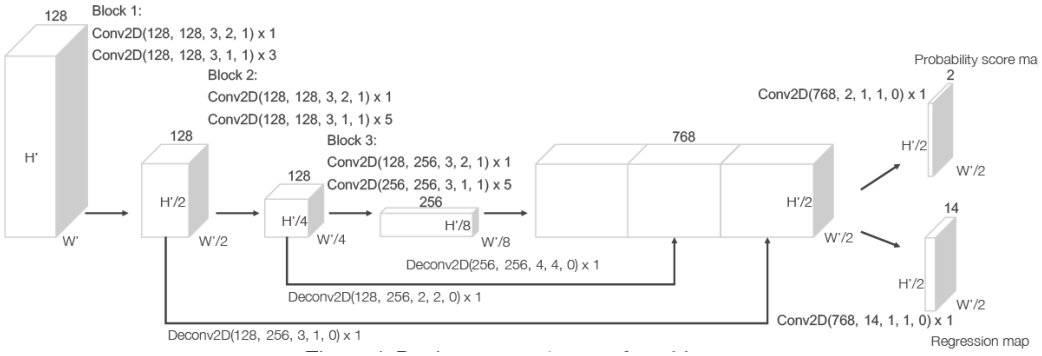



**Figure 3.6:** Region Proposal Network structure.

### 3.2.3 Region Proposal Network

Both for 2D and 3D object detection, it is crucial to implement a **RPN** (*Region Proposal Network*) in order to obtain accurate detection.

In Fig.3.6 we can see that output of Convolutional Middle Layers is fed to a complex structure of convolutional and de-convolutional network in order to create a final feature map, made by up sampling and concatenating each intermediate result.

In this manner, we are able to apply a **MLP** (*Multi-Layer Perceptron*) branch for mapping the final feature vector into:

- a probability score map which can be displayed as an heat map with shape $\kappa \times H'/2 \times W'/2$, where $\kappa$ is the number of pre-defined anchors. They are discussed in Chapter 4;
- a regression map with shape $7\kappa \times H'/2 \times W'/2$;

The number of anchors $\kappa$ affects only the last layers of the overall network, changing the shape of the probability score map and the regression map. The vast majority of rail signals have different sizes which permits to classify the detected object w.r.t. the dimensions of the correct anchor.

The following section will explain how regression task and classification of true predictions is done by implementing the loss function.

## 3.3 Loss Function

Thanks to the loss function gradient w.r.t. network weights $\nabla_{\mathbf{\Theta}}$, we are able to apply the back-propagation algorithm in order to change weights of the neural network.

For this main reason, loss function is a fundamental component of the network.

Let $\{a_i^{pos}\}_{i=1...N_{pos}}$ be the set of $N_{pos}$ positive anchors and $\{a_i^{neg}\}_{i=1...N_{neg}}$ be the set of $N_{neg}$ negative anchors.

We define a generic ground truth as a vector $[x^g, y^g, z^g, l^g, w^g, h^g, \theta^g]$ where:

- $[x^g, y^g, z^g]$ are the coordinates of the 3D box center;
- $[l^g, w^g, h^g]$ are the dimensions of the bounding box;
- $\theta^g$ is the rotation w.r.t. Z axis of the bounding box.

Using the same notation, we define the predicted anchor from the network as the vector $[x^p, y^p, z^p, l^p, w^p, h^p, \theta^p]$, obtained from the regression map of the RPN.

Thanks to the previous values, we introduce the residual vector $\mathbf{u_i^*}$ which contains all the 7 regression targets e.g. $\mathbf{u_i^*} = [\Delta x, \Delta y, \Delta z, \Delta l, \Delta w, \Delta h, \Delta\theta]$.

All the values are evaluated as:

$$\mathbf{u_i^*} = \begin{cases} \Delta x = \frac{x^g - x^p}{d^p} \\ \Delta y = \frac{y^g - y^p}{d^p} \\ \Delta z = \frac{z^g - z^p}{h^p} \\ \Delta l = \log \frac{l^g}{l^p} \\ \Delta w = \log \frac{w^g}{w^p} \\ \Delta h = \log \frac{h^g}{h^p} \\ \Delta\theta = \theta^g - \theta^p \end{cases} \tag{3.4}$$

where $d^p = \sqrt{(l^p)^2 + (w^p)^2}$ is the diagonal of the predicted anchor box base which is used in order to normalize $\Delta x$ and $\Delta y$ for better performances.

Computing the ground truth residual vector $\mathbf{u_i}$ for the i-th box at the loading dataset script, we are able to define the loss function:

$$\mathcal{L}oss = \underbrace{\frac{\alpha}{N_{pos}} \sum_i L_{cls}(p_i^{pos}, 1)}_{\text{Classification Negative Loss}} + \underbrace{\frac{\beta}{N_{neg}} \sum_j L_{cls}(p_j^{neg}, 0)}_{\text{Classification Positive Loss}} + \underbrace{\frac{1}{N_{pos}} \sum_i L_{reg}(\mathbf{u_i}, \mathbf{u_i^*})}_{\text{Regression Loss}} \tag{3.5}$$

where:

- $p_i^{pos}$ and $p_j^{neg}$ are the softmax probability of positive and negative anchors, respectively. It permits to assign high probabilities to boxes which are more likely to describe a true positive object;

- $\alpha$ and $\beta$ are regularization terms in order to balance network decisions;

- $\mathbf{u_i} \in \mathbb{R}^7$ and $\mathbf{u_i^*} \in \mathbb{R}^7$ are ground truth and predicted residual vectors for positive anchor $a_i$.

First two classification terms take into account Binary Cross-Entropy loss whereas Huber Loss or SmoothL1 (combining Mean Squared Error and Mean Absolute Error loss) is applied for regression task.

Regression loss is defined as:

$$L_{reg}(p,q) = \begin{cases} 0.5(p-q)^2 & if \quad |(p-q)| \leq \delta \\ \delta(|(p-q)| - 0.5\delta) & otherwise \end{cases} \tag{3.6}$$

where $p, q$ are prediction and target values, respectively. The hyper parameter $\delta$ trades off between linear and quadratic loss. In our environment, we empirically set $\delta = 1.5$ since it is a good starting value for clipping the gradient of the loss, acting as a regularizer.

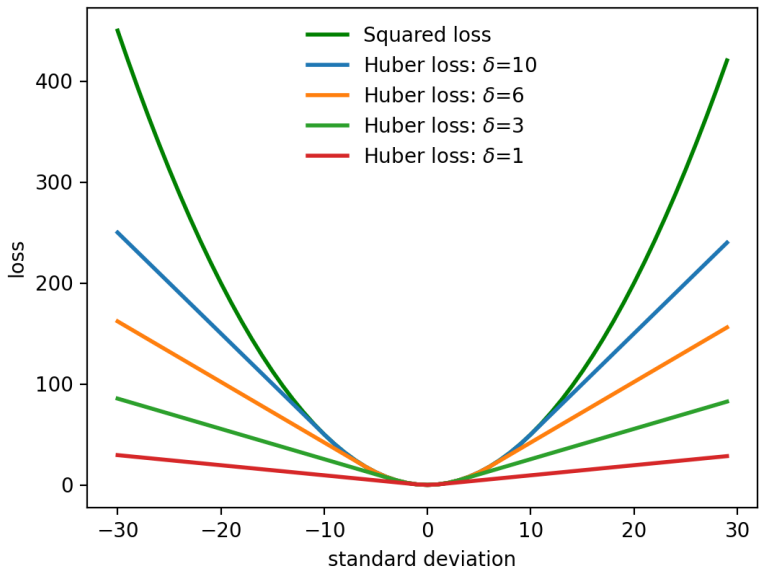



**Figure 3.7:** Huber Loss w.r.t.$\delta$ parameter.

Classification loss is defined as:

$$L_{cls}(\mathbf{p}, \mathbf{q}) = -\frac{1}{N}\sum_{i=1}^{N} p_i \log q_i + (1 - p_i) \log (1 - q_i) \tag{3.7}$$

where $\mathbf{p}$ and $\mathbf{q}$ are the prediction and target vector probabilities with $N$ samples, respectively. During training, we avoid the logarithm of zero by adding a small add-on (approximately $1e^{-6}$).

Classification losses are fundamental since they are responsible for raising softmax probabilities for correct detections and, on the other hand, decreasing them for wrong predictions.

We can inspect examples of binary masks for ground truth in Fig.3.9 and Fig.3.8.

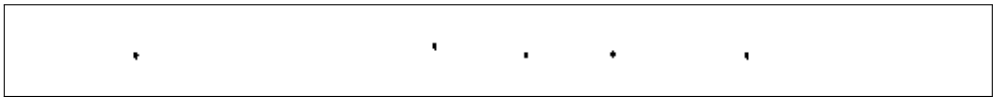


**Figure 3.8:** Example of binary mask displaying set of voxels where black voxels represent one object.

**Figure 3.9:** Example of binary mask displaying set of voxels where white voxels represent one object.

Moreover, during prediction step, detections are filtered as follow:

- Removing predictions with score lower than a fixed threshold $\delta$. After collecting all the predictions of a specific point cloud, each bounding box with score $s_i < \delta$ is discarded and not taken into account for evaluation of metrics;

- Performing a **2D - NMS** (*Non-Maximum Suppression*).

NMS permits to prune away detections that have high IoU overlap with each others.

Let $box_1$ and $box_2$ two predictions of the network targeting the same ground truth which are displayed on bird-view, e.g. $box_1 = [u_{min}, u_{max}, v_{min}, v_{max}]$, then IoU is evaluated as the ratio between intersection and union areas:

$$IoU(box_1, box_2) = \frac{box_1 \bigcap box_2}{box_1 \bigcup box_2} \tag{3.8}$$

In our surrounding, we evaluate NMS on bird-view, e.g. collecting box from the XY-plane for each target and then evaluating IoU.

This algorithm is agnostic to orthogonal transformation and translation and it yields a set of integers indexing into the input collection in descending order of score.

Implementation and configuration of the NMS is fundamental for every 2D/3D object detection project, selecting best bounding boxes for each prediction run.

# Chapter 4

# Training Details

## 4.1 Network Details

Anchors are reference boxes that are placed at different positions in the voxel grid. They can be seen as a powerful hyper parameter of the network which permits to classify different objects w.r.t. their dimensions and shapes.

Let $\kappa$ be the number of different anchors for each voxel.

In our setup, $\kappa = 2$ anchors are generated for each voxel with fixed sizes and two different rotations, 0 and 90 degrees around Z axis.

Thus, we can affirm that number of total predictions is exactly $\kappa \times D' \times H' \times W'$ since $H' \times W'$ represents point cloud voxel grid with depth $D'$.

At the beginning of the training phase, we generate these anchors and assign them positive and negative labels w.r.t. their IoU with ground truth.

Thanks to this procedure, we are able to define a binary feature map which contains 1 if an object is present inside the voxel, 0 otherwise.

In this way, the network learns from previous wrong predictions and corrects its error by back-propagating the derivative of the loss on its weights.

### 4.1.1 KITTI

**Car Detection.** For this specific task, we filter out not useful points into the input range $[0, 70.4] \times [-40, 40] \times [-3, 1]$ meters along X, Y, Z axis respectively. These values refer to the maximum sizes of KITTI point clouds.

We use as voxel size $v_D = 0.4, v_H = 0.2, v_W = 0.2$ meters, which leads to a feature map of dimensions $D' = 10, H' = 400, W' = 352$. Each voxel can contain at the most $T = 35$ 3D randomly sampled points from the encoding phase of the

network.

In the Feature Learning Network, we deploy two VFE layers, VFE-1$(7, 32)$ and VFE-2$(32, 128)$ in order to obtain a sparse tensor of shape $128 \times 10 \times 400 \times 352$.

Three convolution middle layers are employed sequentially in order to aggregate voxel-wise features which are

- $Conv3D(128, 64, 3, (2, 1, 1), (1, 1, 1))$,
- $Conv3D(64, 64, 3, (1, 1, 1), (0, 1, 1))$,
- $Conv3D(64, 64, 3, (2, 1, 1), (1, 1, 1))$.

The final output tensor is resized in order to have a shape of $128 \times 400 \times 352$ which will be fed to the Region Proposal Network. Finally, regression and classification maps are obtained as in Fig.3.6.

We define one anchor with size $l^p = 3.9, w^p = 1.6, h^p = 1.56$ meters centered at $z_c^p = -1.0$ meters with two rotations, 0 and 90 degrees.

Classification of anchors is performed w.r.t. the highest IoU between all the ground truth using the following formula:

$$\begin{cases} 1 \quad if \quad \max IoU > 0.6 \\ 0 \quad if \quad \max IoU < 0.45 \\ \text{Don't Care} \quad if \quad 0.45 \leq \max IoU \leq 0.6 \end{cases} \tag{4.1}$$

Don't Care label means that prediction does not affect classification loss.

Considering Loss function in Eq.3.5, we set $\alpha = 1.5$ and $\beta = 1$.

**Pedestrian and Cyclist Detection.** In this situation, we set the input range as $[0, 48] \times [-20, 20] \times [-3, 1]$ meters along X,Y,Z axis respectively. Using the same voxel sizes for car detection, it yields $D' = 10, H' = 200, W' = 240$.

Nevertheless, we set $T = 45$ for collecting more points in a voxel and then preserving shape information.

Intermediate layers are identical w.r.t. car detection except for the first convolutional middle layer. In this case, we change the stride from 2 to 1 so the layer becomes $Conv3D(128, 64, 3, (1, 1, 1), (1, 1, 1))$. This change permits to refine detections for smaller objects such as pedestrian and cyclist.

Anchor sized are $l^p = 0.8, w^p = 0.8, h^p = 1.73$ meters centered as $z_c^p = -0.6$ with 0 and 90 degrees rotation w.r.t. Z axis.

In order to recognize signal with different shapes, other runs use another anchor size which is $l^p = 0.2, w^p = 0.8, h^p = 1.73$ meters centered as $z_c^p = -0.6$ with 0 degrees rotation w.r.t. Z axis.

Anchor matching criteria is different from the car detection:

$$\begin{cases} 1 \quad if \quad \max IoU > 0.5 \\ 0 \quad if \quad \max IoU < 0.35 \\ \text{Don't Care} \quad if \quad 0.35 \leq \max IoU \leq 0.5 \end{cases} \tag{4.2}$$

Learning rate is one of the most relevant hyper parameter of the back-propagation algorithm, together with batch size. It permits to reach an optimal minima of the loss function by varying network's weights using the delta rule. It can be summarized using the following formula:

$$\mathbf{\Theta_t} = \mathbf{\Theta_{t-1}} - \eta \frac{1}{m'} \nabla_{\mathbf{\Theta}} \sum_{i=1}^{m'} \mathcal{L}(\mathbf{x}^{(i)}, \mathbf{y^{(i)}}, \mathbf{\Theta_{t-1}}) \tag{4.3}$$

where:

- $\mathbf{\Theta}$ is the set of all weights of the network;
- $\eta$ is the learning rate;
- $\nabla_{\mathbf{\Theta}}$ is the gradient w.r.t. network's weights;
- $m'$ is the size of mini batch of training set and $\mathbf{x}^{(i)}, \mathbf{y^{(i)}}$ are input and output of the network, respectively.

Fig.4.1 shows how back-propagation algorithm works during time, changing distribution of the weights accordingly with the gradient of the loss.

Instead of setting a fixed or logarithmic decrease learning rate for the Stochastic Gradient Descent algorithm, we can apply **ADAM** (**Ad**aptive **A**verage **M**oments) which permits to modify the learning rate w.r.t. the estimation of first and second moments of the gradient.

Adam uses exponentially moving averages as we can see from the following equations:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

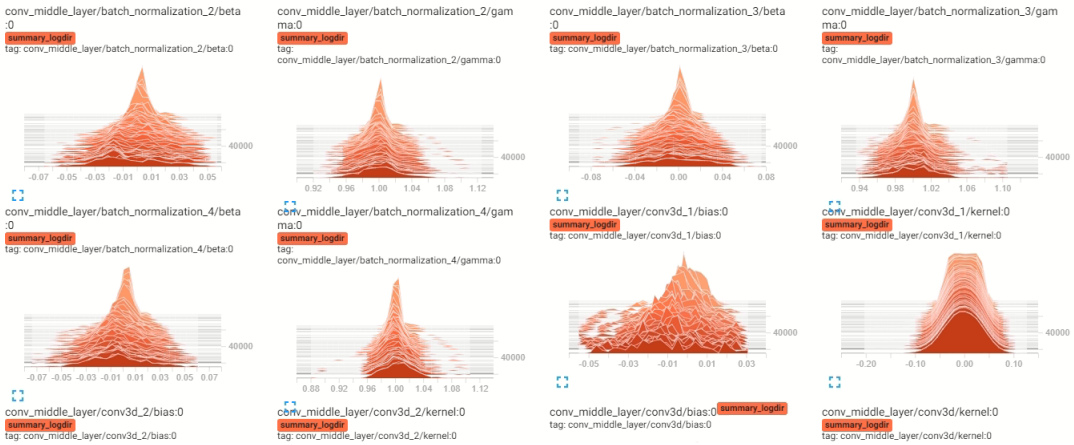



**Figure 4.1:** Display of weights' histograms from the Convolutional Middle Layers.

where $m_t$ and $v_t$ are moving averages of mean and uncentered variance at iteration $t$, $g_t$ is the value of the gradient at time $t$ and $\{\beta_1, \beta_2\}$ are hyperparameters of the Adam algorithm ($\beta_1 = 0.9$ and $\beta_2 = 0.999$ are the most common values, according to state-of-the-art).

At the beginning of the algorithm, we set the two moving averages to 0, i.e. $v_{-1} = 0$ and $m_{-1} = 0$. Our main scope is to have the following propriety:

$$\mathbb{E}[m_t] = \mathbb{E}[g_t]$$
$$\mathbb{E}[v_t] = \mathbb{E}[g_t^2]$$

so expectations of our moving averages can be used as estimators of expectations of gradient and squared gradient $g_t$.

This is not true since our initialization centers our expectations to 0.

For this reason, we are going to use the **unbiased estimators** version. In order to have them, we have to define the following unbiased estimators:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

we can then finally update weights of the network by using the unbiased

estimators using the following formula:

$$w_t = w_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

where $\eta$ is the learning rate and $\epsilon$ is another hyper parameter of the optimizer.

Hence, we can summarize the hyper parameters of the ADAM algorithm with a tuple $(\eta, \beta_1, \beta_2, \epsilon)$.

Moreover, we introduce the *weight decay* parameter which acts as a regularizer of the optimizer.

Model capacity is limited by imposing a norm penalty in the parameters in order to decrease them exponentially during time.

We define the weight decay as follows:

$$\mathcal{L}(\mathbf{x}^{(i)}, \mathbf{y^{(i)}}, \mathbf{\Theta_{t-1}}) = \mathcal{L}(\mathbf{x}^{(i)}, \mathbf{y^{(i)}}, \mathbf{\Theta_{t-1}}) + \frac{\lambda}{2} ||\mathbf{\Theta_{t-1}}||_{\mathbf{2}}^{\mathbf{2}} \tag{4.4}$$

where $\lambda$ is the regularization constant, restricting the size of the weights and avoiding overfitting. The ADAM optimizer with weight decay is defined as ADAMW.

We have to notice that training deep neural network becomes challenging by the fact that distribution of each layer is changing during back-propagation steps, slowing down training process and selecting more precise hyper parameters.

For this main reason, Batch Normalization [24] are concatenate with network layers in order to normalize each training mini-batch input for the next layer, acting as a regularizer of the model.

We introduce, for each neuron activation $x^{(k)}$, a pair of parameters $\gamma^{(k)}$ and $\beta^{(k)}$ which scale and shift the normalized value $\hat{x}^{(k)}$, i.e.:

$$\hat{y}^{(k)} = \gamma^{(k)} \hat{x}^{(k)} + \beta^{(k)} \tag{4.5}$$

where $\hat{y}^{(k)}$ is the normalized output. These parameters are learned during training process in order to fit input distribution so

$$\gamma^{(k)} = \sqrt{Var[x^{(k)}]} \quad , \quad \beta^{(k)} = \mathbb{E}[x^{(k)}] \tag{4.6}$$

An example of their values can be seen in Fig.4.2.

Therefore, we can adopt a more flexible selection of hyper parameters and initialization of the network.

In details, we can:

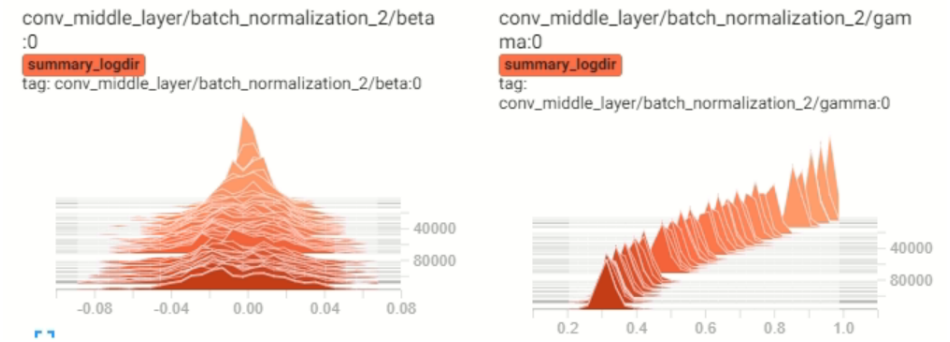
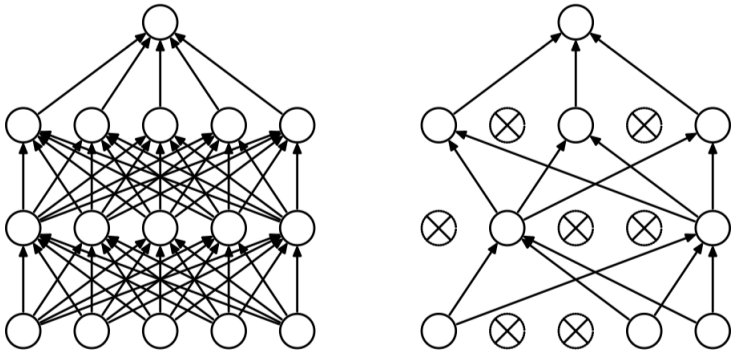



**Figure 4.2:** Values of $\beta^{(k)}$ and $\gamma^{(k)}$ in one Batch Normalization Layer in the Convolutional Middle Layers .

- Increase the learning rate without having ill side effects;
- Avoid adding Dropout [25] layers. They discard a percentage of neurons' activations, randomly, increasing network generalization proprieties.



(a) Standard Neural Net (b) After applying dropout.

**Figure 4.3:** Dropout approach for avoiding overfitting of deep neural networks.

  In our case, Dropout layers add noise to training process without any relevant improvements;

- Speed-up training process since neural networks work well with Gaussian and Uniform weights.

During training, we use ADAMW [23] ($\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1e-07$) optimizer.

Learning rate $\eta$ is 0.001 the first 80 epochs and we decrease the learning rate to 0.0001 for the following 40 epochs. At the end, we apply a learning rate of 0.00001 for the last 30 epochs. For each epoch interval, we use a weight decay of $\lambda = 0.1\eta$.

These value are chosen empirically after some training runs because of the high time complexity of the process. Therefore, they are not optimal and a more detailed hyper parameters research is required.

### 4.1.2 Syntethic Dataset

Since we train the network using a different dataset, we need to apply some changes in order to overcome the following differences:

- KITTI provides ground truth w.r.t. camera FOV. In this specific case, no RGB images are available then no rigid transformations are requested;
- Signals are tiny landmarks in a wide railway environment.

**Signal Detection.** Input range of the point cloud is $[0, 208] \times [0, 28] \times [-3, 1]$ meters along X,Y,Z axis, respectively. We change voxel sizes using $v_D = 0.8, v_H = 0.5, v_W = 0.5 =$, yielding $D' = 5, H' = 56, W' = 416$. Each voxel can contain at the most T=45 3D randomly sampled points.

Convolutional Middle Layers and Region Proposal Network have the same structure as Pedestrian and Cyclist class detection from KITTI dataset.

We decide to perform multiple simulations with different anchor sizes and training hyper parameters. First simulation uses anchor sizes which are $l^p = 0.2, w^p = 0.8, h^p = 0.8$ meters with center $z_c^p = -0.6$ with rotation 0 and 90 degrees w.r.t. Z axis.

Differently, second and third run use two different anchor sizes without rotation which are:

- $l^p = 0.2, w^p = 0.8, h^p = 0.8$ meters with center $z_c^p = -0.6$;
- $l^p = 0.2, w^p = 0.5, h^p = 0.8$ meters with center $z_c^p = -0.6$.

Anchor sizes are selected in order to match the sizes of target signals. In this way, we are able also to classify the detected object by inspecting the values of the 3D bounding box. Anchor matching criteria is described as follow:

$$\begin{cases} 1 \quad if \quad \max IoU > 0.7 \\ 0 \quad if \quad \max IoU < 0.5 \\ \text{Don't Care} \quad if \quad 0.5 \leq \max IoU \leq 0.7 \end{cases} \tag{4.7}$$

Classification of anchors for each voxel is the same as the Pedestrian and Cyclist setup from the KITTI dataset. We modify hyper parameters of the loss

function by setting $\alpha = 1.5$, $\beta = 1$ and $\delta = 4$ in order to reduce rapidly the softmax probabilities of false positive predictions, increasing regression accuracy.

During training, we use ADAMW ($\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1e{-}07$) optimizer using a constant decay learning rate during epochs which is similar to the one used in the KITTI setup.

For the first and second simulation, initial learning rate is 0.001 for the first 80 epochs and we decrease the learning rate to 0.0001 for the next 40 epochs. Finally, we apply a learning rate of 0.00001 for the last 40 epochs.

For each epoch interval, we use a weight decay of $\lambda = 0.1\eta$. Hyper parameters are chosen empirically after some training runs.

The last training process lasts less since overfitting occurs during the first 60 epochs. For this main reason, initial learning rate is 0.001 for the first 5 epochs, we decrease it to 0.0001 for next 15 epochs.

Last 40 epochs are executed using a learning rate of 0.00001. Hence, we train the model for a total of 60 epochs.

## 4.2 Data Augmentation

Data augmentation is a technique which is used to increase the total amount of data for avoiding unsuccessful overfitting of the training set. It allows to introduce some variance to the dataset, acting as a regularizer.

In our setup, we perform three types of augmentation that improve generalization proprieties of the network.

During dataset generation on Matlab, the first augmentation strategy consists in adding uniform noise to dimensions of each 3D box of the ground truth. In a mathematical sense, we sample from a uniform distribution three values $(\hat{l}, \hat{w}, \hat{h}) \sim U[-0.1, 0.1]$ meters and add them to each fixed dimensions of the landmark.

This can be seen as an off-line augmentation method since it is performed outside of the training loop, reducing the complexity of the algorithm.

Since point clouds can be of different sizes (if stereo camera is available in order to increase points' density), we apply a random sampling of LiDAR data on-the-fly during training phase.

Specifically, we remove a percentage of points which is sampled from an uniform distribution $U[0.6, 1]$. In this way, we are able to handle different point cloud sizes without affecting accuracy performances.

Last but not least, we perform a global scaling augmentation. We sample a factor from a uniform distribution $U[0.95, 1.05]$ and then multiply this value with each point coordinates XYZ of the LiDAR data. The same procedure is applied to all ground truth boxes.

All the aforementioned augmentation algorithms improve robustness of the network for detecting landmarks which are different in shapes and locations.

## 4.3 Code Insights

We train the whole network using a single NVIDIA Quadro RTX 5000 which constrains us to use a batch size of 2. Implementation is based on Tensorflow 2.5.0, Python 3.9.5, CUDA 11.3 and TensorBoard 2.5.0. We also used OpenCv 4.5.2. for reading images and writing results and predictions.

*CheckPointManager* class provided by Tensorflow permits to handle multiple checkpoints by keeping some and deleting unneeded ones.

We are able to:

- Store multiple checkpoints and restore them in order not to restart the training loop;
- Preserve all the changes on network weights for performing inference.

Moreover, thanks to *tf.distribute* Tensorflow class, we are able to perform a distributed training on multiple GPUs which permits to speed up computation and increase mini-batch sizes.

Since the whole dataset cannot be fit in the RAM, we deploy a Python *generator* which allows us to:

- Iterate through all the training and validation samples once. Generator state is saved inside the checkpoint, avoiding same samples to be used in the same epoch;
- Allocate memory efficiently. A normal function to return a sequence will create the entire sequence in memory before returning the result. This is an overkill if the number of items in the sequence is very large. Generator implementation of such sequences is memory friendly and is preferred since it only produces one item at a time. In our case, generator yields only one batch when requested.

# Chapter 5

# Evaluation of VoxelNet Performances

## 5.1 Metrics

In this section, we are going to analyze results using the **Average Precision** [27] metric which is used for the KITTI benchmarks. Thus, we introduce some metrics used in object detection tasks.

Let $y_i$ be the score of a predicted box from input $x_i$ belonging to the positive class.

We define the **Recall** as the ratio between all positive samples ranked above a given threshold $\delta$:

$$Recall(\delta) = \mathbb{P}[y_i \geq \delta | x_i \in \mathbb{C}] \tag{5.1}$$

where $\mathbb{C}$ is the set of all positive samples.

Likewise, we define the **Precision** as the ratio between all samples above a given threshold $\delta$ which are from the positive class so:

$$Precision(\delta) = \mathbb{P}[x_i \in \mathbb{C} | y_i \geq \delta] \tag{5.2}$$

We have to notice that either Recall and Precision are parametrized by the threshold value $\delta$. We can introduce a single metric called **mAP** (*mean Average Precision*) which combines the two aforementioned metrics:

$$AP = \frac{1}{11} \sum_{r \in \{0, 0.1, \ldots, 1\}} Precision_{int}(r) \tag{5.3}$$

where $Precision_{int}(r)$ is the interpolated version of the precision function for a recall level $r$, e.g.:

$$Precision_{int}(r) = \max_{\tilde{r}:\tilde{r}\geq r} Precision(\tilde{r}) \tag{5.4}$$

As we can inspect, metric averages 11 different recall values, ranging from 0 to 1 with step size 0.1, for reducing the impact of small variations and then obtaining a more robust metric for object detection purposes.

The intention in interpolating the precision/recall curve in this way is to reduce the impact of the "wiggles" in the precision/recall curve, caused by small variations in the ranking of examples.

This metric was first introduced in the PASCAL VOC dataset challenge. Its workflow is describes as follow:

- We classify each prediction of the network as True Positive, False Positive and False Negative thanks to IoU metric;
- We collect all ranked predictions in descending order of confidence score;
- From top to last ranked prediction, we evaluate recall and precision entries using Eq.5.1 and Eq.5.2. In this way, a value of recall yields a specific level of precision e.g. $Precision(r)$;
- Interpolating as in Eq.5.4, we compute the 11-points average in order to evaluate **AP**.

Since we are interested in evaluating quality of 3D predicted bounding box, we are going to apply the Average Precision metric - with threshold $\delta = 0.25$ - to IoU of volumes ($mAP_{3D}$@0.25) and to Bird's View ($mAP_{BV}$@0.25). Since one class is available, $mAP$ and $AP$ have the same meaning in our scope.

## 5.2 Evaluation on KITTI Dataset

Before we test the network using the artificial dataset, we trained the network using the KITTI dataset for validating the model in the object detection task in wide area point clouds.

As we can see from Tab.5.1, our implementation obtained comparable values to the ones cited from the original paper.

Since test set is not publicly available, we conduct an evaluation by splitting the dataset in training and validation sets which results in 3712 data samples for training and 3769 data samples for validation.

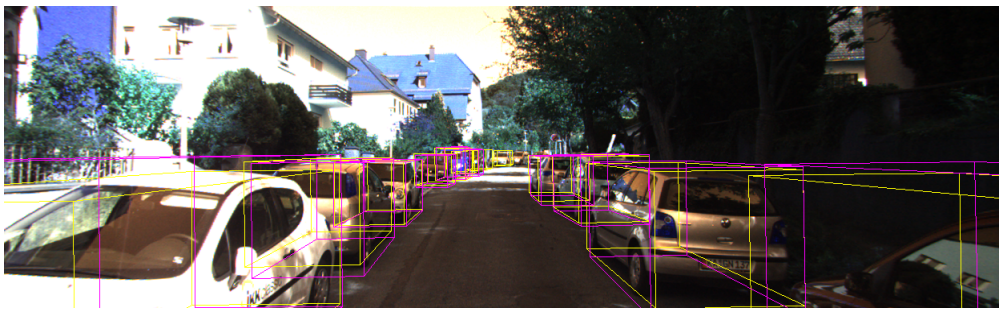


**Figure 5.1:** Example of successful car detection on KITTI dataset using VoxelNet on camera FOV. Purple is the ground truth whereas yellow is the network prediction.

Thanks to this assertion, we use validation data as test set, avoiding misleading results.

We can notice that performances of our implementation are lower with the car class whereas similar or greater with Cyclist and Pedestrian classes.

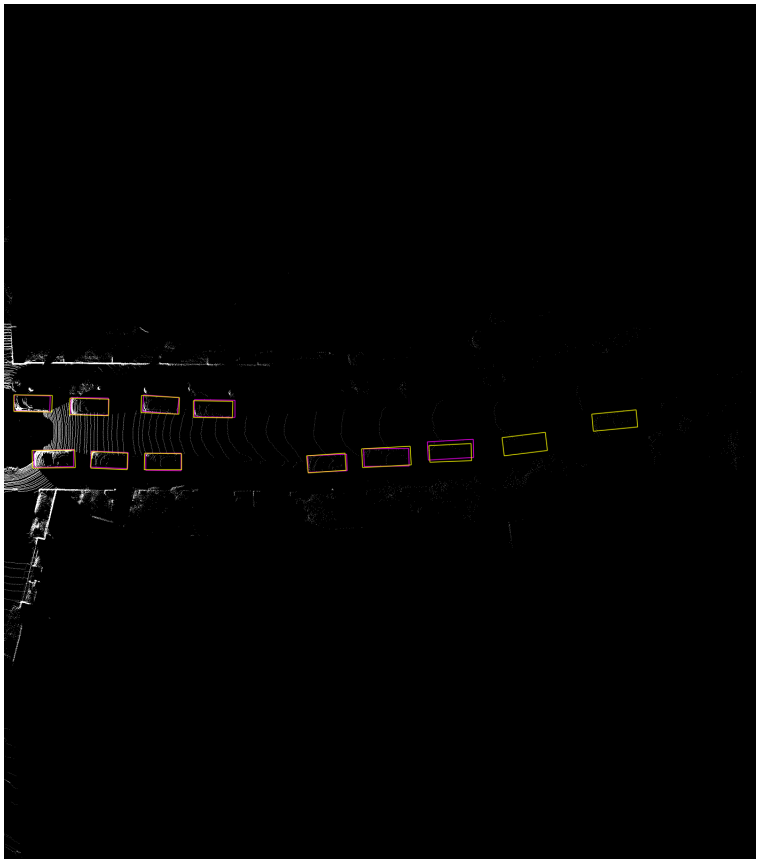


**Figure 5.2:** Example of successful car detection on KITTI dataset using VoxelNet on bird view . Purple is the ground truth whereas yellow is the network prediction.

We can explain this behaviour by analyzing the number of point cloud elaborated for each batch. From the original work, they introduced a batch size of 16 for each class study.

In our case, we increase the detection performances of smaller objects using a batch size of 2.

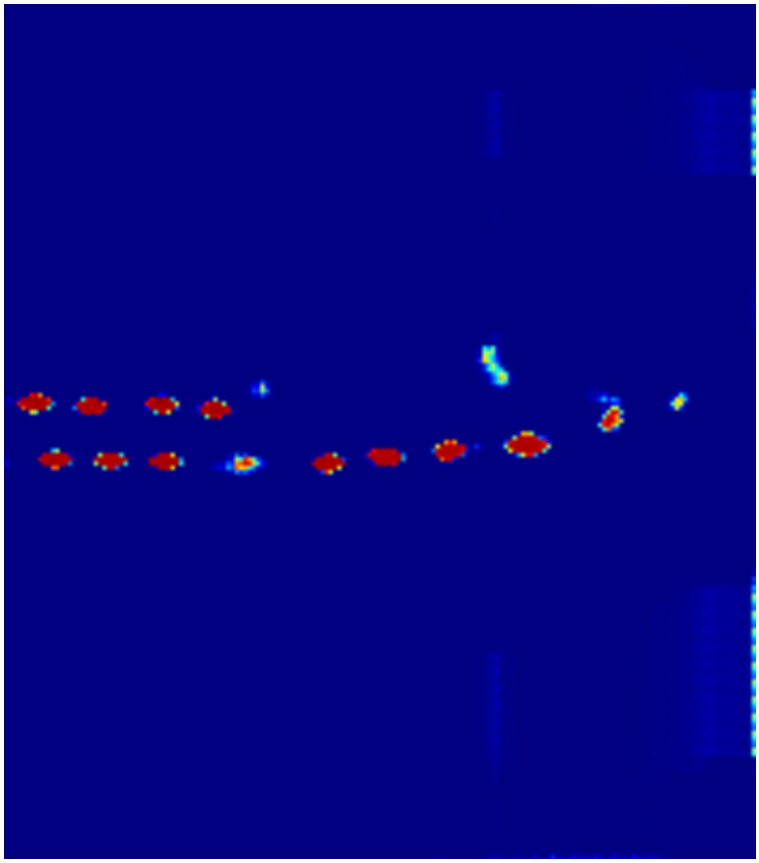


**Figure 5.3:** Example of resulting heat map representing probabilities of the Region Proposal Network using VoxelNet.

This yield that VoxelNet is able to detect smaller object in a more accurate way by using restricted batch sizes. As a counterpart, car detection becomes less precise and reliable.

Reduced batch sizes also extend overall training loop. Each class is trained singularly, as done in the original paper, which needs approximately 6 days of continuous training on the available hardware (one NVIDIA Quadro RTX 5000).

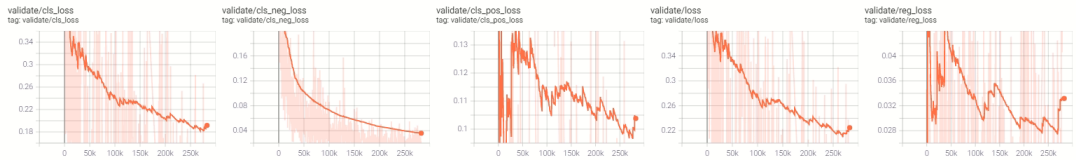
Output of the network can be viewed as follow:

- Camera view of predictions and ground truth (Fig.5.1);
- Bird's Eye View of the point cloud with predictions and ground truth (Fig.5.2);

| Benchmark | Easy | Moderate | Hard |
|---|---|---|---|
| **Our implementation** | | | |
| Car $mAP_{3D}$ | 74.49 | **66.89** | 61.57 |
| Car $mAP_{BV}$ | 80.03 | 79.12 | 73.86 |
| Pedestrian $mAP_{3D}$ | 50.49 | 44.36 | 43.70 |
| Pedestrian $mAP_{BV}$ | 55.16 | 45.48 | 44.78 |
| Cyclist $mAP_{3D}$ | **69.05** | **55.76** | **54.89** |
| Cyclist $mAP_{BV}$ | **77.56** | **69.23** | **59.34** |
| **From Paper** | | | |
| Car $mAP_{3D}$ | **81.97** | 65.46 | **62.85** |
| Car $mAP_{BV}$ | **89.60** | **84.81** | **78.57** |
| Pedestrian $mAP_{3D}$ | **57.86** | **53.42** | **48.87** |
| Pedestrian $mAP_{BV}$ | **65.95** | **61.05** | **56.98** |
| Cyclist $mAP_{3D}$ | 67.17 | 47.65 | 45.11 |
| Cyclist $mAP_{BV}$ | 74.41 | 52.18 | 50.49 |

**Table 5.1:** Our replica and results (mAP in %) from paper to validation results on KITTI.


**(a)** Training Losses of Car class.




**(b)** Validation Losses of Car class.

**Figure 5.4:** Plot of all losses during training.

- Heat Map which shows softmax probabilities of each voxel in bird's eye view (Fig.5.3);

As we can see from Fig.5.4, decreasing learning rate is a successful strategy for capturing more peculiar features from point cloud.

False positive are frequent during predictions, which is clearly evident if we inspect positive classification loss.

It means that, in general, too much predictions are done with high accuracy, reducing precision of the system.

It happens because correct detections of cars on point cloud are labeled as wrong since they are not present in the field of view of the RGB camera.

Improvements on this behaviour can be obtained by fine-tuning the prediction

phase. Changing parameters of NMS and with a longer training run, we can adapt NMS threshold $\delta$ with softmax probabilities of the network.

In conclusion, we can affirm that VoxelNet is able to generalize in wide open space point clouds, moving onward to train the model with the synthetic dataset.

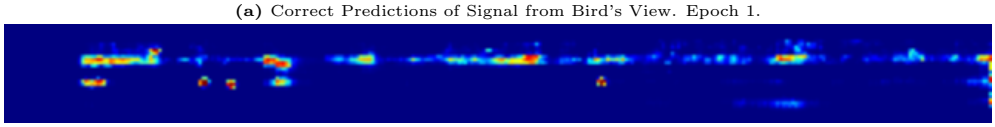
## 5.3 Evaluation on Synthetic Dataset

The main difficulty of the synthetic dataset is to predict 3D bounding box of very small objects w.r.t. the environment.

We can define three training intervals where VoxelNet learns different features of training data, fine-tuning its weights.

As we can see from Fig.5.5, first training epochs consist in discriminating where rail tracks are located, predicting non-zero probabilities on tracks' voxels.

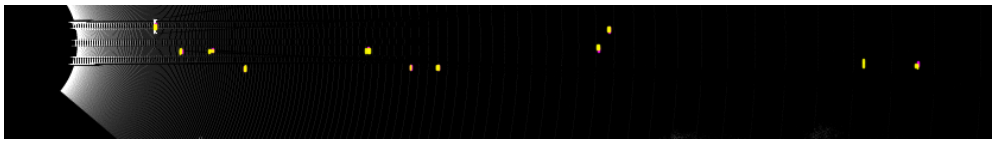
**(a)** Correct Predictions of Signal from Bird's View. Epoch 1.



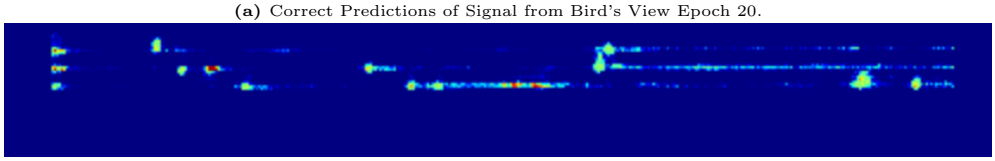
**(b)** Heat map

**Figure 5.5:** Heat map of correct predictions from Bird's View. Epoch 1.

In Fig.5.6 we can inspect that network starts decreasing probabilities of regions where signals are not present. This behaviour occurs during the 20th epoch.



**(a)** Correct Predictions of Signal from Bird's View Epoch 20.



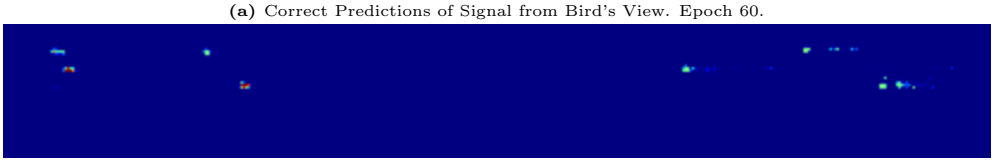
**(b)** Heat map

**Figure 5.6:** Heat map of correct predictions from Bird's View. Epoch 20.

At the end, we can analyze from Fig.5.7 that network is predicting well-defined voxels where objects are located.

Different runs are been performed in order to understand which hyper parameters. We can inspect different AP results in Tab.5.2 where:

**(a)** Correct Predictions of Signal from Bird's View. Epoch 60.



**(b)** Heat map

**Figure 5.7:** Heat map of correct predictions from Bird's View. Epoch 60.

| Benchmark | Run # 1 | Run # 2 | Run # 3 |
|---|---|---|---|
| Signal $mAP_{3D}$@0.25 | 34.67 | 31.98 | **37.68** |
| Signal $mAP_{BV}$@0.25 | 38.49 | 32.27 | **41.51** |

**Table 5.2:** AP on Syntethic Dataset's validation set with different simulations.

- **Run # 1** uses two identical anchor sizes with different Z-rotation (in this case 90 degrees);
- **Run # 2** uses two different anchor sizes without Z-orientation in order to fit different type of signals;
- **Run # 3** uses the same anchors size but reducing the number of points for each voxel i.e. reducing the parameter $T$.

As we can see from the results table, performances can be improved by fine-tuning hyper parameters of the model w.r.t. the target environment.

Prevalent number of not detected objects are far from the sensor because of the reduced number of points. In general, this behaviour occurs when signal or relevant object is at least 150 meters from the LiDAR.

Thanks to the previous results, we can assert that the structure of VoxelNet is able to detect tiny objects, such as signals and rail items, when they are close to the sensor. This happens because of the high points density in the proximity of LiDAR sensor.

# Chapter 6

# Final Considerations

## 6.1 Future Work

In the Machine Learning and Deep Learning field, **hyper parameter optimization** is to objectively search different values for model hyper parameters and choose a subset that results in a model that achieves the best performance on a given dataset. Hyper parameter optimization is required then to get the most out of our machine learning model.

Let the **Search Space** be the set of all possible combinations of hyper parameters of the model. Most common search algorithms of this topic are:

- **Grid Search**: Define a search space as a grid of hyper parameter values and evaluate every position in the grid;
- **Random Search**: We set the Search Space as a bounded domain of hyper parameter values and randomly sample points in that domain;

A possible study is to apply one of the aforementioned method in order to find most suitable hyper parameters for the object detection and classification tasks on point clouds w.r.t. validation loss. Procedure can be challenging because of the time demanding training loop.

VoxelNet can perform detection on point clouds by using a single anchor which is rotated by 0 or 90 degrees around Z axis. Therefore, detection of signals which have very different shape can be challenging.

In order to solve this issue, we can introduce, as in [19], multiple anchors with different sizes. With this implementation, we increase complexity and ability of network, simultaneously. Accordingly, a more focused study is requested for evaluation of the overall system.

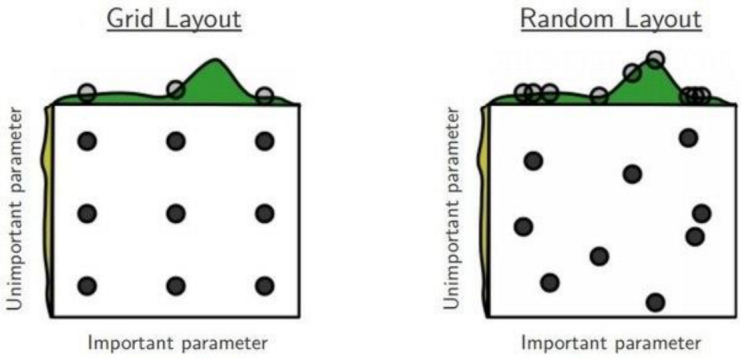



**Figure 6.1:** Example of Grid and Random Search on 2D Search Space.

Adding more augmentation strategies can be a possibility to improve performances of the network. For example, we can perform global rotation of the point cloud, together with ground truth rotation.

Data augmentation permits also to prevent adversarial attacks to the network. An example of attack can be fooling neural networks of self-driving cars by placing stickers on traffic signs. We can inspect an example of this behaviour in Fig.6.2.

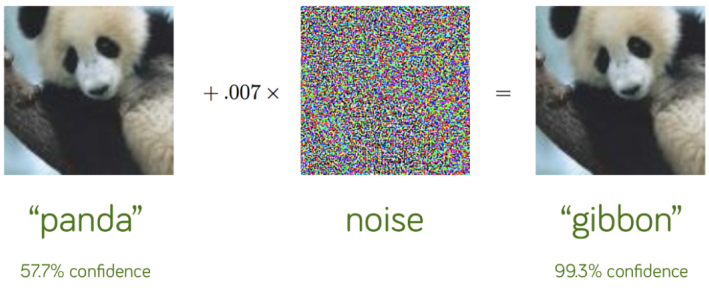



**Figure 6.2:** Example of adversarial attack on a image classification network.

This vulnerability has been discovered recently, so researchers are still exploring possible ways to make the models more robust.

Two possible approaches are:

- **Adversarial Training**: Neural network is optimized in order to achieve high level of accuracy on adversarially-perturbed training examples
- **Randomized smoothing**: Neural network is smoothed by convolution with Gaussian Noise [30], i.e:

$$\hat{f}(\mathbf{x}) = \mathbb{E}_{\epsilon \sim \mathcal{N}(0,\sigma^2 I)}[f(\mathbf{x}+\epsilon)] \tag{6.1}$$

where $\hat{f}(\cdot)$ is the smoothed network, $\epsilon$ is a sample from a isotropic Gaussian noise, $\mathbf{x}$ is the input sample and $f(\cdot)$ is the original network.

Another future work can be network size's compression. Neural network pruning techniques can reduce the parameter counts of trained net- works by over 90%, decreasing storage requirements and improving computational performance of inference without compromising accuracy. There is not a definite state-of-the-art method but we have to cite the most famous ones:

- **Pruning and Quantization** [31]: Removing weights close to zero and retrain the network. We iterate the process until we reach a good trade-off between compression ratio and loss of performances. After this, we quantize network weights for additional compression;
- **Knowledge Distillation** [32]: It is possible to compress the knowledge in an ensemble into a single model which is much easier to deploy, reducing weights' sizes and inference time.

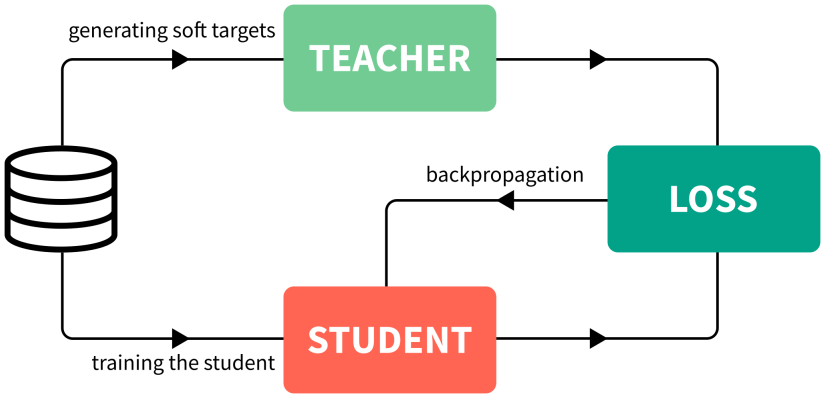



**Figure 6.3:** Teacher-Student paradigm for Knowledge Distillation.

It can be implemented by using the teacher-student paradigm as we can see from Fig.6.3

Unfortunately, no test data is furnished by donors for obtaining final evaluation statistics. However, thanks to previously described procedure, we are able to modify the train simulator in relation to real data, increasing performances and generalizations proprieties of the model.

Last but not least, we are going to train the model with an additional GPU. Thanks to TensorFlow, code is already structured for parallel GPU computing.

It allows us to increase batch sizes - upgrading Batch Normalization layers' functionalities - and reduce training time for better hyper parameters fine-tuning.

## 6.2 Conclusion

We presented the problem statement of object detection in point clouds. We described a possible solution called VoxelNet which transforms sparse point cloud data in a regular shape i.e. voxels.

We inspected some dataset that can be used in automotive scenarios with their pros and cons and the decision to create an artificial dataset using the integration between Unreal Engine™ and Matlab™.

We deeply introduced the train simulator and its functionalities. We illustrated also how point clouds had been collected in order to organize the manufactured dataset.

We provided training details of the network, with hyper parameters selections and motivations.

We explained results from KITTI evaluation and synthetic dataset, showing impressive network skills by only observing artificial data.

Finally, we showed that our Proof of Concept can be successful for detecting railway props using only sparse LiDAR data.

# List of Tables

1.1 Pros and cons of LiDAR sensor. . . . . 6

5.1 Our replica and results (mAP in %) from paper to validation results on KITTI. . . . . 57

5.2 AP on Syntethic Dataset's validation set with different simulations. 59

# List of Figures


1.1 3D Object detection task in the automotive environment from CIA-SSD [15] paper . . . . . 2
1.2 VGG16 deep architecture. . . . . 2
1.3 Frustum PointNet architecture. . . . . 3
1.4 CIA-SSD architecture. . . . . 4
1.5 LIVOX™ Horizon sensor with its functional characteristics. . . . . 5
1.6 Sensor Geometry from lateral view. . . . . 8
1.7 Sensor Geometry from top view. . . . . 8
1.8 **VOLIERA** logo. . . . . 9
1.9 Detecting railways using only raw point clouds. . . . . 10
1.10 Computation of relative position with more than two detected landmarks. . . . . 10
1.11 Rosten & Drummond [13] algorithm for fast feature detection on images. . . . . 12
1.12 Flow chart of the **VOLIERA** project. . . . . 12
1.13 Proposed framework. . . . . 13

2.1 KITTI360 setup and example of obtained point cloud. . . . . 15
2.2 KITTI360 example of point cloud with semantic information. . . . 16
2.3 KITTI360 example of object ground truth. . . . . 16
2.4 KITTI360 example of noisy signal. . . . . 17
2.5 List of files inside the RailSem19 directory. . . . . 18
2.6 Example of image with semantic segmentation and object detection labels. . . . . 19
2.7 Map between bgr colors of uint8 images and labels. . . . . 19
2.8 Structure of ground truth inside json files. . . . . 20
2.9 Example of semantic segmentation map of a sample image. . . . . 21
2.10 Example of left RGB image from KITTI Dataset. . . . . 21

2.11 KITTI folder structure for 3D Object Detection suite. . . . . 22
2.12 Example of KITTI point cloud with 3D bounding boxes. . . . . 23
2.13 Example of left RGB with 3D bounding boxes. . . . . 23
2.14 Main window of Unreal Engine™ software. . . . . 24
2.15 Example of 3D model of a Railway support sign with a traffic light. 26
2.16 Blueprint scripting for generating track rails. . . . . 26
2.17 Blueprint scripting for generating landmark near track rails. . . . 27
2.18 Estimation of LiDAR distortion on Matlab™ simulation. . . . . 28
2.19 Snapshot of the Unreal Engine's viewport during a simulation run. 29
2.20 Example of synthetic point cloud. . . . . 30
2.21 Syntethic point cloud with annotations. . . . . 30

3.1 Structure of the point cloud encoder PointNet. . . . . 31
3.2 3D-BoNet architecture. . . . . 32
3.3 VoxelNet architecture. . . . . 33
3.4 Voxelization of the point cloud. . . . . 34
3.5 Single VFE encoding process. . . . . 35
3.6 Region Proposal Network structure. . . . . 37
3.7 Huber Loss w.r.t.$\delta$ parameter. . . . . 39
3.8 Example of binary mask displaying set of voxels where black voxels represent one object. . . . . 40
3.9 Example of binary mask displaying set of voxels where white voxels represent one object. . . . . 40

4.1 Display of weights' histograms from the Convolutional Middle Layers. 46
4.2 Values of $\beta^{(k)}$ and $\gamma^{(k)}$ in one Batch Normalization Layer in the Convolutional Middle Layers . . . . . 48
4.3 Dropout approach for avoiding overfitting of deep neural networks. 48

5.1 Example of successful car detection on KITTI dataset using VoxelNet on camera FOV. Purple is the ground truth whereas yellow is the network prediction. . . . . 55
5.2 Example of successful car detection on KITTI dataset using VoxelNet on bird view . Purple is the ground truth whereas yellow is the network prediction. . . . . 55
5.3 Example of resulting heat map representing probabilities of the Region Proposal Network using VoxelNet. . . . . 56
5.4 Plot of all losses during training. . . . . 57

5.5 Heat map of correct predictions from Bird's View. Epoch 1. . . . 58
5.6 Heat map of correct predictions from Bird's View. Epoch 20. . . . 58
5.7 Heat map of correct predictions from Bird's View. Epoch 60. . . . 59

6.1 Example of Grid and Random Search on 2D Search Space. . . . . 62
6.2 Example of adversarial attack on a image classification network. . 62
6.3 Teacher-Student paradigm for Knowledge Distillation. . . . . 63

# Acknowledgements

# References


[1] E. Arnold, O.Y. Al-Jarrah, M. Dianati, S. Fallah, D. Oxtoby and A. Mouzakitis, *A Survey on 3D Object Detection Methods for Autonomous Driving Applications*, IEEE Transactions on Intelligent Transportation Systems, Volume: 20, Issue: 10, Oct. 2019

[2] K. Simonyan and A. Zisserman , *Very Deep Convolutional Networks for Large-Scale Image Recognition* in San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings. 2015

[3] X. Chen, K. Kundu, Z. Zhang, H. Ma, S. Fidler, and R. Urtasun, *Monocular 3D object detection for autonomous driving*, in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2016, pp. 2147–2156.

[4] A. Mousavian, D. Anguelov, J. Flynn, and J. Košecká, *3D bounding box estimation using deep learning and geometry*, in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jul. 2017, pp. 5632–5640.

[5] Y. Xiang, W. Choi, Y. Lin, and S. Savarese, *Data-driven 3D voxel patterns for object category recognition*, in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., Jun. 2015, pp. 1903–1911.

[6] J. Deng, W. Dong, R. Socher, L. Li, K. Li, L. Fei-Fe, *ImageNet: A large-scale hierarchical image database*, in 2009 IEEE Conference on Computer Vision and Pattern Recognition, 20-25 June 2009

[7] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, *Multi-view convolutional neural networks for 3D shape recognition*, in Proc. IEEE Int. Conf. Comput. Vis. (ICCV), Dec. 2015, pp. 945–953.

[8] B. Li, T. Zhang, and T. Xia, *Vehicle detection from 3D Lidar using fully convolutional network*, in Proc. Robot., Sci. Syst. XII, AnnArbor, MI, USA, Jun. 2016. [Online]. Available: http://www.roboticsproceedings.org/rss12/

[9] C. R. Qi, H. Su, K. Mo and L. J. Guibas, *PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation*, `http://arxiv.org/abs/1612.00593`, CoRR, 2016.

[10] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, *Multi-view convolutional neural networks for 3D shape recognition*, in Proc. IEEE Int. Conf. Comput. Vis. (ICCV), Dec. 2015, pp. 945–953.

[11] B. Li, T. Zhang, and T. Xia, *Vehicle detection from 3D Lidar using fully convolutional network* ,in Proc. Robot., Sci. Syst. XII, AnnArbor, MI, USA, Jun. 2016. [Online]. Available: http://www.roboticsproceedings.org/rss12/

[12] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas, *Frustum PointNets for 3D object detection from RGB-D data* ,in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2018, pp. 918–927.

[13] E. Rosten , T. Drummond, *Machine Learning for High-Speed Corner Detection.* in Leonardis A., Bischof H., Pinz A. (eds) Computer Vision – ECCV 2006. ECCV 2006. Lecture Notes in Computer Science, vol 3951. Springer, Berlin, Heidelberg.

[14] S. Rusinkiewicz and M. Levoy, *Efficient variants of the ICP algorithm* in Proceedings Third International Conference on 3-D Digital Imaging and Modeling, 28 May - 01June 2001.

[15] W. Zheng, W. Tang, S. Chen, L. Jiang and C. Fu, *CIA-SSD: Confident IoU-Aware Single-Stage Object Detector From Point Cloud*, `https://arxiv.org/abs/2012.03015`, CoRR, 2020.

[16] B. Yang, J. Wang, R. Clark, Q. Hu, S. Wang, A. Markham and N. Trigoni, *Learning Object Bounding Boxes for 3D Instance Segmentation on Point Clouds*, `http://arxiv.org/abs/1906.01140` , CoRR, 2019.

[17] M. Enzweiler and D. M. Gavrila, *A Multilevel Mixture-of-Experts Framework for Pedestrian Classification* in IEEE Transactions on Image Processing, vol. 20, no. 10, pp. 2967-2979, Oct. 2011, doi: 10.1109/TIP.2011.2142006.

[18] Y. Zhou and O. Tuzel, *VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection*, `http://arxiv.org/abs/1711.06396` , CoRR, 2017

[19] S. Ren, K. He, R. Girshick, J. Sun, *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Network* in NIPS'15: Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1December 2015 Pages 91–99

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

[21] J Xie, M. Kiefel, M. Sun and A. Geiger, *Semantic Instance Annotation of Street Scenes by 3D to 2D Label Transfer* in Conference on Computer Vision and Pattern Recognition (CVPR), 2016

[22] O. Zendel, M. Murschitz, M. Zeilinger D.Steininger, S. Abbasi and C. Beleznai, *RailSem19: A Dataset for Semantic Rail Scene Understanding* in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, 2019

[23] D. P. Kingma, J. Ba, *Adam: A Method for Stochastic Optimization* in 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, 2015

[24] S. Ioffe and C. Szegedy, *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*, `http://arxiv.org/abs/1502.03167` in CoRR 2015

[25] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, *Dropout: A Simple Way to Prevent Neural Networks from Overfitting* in Journal of Machine Learning Research, pg 1929-1958, n. 56, v.15, 2014.

[26] S. Manivasagam, S. Wang, K. Wong, W. Zeng, M. Sazanovich, S. Tan, B. Yang, W. Ma, R. Urtasun, *LiDARsim: Realistic LiDAR Simulation by Leveraging the Real World* in CoRR, 2020

[27] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, A. Zisserman, *The PASCAL Visual Object Classes (VOC) Challenge* in International Journal of Computer Vision, Volume 88, Issue 2 June 2010, pp 303–338

[28] I. Loshchilov and F. Hutter, *Decoupled Weight Decay Regularization* in ICLR `https://arxiv.org/pdf/1711.05101.pdf`, 2019

[29] Tensorflow 2.50 documentation `https://www.tensorflow.org` Last access: 23/07/2021

[30] J. M. Cohen, E. Rosenfeld, J. Z. Kolter, *Certified Adversarial Robustness via Randomized Smoothing*, `https://arxiv.org/abs/1902.02918`, CoRR, February 2019

[31] Y. LeCun, J. S. Denker, and S. A. Solla, *Optimal Brain Damage* in Advances in Neural Information Processing Systems, 2.,598-605, 1990

[32] G. Hinton, O. Vinyals, J. Dean, *Distilling the Knowledge in a Neural Network* in NIPS Deep Learning and Representation Learning Workshop, `http://arxiv.org/abs/1503.02531`, 2015