A Deep Learning Approach for LiDAR Resolution-Agnostic Object Detection

Ruddy Theodose, Dieumet Denis, Thierry Chateau, Vincent Frémont, Paul Checchin

A Deep Learning Approach for LiDAR Resolution-Agnostic Object Detection

Ruddy Théodose[1,2], Dieumet Denis[1], Thierry Chateau[2], Vincent Frémont[3] and Paul Checchin[2]

***Abstract*—Existing neural network-based object detection approaches process LiDAR point clouds trained from one kind of LiDAR sensor. In the case of a different point cloud input, the trained network performs with less efficiency, especially when the given point cloud has low resolution. In this paper, we propose a new object detection approach, which is more resilient to variations in point cloud resolution. Firstly, layers from the point cloud are randomly discarded during the training phase in order to increase the variability of the data processed by the network. Secondly, the obstacles are described as Gaussian functions, grouping multiple parameters into a single representation. A Bhattacharyya distance is used as a loss function. This approach is tested on a LiDAR-based network and on an architecture using camera and LiDAR sensors. The networks are trained exclusively on the KITTI dataset and tested on Pandaset and the nuScenes Mini dataset. Experiments show that our method improves the performance of the tested networks on low-resolution point clouds without decreasing the ability to process high-resolution data.**

***Index Terms*—Deep Learning, 3D Object Detection, Sensor Fusion, Intelligent Vehicle, Camera Sensor, LiDAR Sensor.**

I. Introduction

Autonomous driving in urban environments still remains a tremendous challenge. The autonomous vehicle must simultaneously manage multiple aspects of perception such as scene analysis, traffic sign recognition, or moving object localization. It is significant to control these aspects before deciding which actions can be engaged to preserve the safety of other users and the comfort of the passengers. 3D object detection for autonomous vehicles is an active research topic since identifying the parts of the scene that can interfere with the vehicle trajectory is a crucial task within the navigation pipeline. In this field, most works mainly rely, if not exclusively, on LiDAR sensor data.

LiDAR sensors collect 3D point clouds depicting the surroundings in great detail. However, the data structure of such 3D point clouds is usually not unique and normalized, and the spatial positioning of the points is also irregular. Moreover, the information provided on the appearance of surfaces is inadequate. By contrast, 2D cameras deliver structured data with high resolution as per the spectrum captured, e.g., color information, infrared. These characteristics make them particularly adapted for classification or identification tasks, even on small or noisy images. However, the loss of depth information and the narrow field of view makes the full 3D reconstruction of the scene geometry difficult. Over the past years, deep neural networks have proven their efficiency in perception tasks by using camera images. Most methods aim to extend these techniques to 3D point clouds. However, due to the differences between 2D images and 3D LiDAR information, the application of classical neural networks on 3D point cloud data is still an open problem. For example, some methods transform the sparse data into dense and structured grids through voxelization that can then be used by operators designed initially for image processing. Other works exploit architectures based on PointNet [1], [2] with the aim to directly process the 3D point cloud by utilizing the geometrical relationships between points and their neighborhood [3], [4]. No sensor is adapted for all the situations. Hence, autonomous vehicles are usually equipped with at least two types of sensors [5], [6]. Currently, the majority of the 3D object detection methods are purely LiDAR-based [7], [8], [9]. Fewer methods employ the data fusion of the two sensors [10], [11] because of their different data structures and constraints such as calibration, registration of 2D/3D images, etc. However, some methods exploit the outputs of 2D processing techniques to constraint the possible locations of the objects in the scene. For example, the helpful priors for the 3D detection task are the utilization of semantic image segmentation [12] or 3D frustums generated by 2D image detection [13].

While researches focused on the design of architectures that allows to maximize accuracy, most of the existing models are trained from a given set of LiDAR parameters and do not generalize well if tested on another set. Even though solutions have been developed for camera sensors (like domain adaptation [14]), to the best of our knowledge, a few works have analyzed the effects on pre-trained systems of 3D scans issued from different resolutions of LiDAR sensors. Therefore, we believe that this topic is important for three reasons. Firstly, to generalize a detection method for emerging LiDAR sensors. Secondly, we see that most of the existing systems are trained on restricted geographical areas and for a specific set of sensors rendering different networks useless if employed on data outside its training distribution. Finally, the annotation of new data is an expensive and time-consuming process. Moreover, a network performing decently on new data, even if the results are not perfect, can potentially speed up the annotation process.

In this paper, we study the consequences of input distribution changes on networks and propose an approach to alleviate these effects. First, we compare how methods exploiting both sensors and methods using only 3D point clouds react to

[1] Sherpa Engineering, R&D Department, 333 Avenue Georges Clemenceau, 92000 Nanterre, France. [r.theodose, d.denis]@sherpa-eng.com

[2] Université Clermont Auvergne, CNRS, Institut Pascal, F-63000 Clermont-Ferrand, France. firstName.lastName@uca.fr

[3] Centrale Nantes, LS2N, UMR CNRS 6004, Nantes, France. vincent.fremont@ls2n.fr

variations in resolution. Our study revolves around a network inspired by PointPillars [15] and MVX-Net [11]. However, the main contribution concerns the way the network is trained. We propose a simple augmentation procedure in order to increase the variability of inputs seen in the training stage and then to make networks more robust to point cloud reduction. We have also developed a new representation for the obstacles and its corresponding loss function. Each target is represented by a Gaussian function. This loss function focuses on the occupation of the obstacle instead of regressing parameters which can only be accurately estimated if sufficient data are available. The performances of our approach are studied across data from three datasets: the KITTI dataset, the Pandaset dataset and the nuScenes Mini dataset. The results show that our approach can help a network to converge better and to produce accurate detections even if the input point cloud is extremely sparse.

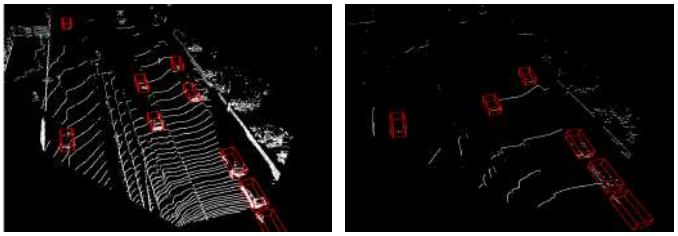


Fig. 1: The same network is able to process point clouds with different resolutions. The missing detections on the sparse case come from the lack of 3D points.

In this paper, we propose the following contributions:

- An extensive bibliographical study of the state-of-the-art methods for 3D object detection in Section II. According to the type of data used (images, 3D LiDAR points or both), a summary of published networks is presented and discussed;
- A new training procedure to increase the variability of the inputs during the training stage;
- A novel way to represent the obstacles as normal distribution is introduced in Section IV-A to use statistical distances and to study the similarities between a target and a prediction in terms of occupation instead of regressing directly the box parameters;
- The evaluation of our approach is conducted on multiple datasets. The behaviors of our approach over inputs never seen during the training stage are detailed in Section VII.

II. Related work

In this section, some relevant networks aiming at 3D object detection on point clouds and RGB images are first introduced and then secondly, methods targeting the management of point clouds at different resolutions are presented.

A. 3D Object Detection Methods

We split the methods available in the literature into three groups. The first group includes networks working exclusively with 3D point clouds. Techniques from the second group only exploit images. The last group of methods process both types of data. Table I summarizes these approaches by mentioning the raw input data used and the main paradigms existing in the literature. These three groups of 3D detection methods are analyzed below.

1) 3D Object Detection using 2D Images: Camera-based 3D object detection methods are generally less accurate than LiDAR-based methods because of the lack of depth information in the image. Hence, most of them rely on geometrical constraints priors. Here, we focus only on monocular camera methods. In [16], a 3D grid is populated with features from the input image by projecting voxels on the image using the camera parameters. The resulting 3D grid is then collapsed on the vertical axis to reduce the computational cost. The authors of [19] train an encoder-decoder architecture for depth estimation. The resulting intermediate features (after the encoder, before the decoder) are then used for 3D object detection. Monocular 3D Region Proposal Network [17] proposes a two-branch architecture, where one branch uses regular convolutions across the whole image to estimate global features. The other splits the image into row blocks then applies kernels, distinct for each bin. These convolution kernels are called "depth aware convolutions". The primary assumption is that each row block can be associated with a discretized depth due to perspective projection (for example, lower rows often represent road and close objects) and then a different operation for each block is applied. In [18], a 2D object detection method is followed by a 3D object detection one. However, their loss function splits the box parameters into groups to simplify the optimization process.

2) Point Cloud-based Methods: This group only uses the 3D point clouds provided by LiDAR sensors as inputs. Methods belonging to this group can be split into two subgroups: grid-based methods and point-based methods.

Grid-based methods. The main idea of grid-based methods consists in turning the 3D point cloud into structured data to allow the use of concepts and operators which were successfully applied in 2D image processing such as convolutions. The main drawback is that the precision depends on the discretization of the grid. Some methods are based on existing 2D architectures to extract detections from BEV pseudo images. BirdNet [20] and Complex YOLO [21] are based on Faster-RCNN [31] and YOLO [32], [33], respectively. Their corresponding networks are applied on the 3-channel pseudo-images by concatenating a height map, a reflectance map, and a density map. VoxelNet [7] was a major milestone. In their paper, the authors describe an architecture including encoding voxels features through PointNet-like methods. The encoded grid is given to 3D dense convolutions and afterwards to 2D convolutions in order to return the selected prior boxes and their corrections. However, the use of 3D dense convolutions on a large 3D grid is slow and computationally expensive. SECOND [8] rectifies this by applying sparse convolutions [34] on the grid. PointPillars [15] extends the concept by using, directly, columns in a 2.5D pseudo image instead of voxels in a 3D grid in order to speed up the computation. Voxel-FPN [35], inspired by the Feature Pyramid Networks [36], uses voxel grids with different resolutions to encode voxels at different scales. Anchor-free methods,

TABLE I: Summary of published networks aiming at 3D object detection from point clouds and/or RGB images. An overview of the state of experimental evaluation of methods useful for autonomous driving.

Reference	Raw Input Data	Raw Data Processing	Overview	Object Types	Dataset(s) used
OFT [16]	RGB Image	–	Resnet Backbone, projection of 3D voxels on image feature maps, each 3D assigned with features inside the projection, then Voxelnet-like processing	Cars	KITTI
M3D-RPN [17]	RGB Image	–	DenseNet Backbone then 2 branches: (1) global feature extraction on the whole image, (2) local "Depth-Aware" feature extractions. Horizontally sliced image. Each slice is fed to its own convolution	Car, Pedestrian, Cyclist	KITTI
MonoDIS [18]	RGB Image	–	ResNet Backbone, 2D and 3D heads, box parameters divided into groups for better convergence	Car, Pedestrian, Cyclist	KITTI
CubifAE-3D [19]	RGB Image	–	Multiple trainings: (1) depth estimation through encoder-decoder, (2) depth latent space for 3D detection	Car, Pedestrian, Cyclist and others	KITTI, nuScenes, KITTI Virtual 2
BirdNet [20]	Point Cloud	Discretization into BEV 3-channel image (height, density, reflectance)	Faster RCNN-like network	Car, Pedestrian, Cyclist	KITTI
Complex-YOLO [21]	Point Cloud	Discretization into BEV 3-channel image (height, density, reflectance)	YOLO-like network, complex angles for regression	Car, Pedestrian, Cyclist	KITTI
VoxelNet [7]	Point Cloud	3D voxelization	Learned voxel encoding for each voxel, 3D convolutions to flatten on the vertical axis then RPN	Car, Pedestrian, Cyclist	KITTI
SECOND [8]	Point Cloud	3D voxelization	Learned voxel Encoding, sparse 3D convolution to reduce the computational burden, RPN	Car, Pedestrian, Cyclist	KITTI (nuScenes thereafter)
PointPillars [15]	Point Cloud	Column voxelization (no slices on vertical axis)	Learned voxel encoding, 2D convolutions, RPN	Car, Pedestrian, Cyclist	KITTI, nuScenes
OHS [22]	Point Cloud	3D voxelization, voxels represented by the mean content	Objects as sets of hotspots (non-empty voxels belonging to an object), architecture inspired by SECOND or PointPillars	Car, Pedestrian, Cyclist and others	KITTI, nuScenes
PointRCNN [3]	Point Cloud	–	Two-stage method: (1) 3D proposal generation through point cloud segmentation, each foreground point generating its own 3D proposal, (2) 3D box refinement	Car, Pedestrian, Cyclist	KITTI
Fast Point R-CNN [4]	Point Cloud	3D voxelization	Two-stage method: (1) 3D region proposal through VoxelNet architecture, (2) pooling on VoxelNets feature maps, fusion with the corresponding region of point cloud, the augmented point cloud is then processed by a refinement network	Car, Pedestrian, Cyclist	KITTI
CenterPoint [23]	Point Cloud	3D or column voxelization	Objects represented by their center on the classification map, architecture inspired by VoxelNet & PointPillars	nuScenes classes	nuScenes
PV-RCNN [9]	Point Cloud	3D voxelization + furthest point sampling to extract keypoints	Two-stage detection: (1) architecture inspired by SECOND for the region proposal, the 3D feature maps sampled at multi-scales using the computed keypoints, (2) RoI-Grid Pooling on the keypoints then Refinement Network	Car, Cyclist	KITTI
Part-A^2 Net [24]	Point Cloud	–	Two-stage detection: (1) point cloud semantic segmentation, part estimation for positive 3D points then 3D proposals, (2) point cloud pooling then refinement network to aggregate the estimated parts	Car, Pedestrian, Cyclist	KITTI
MV3D [25]	Point Cloud, RGB image	Discretization into BEV 3-channel image (height, density, reflectance) and Cylindrical projection	Two-stage detector: (1) BEV generates 3D proposals, (2) 3D proposals are projected on the BEV, the RGB image and on the cylindrical view, features from the 3 feature maps (each view) are pooled then merged to refine the boxes	Car	KITTI
Frustum PointNet [13]	Point Cloud, RGB image	–	The 2D detector on the image to create 3D frustums, for each result, the 3D points inside a frustum used to estimate the corresponding The 3D box with an architecture similar to PointNet++	Car, Pedestrian, Cyclist	KITTI
AVOD [10]	Point Cloud, RGB	Discretization into BEV 3-channel image (height, density, reflectance)	Two-stage detector: (1) BEV and RGB images merged to generate 3D proposals (2) 3D proposals are projected on the BEV and the RGB image feature maps, the corresponding features are pooled then merged to refine the 3D boxes	Car, Pedestrian, Cyclist	KITTI
PointFusion [26]	Point Cloud, RGB image	–	2D detector to extract 2D crop and 3D frustum, 2D crop and corresponding point cloud processed by a ResNet and a PointNet, respectively. Features merged to estimate a refined 3D box	Car, Pedestrian, Cyclist	KITTI, SUN-RGBD
ContFusion [27]	Point Cloud, RGB image	–	The 2D feature extraction, feature sampling through continuous convolution with the point cloud, reprojection of the sampled features on a BEV then 3D box prediction	Car	KITTI, TOR4D (private)
IPOD [28]	Point Cloud, RGB image	2D Semantic Segmentation	3 parts: (1) using the 2D segmentation, all the 3D background points are discarded and proposals are generated from the remaining points, (2) a backbone network extracts local and global features from the whole point cloud, (3) the features are extracted from the proposals and used in a refinement network	Car, Pedestrian, Cyclist	KITTI
Frustum ConvNet [29]	Point Cloud, RGB image	–	The 2D detector on the image to create 3D frustums, for each result, point cloud part inside the frustum is kept and voxelized along the frustum axis, convolutions are used along this axis to estimate the 3D box	Car, Pedestrian, Cyclist	KITTI
PointPainting [12]	Point Cloud, RGB image	2D Semantic Segmentation	Point features (3D coordinates, reflectance) concatenated with the score given by 2D segmentation. Experiments on multiple detectors (PointRCNN, PointPillars, etc.)	Car, Pedestrian, Cyclist	KITTI, nuScenes
MVX-Net [11]	Point Cloud, RGB image	3D voxelization	VoxelNet used for the 3D detection and Backbone Faster RCNN for the 2D feature maps, 2 fusion methods tested: (1) 2D features sampled on point level and merged before voxel encoder, (2) 2D features sampled on voxel level and merged after voxel encoder	Car	KITTI
MMF [30]	Point Cloud, RGB image	3D voxelization	Two-stage detector: (1) RGB image processed to generate intermediate feature maps and a depth map. Maps are merged with LiDAR features to generate 3D proposals. (2) LiDAR and RGB feature maps are pooled and merged to refine the estimated boxes	Car (main)	KITTI, TOR4D (private)

assimilating object detection to keypoint detection [37], [38] have also been transposed to 3D object detection [22], [23]. The main goal of these methods is not to change the input data representation but the output format, as detections are no more related to prior boxes.

Point-based methods. Point-based methods aim to deal with raw point clouds at some stage directly, in contrast to the grid-based approach. This paradigm can be applied for the whole method or jointly with grid representations. Moreover, most of these methods are two-stage detectors. PointRCNN [3] adopts a two-stage approach: A first stage generates proposals from foreground point segmentation followed by the refinement of the proposals to estimate targeted boxes. Fast PointRCNN [4] and PV-RCNN [9] employ a VoxelNet-like to generate proposals processed by PointNet-like networks. Part-A^2 [24] initially estimates object parts (top left, bottom right, etc.) from point cloud to improve the 3D box refinement stage.

3) 3D Object Detection using Multi-Sensor Fusion: This section focuses on camera-LiDAR data fusion to localize surrounding objects, in contrary to the previously mentioned methods that exploit only one sensor. These methods can be divided into two subgroups. The first makes use of both sensor data within the same system. The second uses outputs from existing and pre-trained 2D methods as priors or inputs for methods that focus on point clouds. These are introduced in the following paragraphs.

Parallel flows. These methods generally use images as a second input to exploit the complementary nature of data. MV3D [25] and AVOD [10] are methods that simultaneously process images and point clouds to extract intermediate proposal. MV3D [25] generates 3D proposals from a LiDAR scan and then projects them onto the LiDAR BEV. Features inside the projection of each modality are extracted through ROI pooling and merged to estimate final boxes. AVOD [10] extends the idea by projecting all pre-built prior boxes on high-resolution feature maps from LiDAR BEV and RGB images. MVX-Net [11] adds pooled features from RGB feature maps to voxels used in VoxelNet. The method described in [27] takes advantage of leveraging continuous convolutions to merge feature maps from each sensor. The authors of [30] improve this architecture by adding related tasks such as ground estimation and depth completion to boost the 3D refinement subnetwork.

Sequential streams. The methods introduced in this paragraph operate sequentially and are 2D-driven. These processes exploit results from RGB processing as filters on the point cloud, contrary to the previously mentioned techniques. Frustum PointNet [13], Frustum Convnets [29] and PointFusion [26] exploit results from a 2D detector to extract points that fall inside each detection and then reduce the search space for each object. The frustums are then processed to precisely localize objects. IPOD [28] removes the background 3D points by exploiting a segmentation map from RGB camera images. Each foreground point is used as a location for prior boxes and then processed afterwards. In [12], the authors show the improvement of performances on several LiDAR-based detectors only by integrating the semantic information from a segmentation map in the input.

B. Point Cloud-based Resolution-Agnostic Deep Learning

To the best of our knowledge, domain adaptation has become an active research field on images. However, less research has been conducted on domain adaptation or portability on 3D point clouds, especially in an outdoor sensor data. As shown in [39], each 3D LiDAR sensor has its own set of characteristics corresponding to a range, point distribution, data coherency on disturbed conditions,etc. The authors of [40] perform an in-depth analysis of the performance of their architecture for semantic segmentation of point clouds from a 32-channel and a 128-channel LiDAR. 3D Domain Adaptation is studied and analyzed on 3D CAD (Computer-Aided Design) object datasets such as ModelNet [41]. Studies on outdoor scenes are less usual and most of them target the semantic segmentation of point clouds. In [42], a shared representation is learned in a self-supervised fashion through the reconstruction of deformed 3D point clouds. With PointDAN [43], an architecture is designed to learn cross domain local and global features to align objects from two distinct distributions. Some methods such as [44] have recourse to Generative Adversarial Networks to close the gap between synthetic data and real-world data. In [45], the authors propose a point cloud completion method by assimilating the task of reconstructing the underlying surface. A semantic segmentation network is then trained from the reconstructed surface that serves as the new canonical domain. Our method does not directly aim at the LiDAR domain adaptation. Instead, we aim at enforcing some aspects of the network in the training phase in order to make it more resilient to variations in point cloud resolution.

III. Object Detection Network Independent of Point Cloud Resolution

In this paper, we propose an end-to-end trainable method for 3D object detection. The network takes point clouds and images as inputs.

A. LiDAR Data Preprocessing

This section lists the set of operations applied on the 3D LiDAR point cloud.

A 3D point cloud is defined as a set $\mathbf{P} = \{[x_i, y_i, z_i]^T \in \mathbb{R}^3\}_{i \in 1...N}$ with $[x_i, y_i, z_i]^T$ the location of the i-th point in the 3D space. The common frame is defined with x-axis oriented forward, the y-axis leftward and the z-axis upward.

The input point cloud is discretized according to the method described in [15]: The point cloud is turned into column voxels with no vertical discretization compared to a 3D voxel grid that slices the point cloud over the three dimensions. This way, a BEV pseudo-image can be generated where each non-zero pixel is assigned to a non-empty voxel. The terms “pixel” and “voxel” are used interchangeably when referred to the BEV image: One column voxel (volume) is linked to only one cell of the BEV map.

Each voxel has two data representations: A version containing the 3D points, name *3D voxel*, and a version containing only the projections of these points on the image plane, named *RGB voxel*.

The search space is restricted to the interval $[x_{min}, x_{max}]$ (in meters) on the x axis and $[y_{min}, y_{max}]$ (in meters) on the y axis. This point cloud is discretized into a maximum of $n_x^{in} \times n_y^{in}$ voxels over the two dimensions.

B. Network Architecture

Figure 2 describes the global structure of the system. The RGB image is turned into a feature map which is sampled using the projections of the 3D points on the image plane. The 3D points and the extracted features are then concatenated. The resulting voxels are then processed by a PointPillars architecture [15]. The approach looks like the one described in [12] except the extracted features are not gathered from a semantic segmentation output. Moreover, in opposition with MVXNet whose image network is a pre-trained Faster-RCNN, we train our network from the beginning.

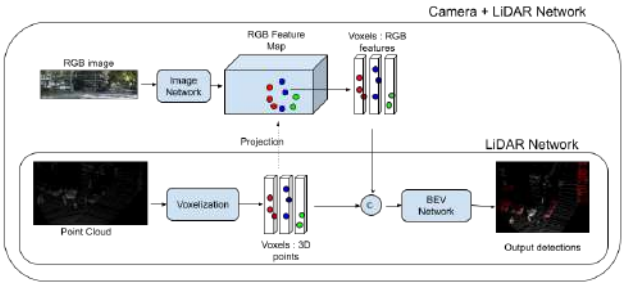


Fig. 2: Overview of the proposed network architecture.

a) Image Network: This network, processing input RGB images to produce feature maps to sample, has its architecture detailed in Figure 3. In this paragraph,

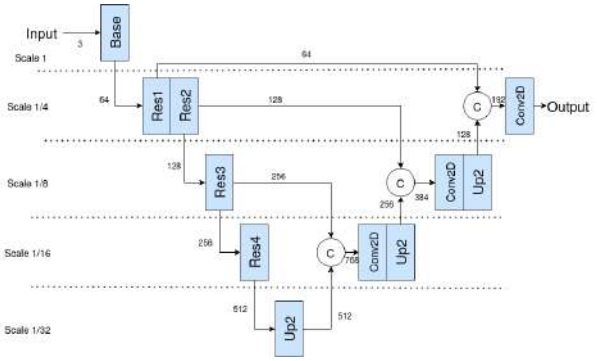


Fig. 3: Architecture of the image network.

Conv2D(c_{in}, c_{out}, k, s, p) refers to a 2D convolution operator where c_{in}, c_{out} refers to the number of input and output channels, k the kernel size, s the stride, p the padding. Likewise, MaxPool(k, s, p) denotes a Max Pooling operator. Blocks named "Up2" represent an up-sampling with a factor 2, respectively. Linear(c_{in}, c_{out}) represents a linear layer where c_{in}, c_{out} gives the number of input and output channels. The block named "Base" used in the Image Feature Extractor's Core is composed of a Conv2D(3, 64, 7, 2, 3) (for the 3 RGB channels), a batch normalization, a ReLU activation and a MaxPooling(3,2,1). Block labeled Res_i comes from a ResNet18 architecture [46]. As input images size may vary with the dataset, we chose a ResNet18 as a basis because of it small size and its small computation charge.

b) Sampling: The image use is heavily inspired by the *PointFusion* method described in [11]. The feature map computed by is then sampled through the projections contained in each *RGB voxel*.

c) BEV Network: As indicated, the network processing the voxels is a PointPillars network [15]. Each voxel, containing 3D points and their corresponding RGB features, are first processed by a *Voxel Feature Encoder* (VFE) in order to represent its content as a vector. All these vectors are scattered across a BEV pseudo-image which is then fed to a *Region Proposal Network* (RPN). The network returns 3 outputs: a classification map, a direction map and a regression map.

C. Network Output

In common datasets, 3D obstacles are represented by a position $[x,y,z]^T$, dimensions $[h,w,l]^T$ and an orientation θ. The orientation is often restrained to the vertical axis. The set of parameters is $S = \{x,y,z,h,w,l,\theta\}$.

We decide to represent each obstacle as a normal distribution: $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$:

$$\boldsymbol{\mu} = [x,y]^T \in \mathbb{R}^2,\ \boldsymbol{\Sigma} = \begin{pmatrix} a & b \\ b & c \end{pmatrix} \in \mathbb{R}^{2\times 2} \tag{1}$$

with

$$a = \frac{\cos^2(\theta)}{2\sigma_w^2} + \frac{\sin^2(\theta)}{2\sigma_l^2},\ b = -\frac{\sin(2\theta)}{4\sigma_w^2} + \frac{\sin(2\theta)}{4\sigma_l^2},$$
$$c = \frac{\sin^2(\theta)}{2\sigma_w^2} + \frac{\cos^2(\theta)}{2\sigma_l^2},\ \sigma_w = \frac{w}{3},\ \sigma_l = \frac{l}{3},$$

$\boldsymbol{\Sigma}$ being symmetric positive-definite.

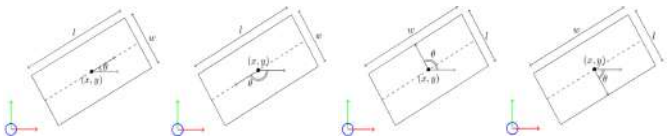
We note

$$F: \mathbb{R}^7 \to \mathbb{R}^2 \times \mathbb{R}^{2\times 2}$$
$$(x,y,z,h,w,l,\theta) \mapsto \boldsymbol{\mu}, \boldsymbol{\Sigma}$$

the function that turns the parameters S into a multivariate Gaussian function.

With this expression, an obstacle is defined as a Gaussian function. This representation focuses on the shape of the Gaussian function and the space occupied by its related obstacle. However, it does not provide information about the target's orientation. Therefore, ambiguities arise from the fact that multiple configurations can produce the same Gaussian function as illustrated in Figure 4.

The network returns three same size maps: A map for classification, a map for regression of boxes' parameters S, and one for orientation disambiguation. An anchor-free approach inspired from [37] is used here to represent the object where each predicted object is described by its center. This method is contrary to many approaches defining prior boxes with various sizes and orientations located in fixed positions. Indeed, an anchor-based method requires a set of hyperparameters specifically tuned to work correctly (size of anchors, Intersection of Union (IoU) threshold to define a positive anchor, etc.). Anchor-free method removes this constraint, allowing to reduce the number of critical parameters that may significantly affect the training process.



(a) $w = w_{ref}$, $l = l_{ref}, \theta = \theta_{ref}$ (b) $w = w_{ref}$, $l = l_{ref}, \theta = \theta_{ref} - \pi$ (c) $w = l_{ref}$, $l = w_{ref}, \theta = \theta_{ref} + \frac{\pi}{2}$ (d) $w = l_{ref}$, $l = w_{ref}, \theta = \theta_{ref} - \frac{\pi}{2}$

Fig. 4: Parameter sets producing the same Gaussian function, $w_{ref}, l_{ref}, \theta_{ref}$ are the reference values.

IV. Implementation Details

A. Training

1) Ground Truth Formatting: Each ground truth object i is defined by a class and its box parameters $[x_i, y_i, z_i, h_i, w_i, l_i, \theta_i]^T$ with its position $[x_i, y_i, z_i]^T$, its dimensions $[h_i, w_i, l_i]^T$ and its orientation θ_i.

Ground truth maps are arranged using a process described in [37]. Each label is represented by a Gaussian function, whose mean is the center of the object. A voxel is named positive if its ground truth value is 1 (the center of the object belongs to this voxel). One label is then assigned to one voxel in the classification map.

The regression map is composed of 7 channels, one for each box parameter. Let us denote Δx the first channel, Δy the second one and so on, see (2). Voxels are defined according to a static grid with a fixed position and each voxel on the classification map represents a portion on the metric space. For the ground truth object i, its associated voxel, whose center is located at $(\tilde{x}_i, \tilde{y}_i)$ of the metric space and at (u, v) on the regression map, the corresponding target for the regression is defined as:

$$\begin{aligned} \Delta x = \tilde{x}_i - x_i,\ \Delta y = \tilde{y}_i - y_i,\ \Delta z = z_i,\ \Delta h = h_i, \\ \Delta w = w_i,\ \Delta l = l_i,\ \Delta \theta = \theta_i \end{aligned} \tag{2}$$

The orientation map is a four-channel image, each channel representing one of the four quadrants defined in Figure 5. This map is defined to select the correct case from the ones defined in Figure 4.

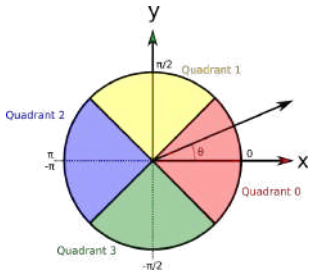


Fig. 5: The four quadrants of the orientation map.

2) Loss Functions: A multitask-loss is used:

$$L = \gamma_{cls} L^{cls} + \gamma_{reg} L^{reg} + \gamma_{ori} L^{ori} \tag{3}$$

with L^{cls}, L^{reg}, L^{ori} the losses related to scene classification, parameter regression, orientation ambiguities removal. γ_{cls}, γ_{reg}, γ_{ori} are their respective weights.

a) Scene Classification: L^{cls} is a mean of *focal losses* applied on each pixel of the classification map.

b) Regression: The following computations occur only on the locations of positive voxels. Then, we suppose the box parameters extracted from the corresponding prediction and target maps. The box parameters are split into two sets, $S_c = \{x, y, z, h\}$ and $S_{amb} = \{w, l, \theta\}$, S_{amb} is related to rotation ambiguities.

The obstacles can be defined as probability laws, therefore the definition of a statistical distance between a target obstacle and a predicted obstacle is possible. In our case, we decided to use the Bhattacharyya distance. If P, Q are two multivariate Gaussian functions $P = \mathcal{N}(\boldsymbol{\mu}_P, \boldsymbol{\Sigma}_P), Q = \mathcal{N}(\boldsymbol{\mu}_Q, \boldsymbol{\Sigma}_Q)$, the Bhattacharyya distance is expressed as:

$$D_B(P,Q) = \frac{1}{8}(\boldsymbol{\mu}_P - \boldsymbol{\mu}_Q)^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{\mu}_P - \boldsymbol{\mu}_Q) + \frac{1}{2} \ln \frac{\det \boldsymbol{\Sigma}}{\sqrt{\det \boldsymbol{\Sigma}_P \det \boldsymbol{\Sigma}_Q}}. \tag{4}$$

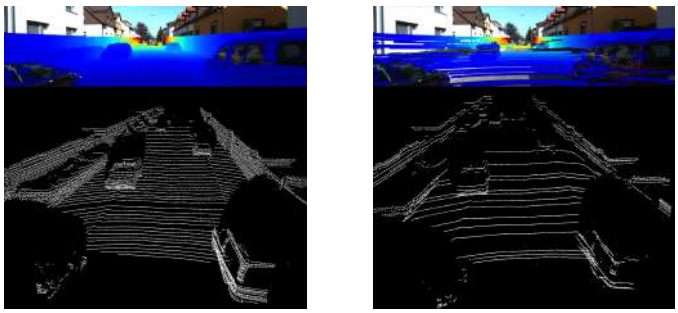
The regression loss is then defined as:

$$L^{reg} = \frac{1}{N_{pos}} \sum_{pos} SL1(S_c, \hat{S}_c) + D_B(F(S_c, \hat{S}_{amb}), F(S_c, S_{amb})) \tag{5}$$

with pos the locations of the positive voxels, N_{pos} the number of positive voxels, $SL1$ the *Smooth L1* loss, S_* and $\hat{S}_*$ respectively the targets and the predictions.

c) Orientation disambiguation: L_{ori} is a *focal loss* applied on the positive pixel locations of the estimated and the ground truth orientation maps.

3) Data Augmentation: In the field of autonomous driving, the most common LiDAR sensors used are rotating mechanical sensors, delivering 3D points from one emission location in a layered manner. Points from the same layer share the same latitude in a spherical frame centered on the sensor. For each sample, some layers are randomly discarded as illustrated in Figure 6. During the training, they are randomly discarded until between 25% and 60% in each point cloud.



(a) Initial point cloud (b) Point cloud after layer reduction

Fig. 6: Illustration of randomized layer removal (b). (a) Initial point cloud. On the top images, the point cloud projections are displayed. Colors represent distance from the LiDAR sensor.

V. Experimental protocol

We describe here the conducted experiments and how they are evaluated across the datasets.

A. Description of Datasets

In this study, the experiments are conducted on three datasets: The KITTI dataset, the nuScenes dataset and the

Pandaset dataset. Table II sums up some features of the studied datasets.

a) KITTI: The embedded sensors include a 64-channel LiDAR (Velodyne HDL-64E) on the top of the vehicle and two front color cameras. In this study, only the subset dedicated to 3D object detection is used. This dataset contains 7481 annotated training samples and 7518 testing samples. The 7481 train samples are commonly split into a *train* group of 3712 samples and a *validation* group of 3769 samples. Each sample consists of a panoramic 3D point cloud. The RGB images from each camera are synchronized. The calibration parameters are provided. The annotations are only defined in the camera's field of view.

b) Pandaset: Pandaset [47] contains scenes recorded on San Francisco and El Camino Real. The robotic vehicle is equipped with six cameras, one spinning 64-channel LiDAR and one front solid-state LiDAR. In this study, we focus on the spinning LiDAR Hesai Pandar64.

c) nuScenes Mini: nuScenes [48] is a dataset that provides data recorded through Boston and Singapore on 360° thanks to one 32-channel LiDAR and six cameras. One specificity of this dataset is that it encourages the accumulation of point clouds over multiple frames (called *sweeps*) to get a denser cloud. If the timestamps are taken into account, the estimation of the velocity of the targets is possible. For our experiments, we aim at detecting on synchronized point cloud and RGB image. Thus, the intermediate sweeps are not used. For the current experiments, we focus on the "Mini" subset containing 10 sequences.

B. Comparison Methodology

Each annotated dataset contains biases. Thus, for the same tasks, the required criteria may greatly vary among the datasets. In the following paragraphs, we detail our choices in order to compare the results on different datasets with the same setup. KITTI training split is used for training our experiments. Hence, we modify point clouds of testing point clouds sets to fit into the training conditions of the network. The x-axis is oriented forward, the y-axis leftward and the z-axis upward. All datasets have at least a 360° mechanical LiDAR and at least one front camera. However, due to restrictions in the KITTI dataset, we only exploit the section of the point cloud inside the frustum of the front camera. Hence, this restriction is also applied on validation data. Moreover, the experiments are trained for a specific range. Consequently, all validation point cloud are cropped to fit into this range. Each dataset defines its class names and, for each similar category, the targets may greatly differ. For example, the "Car" class may include large personal vehicles such as SUVs and Vans in one dataset. In another dataset, the same "Car" class may only include small vehicles and two other classes for the SUVs and the Vans might be specifically created. For these experiments, we tried to establish correspondences of class names that represent the same entities in the different datasets. Each dataset defines its own obstacle representation for obstacles. For example on nuScenes, the orientation of annotations are defined as quaternions, allowing the rotation on the three axes. On KITTI however, the road is supposed horizontal, thus the orientation is defined as a scalar representing the rotation angle around the vertical axis. Some datasets also define attributes to describe the temporary states of the targets. For example, the dynamic status of a vehicle can be described as "Moving", "Stopped" or "Parked". For this study, each obstacle is defined according to the KITTI standards: Semantic class, position, dimensions and rotation around the vertical axis. Moreover, for nuScenes and Pandaset, we check if an annotation includes at least 5 points. If not, the annotation is discarded. All datasets use the Average Precision (AP) as a performance metric. However, depending on the method, criteria to separate true detections from errors may differ. For example, the nuScenes compute the AP using distances between the center of the targets and the proposals. In this study, we keep the process used in KITTI: An IoU-based AP computation. For the KITTI validation split, we consider the difficulty-based clustering introduced by the dataset. Moreover, the IoU threshold is fixed at 0.7. However, for Pandaset and nuScenes, all labels are considered as belonging to the same group. Easy and hard cases are grouped under the same value. In fact, KITTI difficulties are defined according to parameters on the image plane that are not explicitly defined on the other datasets. Moreover, for Pandaset and nuScenes, we also display the AP with an IoU threshold at 0.5. The reason is that each dataset has its own biases, especially on the box fitting. A high threshold would have supposed that the annotators on different bases have the same sensibility about data extrapolation and box fitting. We focus our study on BEV detection. In fact, this work also concerns the impact of low-resolution point clouds for the evaluation. Ground and car tops may not be defined enough to estimate features related to the vertical axis (height and altitude). However, the main goal remains the object detection on data recorded on unknown environments with different LiDAR sensors. Depending on the equipped sensors and the choices of the annotators, the maximum labeling range varies greatly among datasets. nuScenes limits its annotations to 50 m for cars while Pandaset can provide labels beyond 100 m. However, our networks are trained to work until 80 m. Thus, for a shorter range (nuScenes), the point cloud is used as if it reaches 80 m but the outputs beyond the labeling range are discarded. For a longer range, the input point cloud is cropped to fit into the 80 m along the x axis.

C. Conducted Experiments

In order to evaluate the contributions of the presented techniques, we conduct different experiments, each one with a different configuration.

The input range is set to $[x_{min}, x_{max}] \times [y_{min}, y_{max}] = [0.0, 70.4] \times [35.2, 35.2]$. The base surface of each voxel (on the xy plane) measures 0.22×0.22 m^2, leading to a voxel grid with 320×320 cells. $(\gamma_{cls}, \gamma_{reg}, \gamma_{ori})$ are defined as $(2.0, 1.0, 0.2)$. For the focal loss, (α, β) is defined as $(2, 4)$. These values come from an improved implementation of [8]. All models, *CL* and *L*, are trained for 100 epochs using an Adam optimizer and a one-cycle learning rate scheduler with a maximum learning rate set to 0.001. About the common data augmentation techniques applied, the point cloud is first randomly mirrored along the y axis. We then perform random global translations,

TABLE II: Datasets used for experimental evaluation.

Name	Number of scenes	Weather conditions	Image size	LiDAR
KITTI	7481 (Train)	Sunny	Variable (about 1242×375)	Velodyne HDL-64E
nuScenes Mini	404 (10 sequences)	Sunny, Night, Cloudy	1600×900	Reference not provided (32 layers)
Pandaset	8240 (103 sequences)	Sunny, Rainy, Night	1920×1080	Hesai Pandar64

rotations and scalings on the point clouds and the annotations. On camera-LiDAR methods, all images in the dataset do not have the same sizes. Hence, to allow batch processing, each image is randomly cropped, so the result has 1024×256 pixels. Moreover, noise is added to images and parameters such as saturation and value are altered.

All the experiments are trained on the *train* split of the KITTI dataset. Each experiment is identified with a name whose structure is defined as *Sensor-TrainNumLayers-Loss*. The part *Sensor* can take the values *L* for LiDAR or *CL* for Camera + LiDAR. *TrainNumLayers* qualifies the number of layers in point cloud used for the training phase, 64/8 when the number of layers is fixed, *Var* when the randomized layer reduction is applied. *Loss* refers to the loss used, *Std* for the loss described in [15], *G* for the loss function described in IV-A2. For example, *L-Var-G* refers to the experiment using 3D point clouds, layer reduction to generate randomized distribution and the statistical loss. Note that all experiments, including *CL*-experiments are trained from scratch. The RGB parts do not come from pre-trained networks. Consequently, both RGB and BEV parts are trained as a single network.

In order to get a reference, we also evaluate the performances of a pre-trained PointPillars network with its original parameters (anchor-based, 0.16×0.16 m^2 voxels, ...) and a pre-trained PV-RCNN [9].

VI. Experiments on KITTI Dataset

In this section, we introduce the experiments conducted on the KITTI dataset [49]. Results of all experiments on the *validation* split are displayed in Table III. We detail the different observations in the following paragraphs.

The *CL*-experiments run at 35 ms and the *L*-experiments run at 20 ms on a computer equipped with a GPU NVidia 1080Ti and a CPU Intel Core i7−7700K.

A. Differences between Camera-LiDAR Methods and LiDAR Networks

On 64 layer point clouds and for the same number of epochs, better performances are observed on networks using only point clouds in comparison with networks exploiting both sensors.

First, all networks are trained with the same hyperparameters. Hence, under the same conditions, *CL*-networks, which contain more parameters due to the image subnetwork, are more difficult to train. Moreover, in opposition with 3D point clouds which are produced by an active sensor and contain only geometric information, images tend to be denser and more sensitive to photometric phenomena and camera artifacts.

When trained on low resolution point clouds, experiments (#5 *CL-8-Std*, #6 *CL-8-G*) perform better than their *L*-counterparts (#11 *L-8-Std*, #12 *L-8-G*) on high resolution data. The same observation is visible when the *CL*-networks are trained on 64-layer point clouds and applied on 8-layer data. This means that the network does not exclusively focus on the layout of the 3D points but also exploit the features extracted from the RGB image. Experiments (#11 *L-8-Std*, #12 *L-8-G*) only deal with extremely sparse point clouds that depict surfaces with low resolution. Hence, networks focus on optimizing this difficult train set and then are more prone to overfitting.

B. Influence of Layer Reduction

We first analyze the effects of the layer reduction on *CL*-experiments with standard loss (experiments #1 *CL-64-Std*, #3 *CL-Var-Std* and #5 *CL-8-Std*). On 64-layer point clouds, the layer reduction seems to have few impact on the experiments. In fact, both image features and detailed point cloud are useful for target identification. However, on 8-layer point clouds, experiment #1 *CL-64-Std* returns lower scores as no examples of such type were seen during the training stage. Experiment #5 *CL-8-Std* is highly specialized on low-resolution point clouds. The randomized layer removal helped experiment #3 *CL-Var-Std* to get the best trade-off.

Concerning the *L*-experiments, we focus on experiments #7 *L-64-Std*, #9 *L-Var-Std* and #11 *L-8-Std*. On 64-layer point clouds, we notice an improvement between experiments #7 *L-64-Std* and #9 *L-Var-Std*. The layer removal helped the network to learn a better representation of the targets as more diverse examples were associated to the same goal. Once again, this operation also helped to improve the outputs on low-resolution point clouds without specializing the network on the latter point cloud distribution.

These experiments show that random layer removal can be an easy way to improve the performances of a detector on low-resolution point cloud with minor loss (LiDAR methods) or even improvement (fusion methods) on high-resolution data.

C. Influence of Gaussian Representation

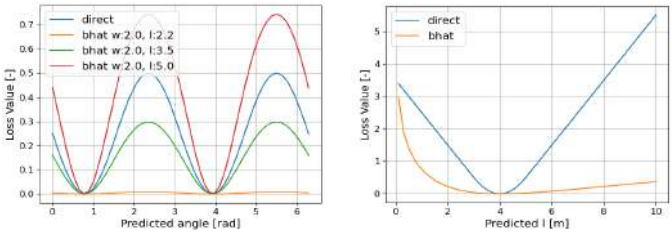
We focus now on the effects of the use of a statistical distance as a loss. On almost all cases, we observe a slight degradation of the results comparing to the experiments using Smooth-L1 losses.

Figure 7 shows the profiles of the Bhattacharyya loss D_B (Eq. 4) for variations according w, l and θ. In Figure 7a, all the parameters are fixed except θ. The curve looks like the sine function with global minima at $k\pi, k \in \mathbb{Z}$, allowing multiple solutions. The main difference with the sine function

TABLE III: Evaluation of all experiments for BEV detection on the KITTI validation set, nuScenes Mini and Pandaset datasets, 64 layers and evenly spaced 8 layers. All the results are computed with Average Precision (%). The last number in dataset identifier indicates the IoU threshold used for computing the scores.

Exp. ID	KITTI 64 layers 0.7			KITTI 8 layers 0.7			nuScenes 0.5	Pandaset 0.5	Mean Datasets
	Easy	Moderate	Hard	Easy	Moderate	Hard			
#1 *CL-64-Std*	75.02	59.22	54.71	41.81	29.48	27.29	9.09	11.26	29.04
#2 *CL-64-G*	72.07	53.10	53.31	42.11	29.59	27.47	9.09	15.31	29.23
#3 *CL-Var-Std*	77.46	58.49	54.50	61.57	40.46	38.83	15.65	14.39	35.11
#4 *CL-Var-G*	70.18	51.90	52.55	57.14	38.61	36.34	16.61	7.41	31.56
#5 *CL-8-Std*	71.55	58.72	54.57	65.30	46.96	41.10	16.26	4.73	33.43
#6 *CL-8-G*	65.55	48.19	47.92	64.34	45.12	40.06	16.35	9.09	32.29
#7 *L-64-Std*	85.35	75.16	71.13	27.55	20.95	18.21	37.69	49.35	46.62
#8 *L-64-G*	84.93	75.02	71.59	27.21	18.99	16.62	50.12	**52.63**	50.21
#9 *L-Var-Std*	86.46	75.69	74.43	58.61	40.34	35.81	47.86	49.00	55.16
#10 *L-Var-G*	85.90	75.45	72.30	58.16	40.56	35.48	**66.91**	52.26	**60.44**
#11 *L-8-Std*	48.54	44.18	44.53	**67.65**	**48.26**	**43.62**	55.04	16.82	42.69
#12 *L-8-G*	43.85	42.18	41.95	66.32	47.18	42.83	44.43	11.58	37.69
PointPillars [15]	89.65	87.17	84.37	46.99	32.34	27.85	29.38	30.47	45.66
PV-RCNN [9]	**90.26**	**88.04**	**87.39**	40.02	28.66	26.48	32.96	39.73	48.24

concerns the impact of the target's dimensions. We tested D_B for different target lengths. The closer to 1 the ratio $\frac{l}{w}$ is, the less the occupancy of the target changes if rotated, and less the error on θ affects the loss. Thus, this loss function acts as a weight on targets as it focuses on an angular error from more elongated targets. The loss function has one global minima and the function is convex. We note that the loss value is greater if the prediction is smaller than the target. However, the values returned by the loss tend to be much lower than the ones returned by a Smooth-L1 loss, leading to a more difficult convergence on the dimensions as shown in Figure 7b.



(a) $D_B(\theta)$ for different object sizes with $\theta_{gt} = \frac{\pi}{4}$

(b) $D_B(l)$ with $l_{gt} = 4\,\text{m}$

Fig. 7: Variations of D_B as a function of a single parameter, the others being fixed at ground truth.

D. Conclusion

The effects of our training procedures are studied over data from the KITTI validation split. The layer reduction allows improving the performances of a network in various cases easily without any priors. However, while the point cloud distribution varied, the scene remains the same, which does not indicate if the network can be applied on other places and with other sensors. On the KITTI dataset, the Gaussian representation seems to degrade the network as it is less precise than the Smooth-L1. However, its benefits are more visible on out-of-distribution data.

VII. Experiments with other Datasets

Until now, we have only generated situations where the point clouds vary but the environments and the annotation biases stay the same. In this section, we experiment with our trained networks on nuScenes Mini and Pandaset. By testing on other datasets, we can evaluate how our complete network reacts to stimuli of different environments and capture setups. Table III provides the values computed on the nuScenes and Pandaset datasets.

A. Common Analyzes

The main observation is that methods exploiting RGB images perform poorly in all cases. Networks using only point clouds tend to resist better to these changes. In fact, only bare geometrical information is available on 3D point clouds. Targets are often recognizable by their shape. Additional data such as RGB information that could be affected by external phenomena are integrated by the network during the training stage. When training on KITTI without constraints on images features, the networks implicitly learn the specificities of the camera (noise artifacts, dynamic range, deformations…) but also what characterize the scenes. This could be lighting (e.g. twilight and night scenes missing in KITTI), the different urban infrastructures or even the car models available in restricted geographic regions. Hence, changing the image domain has an important impact on methods using both sensors.

B. Analyzes on nuScenes Mini

As expected, the performances drop significantly on the new dataset. The labels are originally designed for point cloud accumulation. Thus, detection on a single scan is more difficult. Furthermore, the results on experiments #8 *L-64-G* and #10 *L-Var-G* are now better on this dataset than they were before on the KITTI dataset and surpass their counterparts (experiments #7 *L-64-Std* and #9 *L-Var-Std*, respectively). For out-of-distribution data, the new loss function helped the network to acquire a more flexible (but less precise) representation of a car, allowing a better knowledge transfer on unknown data.

Figure 8 illustrates qualitative results for experiment #10 *L-Var-G* on the nuScenes dataset. We can observe the contrast between the scene captured by the camera and the sparsity of one point cloud.

C. Analyzes on Pandaset

We observe the same general dynamics on the unaltered point clouds as during the comparison on the KITTI validation set. Despite the similar number of points available on both point clouds, AP values are much lower on Pandaset than on KITTI dataset. First, even if the KITTI train and validation splits are distinct, all samples come from the same sequences, inducing an important bias. Moreover, compared to KITTI, the density of labels is higher on Pandaset data. Among the labels, many of them are located outside the roadway, in a mall parking lot for example. Another reason related to possibly missed obstacles concerns the intra class variability. Depending on the city, the distribution of car models may vary considerably. On the recorded scenes, we observe vehicle models that do not exist in the KITTI dataset (recorded in Germany). Thus, the network tends to return the same parameters as it only observe a small part of the spectrum of vehicle models during the training phase.

Figure 9 presents qualitative results obtained for experiment #10 *L-Var-G* on Pandaset data. We observe that the network provides relevant BEV detections despite the context difference. Even if the number of targets is large in the scene, the network is still able to extract most of the obstacles.

VIII. Conclusion

In this paper, we study how 3D object detection networks react over datasets and point cloud distributions. We propose an approach to improve performances over unknown data without supplementary knowledge. The approach rests on two points. The first one consists in randomly discarding point cloud layers during the training. This technique allows to enforce the network to not overfit a 3D point distribution. The second one is a new obstacle representation. Each obstacle is represented by a Gaussian function, allowing the use of statistical distances as loss functions. Despite not being a domain adaptation method, our approach, exclusively trained on a subset of the KITTI dataset, provides accurate outputs on data from nuScenes and Pandaset datasets that were never revealed at the training stage. The work presented in this paper focuses on car detection. In future work, it would be needed to study how the method and the loss function act in multi-class detection. When working on low-resolution point clouds, it is hard to estimate the features of a target with complete exactitude. Finally, it would be interesting to investigate uncertainty estimation for object detection, especially for a critical field such as autonomous driving.

Acknowledgments

This work has been sponsored by Sherpa Engineering and ANRT. This work has been sponsored also by the French government research Program Investissements d'Avenir through the RobotEx Equipment of Excellence (ANR−10−EQPX−44) and the IMobS3 Laboratory of Excellence (ANR−10−LABX−16−01), by the European Union through the program Regional competitiveness and employment 2014−2020 (FEDER − AURA region) and by the AURA region.

References

[1] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," in *IEEE Conf. on Computer Vision and Pattern Recognition*, 2017, pp. 652–660.

[2] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space," in *Advances in Neural Information Processing Systems*, 2017, pp. 5099–5108.

[3] S. Shi, X. Wang, and H. Li, "PointRCNN: 3D Object Proposal Generation and Detection From Point Cloud," in *IEEE Conf. on Computer Vision and Pattern Recognition*, June 2019.

[4] Y. Chen, S. Liu, X. Shen, and J. Jia, "Fast Point R-CNN," in *IEEE Intl. Conf. on Computer Vision (ICCV)*, October 2019.

[5] "Waymo driver." [Online]. Available: https://waymo.com/waymo-driver/

[6] "Motional." [Online]. Available: https://motional.com/

[7] Y. Zhou and O. Tuzel, "Voxelnet: End-to-end learning for point cloud based 3d object detection," in *IEEE Conf. on Computer Vision and Pattern Recognition*, 2018, pp. 4490–4499.

[8] Y. Yan, Y. Mao, and B. Li, "SECOND: Sparsely Embedded Convolutional Detection," *Sensors*, vol. 18, no. 10, 2018.

[9] S. Shi, C. Guo, L. Jiang, Z. Wang, J. Shi, X. Wang, and H. Li, "PV-RCNN: Point-Voxel Feature Set Abstraction for 3D Object Detection," in *IEEE/CVF Conf. on Computer Vision and Pattern Recognition*, June 2020, pp. 10 529–10 538.

[10] J. Ku, M. Mozifian, J. Lee, A. Harakeh, and S. L. Waslander, "Joint 3D Proposal Generation and Object Detection from View Aggregation," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2018, pp. 1–8.

[11] V. A. Sindagi, Y. Zhou, and O. Tuzel, "MVX-Net: Multimodal voxelnet for 3D object detection," in *Intl. Conf. on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 7276–7282.

[12] S. Vora, A. H. Lang, B. Helou, and O. Beijbom, "PointPainting: Sequential Fusion for 3D Object Detection," in *IEEE/CVF Conf. on Computer Vision and Pattern Recognition*, June 2020.

[13] C. R. Qi, W. Liu, C. Wu, H. Su, and L. J. Guibas, "Frustum PointNets for 3D Object Detection from RGB-D Data," in *IEEE Conf. on Computer Vision and Pattern Recognition*, 2018, pp. 918–927.

[14] M. Wang and W. Deng, "Deep visual domain adaptation: A survey," *Neurocomputing*, vol. 312, pp. 135–153, 2018.

[15] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "PointPillars: Fast Encoders for Object Detection From Point Clouds," in *IEEE Conf. on Computer Vision and Pattern Recognition*, June 2019.

[16] T. Roddick, A. Kendall, and R. Cipolla, "Orthographic Feature Transform for Monocular 3D Object Detection," *arXiv preprint arXiv:1811.08188*, 2018.

[17] G. Brazil and X. Liu, "M3D-RPN: Monocular 3D Region Proposal Network for Object Detection," in *IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, Seoul, South Korea, October 2019, pp. 9287–9296.

[18] A. Simonelli, S. R. Bulo, L. Porzi, M. Lopez-Antequera, and P. Kontschieder, "Disentangling Monocular 3D Object Detection," in *IEEE/CVF Intl. Conf. on Computer Vision (ICCV)*, October 2019.

[19] S. Shrivastava and P. Chakravarty, "CubifAE-3D: Monocular Camera Space Cubification on Autonomous Vehicles for Auto-Encoder based 3D Object Detection," *arXiv preprint arXiv:2006.04080*, 2020.

[20] J. Beltran, C. Guindel, F. M. Moreno, D. Cruzado, F. Garcia, and A. De La Escalera, "BirdNet: a 3D Object Detection Framework from LiDAR information," in *Intl. Conf. on Intelligent Transportation Systems (ITSC)*. IEEE, 2018, pp. 3517–3523.

[21] M. Simon, S. Milz, K. Amende, and H.-M. Gross, "Complex-YOLO: An Euler-Region-Proposal for Real-Time 3D Object Detection on Point Clouds," in *Computer Vision – ECCV 2018 Workshops*, L. Leal-Taixé and S. Roth, Eds. Cham: Springer Intl. Publishing, 2019, pp. 197–209.

[22] Q. Chen, L. Sun, Z. Wang, K. Jia, and A. Yuille, "Object as Hotspots: An Anchor-Free 3D Object Detection Approach via Firing of Hotspots," *arXiv preprint arXiv:1912.12791*, 2019.

[23] T. Yin, X. Zhou, and P. Krähenbühl, "Center-based 3D Object Detection and Tracking," *arXiv:2006.11275*, 2020.

[24] S. Shi, Z. Wang, X. Wang, and H. Li, "Part-A^2 Net: 3D Part-Aware and Aggregation Neural Network for Object Detection from Point Cloud," *arXiv preprint arXiv:1907.03670*, 2019.

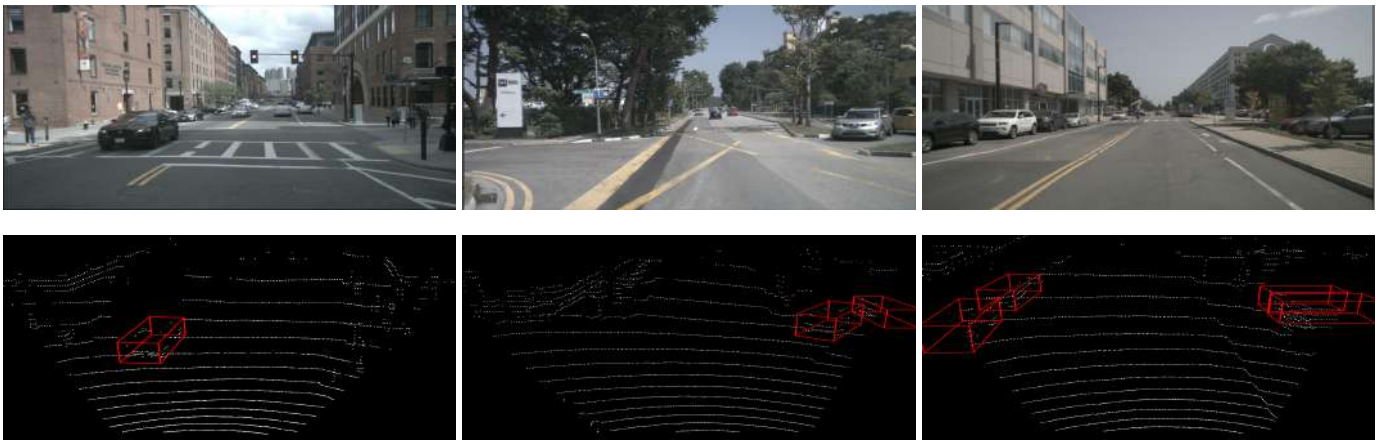


Fig. 8: Qualitative results for experiment #10 *L-Var-G* on nuScenes.

Fig. 9: Qualitative results for experiment #10 *L-Var-G* on Pandaset.

[25] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-View 3D Object Detection Network for Autonomous Driving," in *IEEE Conf. on Computer Vision and Pattern Recognition*, 2017, pp. 1907–1915.

[26] D. Xu, D. Anguelov, and A. Jain, "PointFusion: Deep Sensor Fusion for 3D Bounding Box Estimation," in *IEEE Conf. on Computer Vision and Pattern Recognition*, 2018, pp. 244–253.

[27] M. Liang, B. Yang, S. Wang, and R. Urtasun, "Deep Continuous Fusion for Multi-Sensor 3D Object Detection," in *European Conf. on Computer Vision (ECCV)*, 2018, pp. 641–656.

[28] Z. Yang, Y. Sun, S. Liu, X. Shen, and J. Jia, "IPOD: Intensive Point-based Object Detector for Point Cloud," *arXiv preprint arXiv:1812.05276*, 2018.

[29] Z. Wang and K. Jia, "Frustum ConvNet: Sliding Frustums to Aggregate Local Point-Wise Features for Amodal 3D Object Detection," in *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2019, pp. 1742–1749.

[30] M. Liang, B. Yang, Y. Chen, R. Hu, and R. Urtasun, "Multi-task multi-sensor fusion for 3d object detection," in *IEEE Conf. on Computer Vision and Pattern Recognition*, 2019, pp. 7345–7353.

[31] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, 2017.

[32] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in *IEEE Conf. on Computer Vision and Pattern Recognition*, 2016, pp. 779–788.

[33] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[34] B. Graham and L. van der Maaten, "Submanifold Sparse Convolutional Networks," *arXiv preprint arXiv:1706.01307*, 2017.

[35] H. Kuang, B. Wang, J. An, M. Zhang, and Z. Zhang, "Voxel-FPN: Multi-Scale Voxel Feature Aggregation for 3D Object Detection from LiDAR Point Clouds," *Sensors*, vol. 20, no. 3, 2020.

[36] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature Pyramid Networks for Object Detection," in *IEEE Conf. on Computer Vision and Pattern Recognition*, 2017, pp. 2117–2125.

[37] X. Zhou, D. Wang, and P. Krähenbühl, "Objects as points," *arXiv preprint arXiv:1904.07850*, 2019.

[38] H. Law and J. Deng, "Cornernet: Detecting objects as paired keypoints," in *European Conf. on Computer Vision (ECCV)*, 2018, pp. 734–750.

[39] J. Lambert, A. Carballo, A. M. Cano, P. Narksri, D. Wong, E. Takeuchi, and K. Takeda, "Performance Analysis of 10 Models of 3D LiDARs for Automated Driving," *IEEE Access*, vol. 8, pp. 131 699–131 722, 2020.

[40] F. Piewak, P. Pinggera, and M. Zöllner, "Analyzing the Cross-Sensor Portability of Neural Network Architectures for LiDAR-based Semantic Labeling," in *Intelligent Transportation Systems Conf. (ITSC)*. IEEE, 2019, pp. 3419–3426.

[41] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3D ShapeNets: A Deep Representation for Volumetric Shapes," in *IEEE Conf. on Computer Vision and Pattern Recognition*, 2015, pp. 1912–1920.

[42] I. Achituve, H. Maron, and G. Chechik, "Self-Supervised Learning for Domain Adaptation on Point-Clouds," *arXiv preprint arXiv:2003.12641*, 2020.

[43] C. Qin, H. You, L. Wang, C.-C. J. Kuo, and Y. Fu, "PointDAN: A Multi-Scale 3D Domain Adaption Network for Point Cloud Representation," in *Advances in Neural Information Processing Systems*, 2019, pp. 7192–7203.

[44] B. Wu, X. Zhou, S. Zhao, X. Yue, and K. Keutzer, "Squeezesegv2: Improved model structure and unsupervised domain adaptation for road-object segmentation from a LiDAR point cloud," in *Intl. Conf. on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 4376–4382.
[45] L. Yi, B. Gong, and T. Funkhouser, "Complete & Label: A Domain Adaptation Approach to Semantic Segmentation of LiDAR Point Clouds," *arXiv preprint arXiv:2007.08488*, 2020.
[46] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *IEEE Conf. on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.
[47] "Pandaset - open-source dataset for self-driving cars." [Online]. Available: https://pandaset.org/
[48] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuScenes: A multimodal dataset for autonomous driving," *arXiv preprint arXiv:1903.11027*, 2019.
[49] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The KITTI Dataset," *The Intl. Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

Ruddy Théodose received his Engineer Degree from the INP-ENSEEIHT Engineering school in 2016. He integrated Sherpa Engineering where he is currently pursuing a Ph.D. degree in partnership with Institut Pascal in Clermont-Ferrand, France. His Ph.D. thesis targets camera-LiDAR fusion for urban object detection. His research interests involve autonomous driving, multi-sensor fusion, deep learning and vehicle perception.

Dieumet Denis received, in 2015, the Ph.D. degree in Control of Mobile Robots from Institut Pascal, Université Blaise Pascal in Clermont-Ferrand, France. During his Ph.D., he largely studied the thematic of vehicles dynamics stability. Since 2016, Dr DENIS joined Sherpa Engineering company as Project Manager and Research Engineer in Autonomous Vehicles field. He is currently responsible for the Sherpa ADAS R&D Division in Clermont-Ferrand.

Thierry Chateau is Full Professor at Clermont Auvergne University. He is a member of the ISPR research team (80 researchers) at Institut Pascal. His main research topics are in Visual Tracking, Pattern Recognition and machine learning, within the field of Computer Vision.

Vincent Frémont is a Full Professor at Ecole Centrale de Nantes within the ARMEN team at the LS2N Lab, UMR CNRS 6004. His research interests belong to perception systems for autonomous vehicles with an emphasis on computer vision, machine learning and multi-sensor fusion.

Paul Checchin is a Full Professor at Université Clermont Auvergne and a member of the ISPR Department carrying on research in Computer Vision and Robotics, within the Institut Pascal (UMR 6602 UCA/CNRS). His main research interests focus on robot autonomy, localization, mapping, scene understanding and perception applied to SLAM of mobile robots, moving object tracking in extensive outdoor environments based on radar, LiDAR and visual sensors.