Computers & Graphics 99 (2021) 153–181

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Technical Section

A comprehensive survey of LIDAR-based 3D object detection methods with deep learning for autonomous driving

Georgios Zamanakos[a,*], Lazaros Tsochatzidis[a], Angelos Amanatiadis[b], Ioannis Pratikakis[a]

[a] Department of Electrical and Computer Engineering, Democritus University of Thrace, Xanthi, 67100, Greece
[b] Department of Production and Management Engineering, Democritus University of Thrace, Xanthi, 67100, Greece

ARTICLE INFO

Article history:
Received 2 April 2021
Revised 30 June 2021
Accepted 2 July 2021
Available online 7 July 2021

Keywords:
3D Object detection
Deep learning
Autonomous driving

ABSTRACT

LiDAR-based 3D object detection for autonomous driving has recently drawn the attention of both academia and industry since it relies upon a sensor that incorporates appealing features like insensitivity to light and capacity to capture the 3D spatial structure of an object along with the continuous reduction of its purchase cost.

Furthermore, the emergence of Deep Learning as the means to boost performance in 3D data analysis stimulated the production of a multitude of solutions for LIDAR-based 3D object detection which followed different approaches in an effort to respond effectively to several challenges.

In view of this, this paper presents a comprehensive survey of LIDAR-based 3D object detection methods wherein an analysis of existing methods is addressed by taking into account a new categorisation that relies upon a common operational pipeline which describes the end-to-end functionality of each method. We next, discuss the existing benchmarking frameworks and present the performance achieved by each method in each of them. Finally, a discussion is presented that provides key insights aiming to capture the essence of current trends in LIDAR-based 3D object detection.

1. Introduction

The first Grand Challenge by DARPA in 2004, aiming to develop a high speed autonomous vehicle for desert roads, remained unfulfilled since no team managed to cope with the challenge. However, in the second DARPA challenge in 2005, four vehicles managed to accomplish the challenge [1]. Perhaps even more famous is the 2007 DARPA Urban Challenge in which autonomous vehicles had to navigate in an rural environment, where extra attention should be paid to dynamic obstacles with unpredictable behavior such as 'car', 'pedestrian' and 'cyclist' among others [2]. Autonomous cars can be therefore considered as intelligent agents that have to provide solutions to many sub-tasks, one of which is to perceive the environment. Such a perception system should provide accurate, robust and reliable information concerning the environment around the autonomous car, in real-time [3].

A perception system usually consists of many different complementary sensors, out of which cameras and Light Detection And Ranging (LiDAR) sensors are the two most common ones. A camera provides data, rich in context information through a well structured but of limited spatial dimensions and resolution form, of a 2D image. On the other hand, a LiDAR sensor provides data of low-context information through a precise and high spatial dimension and resolution form, of a point cloud in 3D space. Compared to cameras, LiDAR sensors are not restricted by lighting conditions and furthermore capture data in a privacy-free manner, which makes them ideal for a complex outdoor environment both from a technical and ethical aspect.

Object detection in a 2D image plane is a well studied topic and recent advances in Deep Learning have demonstrated remarkable success in real-time applications [4–6]. However in autonomous driving applications, other than detecting objects in a 2D image plane, certain objects classes such as 'car', 'pedestrian' and 'cyclist', among others, must also be detected in the 3D space.

The advances in Deep Learning architectures along with publicly accessible datasets for autonomous driving and affordable LiDAR sensors, have a positive influence for the 3D object detection task, resulting in many emerged LiDAR-based 3D object detectors. A LiDAR-based 3D object detector uses solely the LiDAR sensor data without relying on data from any other sensor.

However, LiDAR-based 3D object detection for autonomous driving applications is a challenging task mainly due to the demand for high detection performance and fast inference. The challenge of effectiveness is directly related to the quality of LiDAR sensor data. While LiDAR operates without visible light which fa-

* Corresponding author.
E-mail address: gzamanak@ee.duth.gr (G. Zamanakos).

https://doi.org/10.1016/j.cag.2021.07.003
0097-8493/

cilitates the environmental perception for night driving or low light conditions, in the case of adverse weather conditions, extra noise is introduced that dithers the quality of the data, which affects the detection performance. Furthermore, the potential partial occlusions or self-occlusions in a scene produce incomplete data which coupled with the inherent sparsity of LiDAR as a function of the distance from the sensor, results in reducing the quality of the data. The challenge of efficiency is directly related to the large quantities of LiDAR data produced for the acquisition of the scenes where the detection occurs. The potential to apply a stitching of several point clouds to reduce the sparsity of a scene for areas far from the sensor makes the challenge more difficult to overcome.

Extensive surveys have been published for 3D object detection in point clouds [7] and for autonomous driving applications [8]. Other related reviews focus on object detection and semantic segmentation for autonomous driving applications, with Deep Learning networks using multi-modal techniques [3]. Closest to our paper, a review has been conducted for Deep Learning methods using LiDAR point clouds for autonomous driving applications [9], for 3D object detection using LiDAR data [10] and for categorizing point-cloud based 3D object detectors for autonomous driving [11].

Compared to the aforementioned previous efforts, the contributions of our comprehensive survey is summarized as follows:

1. Presentation of a common operational pipeline that sets the base for a structured categorisation which facilitates comparison and emerges the similarities and dissimilarities among the presented methods.
2. Presentation of an exhaustive up-to-date list of 3D object detectors. The method's description focuses on the particularities of each method with respect to the operational pipeline identifying pros and cons towards an effective and efficient detection outcome.
3. Fruitful discussion that aims to identify key features which should be either adopted or avoided in the design of new 3D object detectors.

The structure of the paper is as follows. In Section 2 the basic background information for the LiDAR sensor and object encoding in 3D space is outlined. Section 3 analyses the operational pipeline for LiDAR-based 3D object detectors and in Section 4, the detectors are reviewed with respect to the operational pipeline as well as the individual particularities of each methodology. Section 5 presents the three open datasets for autonomous driving along with their evaluation metrics for 3D object detection. In Section 6, key features are identified while the findings and key insights for each group of methods are presented and in Section 7 the future trends and open problems are discussed. Finally, conclusions are drawn in Section 8.

2. Background

2.1. LiDAR sensor and point cloud

The basic working principle of a LiDAR sensor is to measure the time of flight (ToF) of an emitted laser beam. By multiplying the speed of light with the measured ToF, an accurate distance is calculated. Different types of LiDAR sensors exist, out of which the mechanical LiDAR is the most common one, usually found also in autonomous driving applications. Mechanical LiDAR rotates horizontally a set of vertically configured laser emitters capturing a 360-degree field of view, with their number determining the points to be measured simultaneously on the vertical axis. The laser emitters are placed in different pitch angles and usually 64, 32 or 16 emitters are used.

The resulting output of a LiDAR sensor is a set of points, referred as a point cloud. Each point is usually encoded by a four dimensional vector containing its 3D coordinates $[x, y, z]$ and the laser reflection intensity r. The laser reflection intensity encodes information about the object's reflected surface. Certain LiDAR sensors [12] can encode each point with a five dimensional vector, providing an extra value e, corresponding to the elongation of the laser pulse beyond its nominal width [12]. The 3D coordinates of each point, in the LiDAR sensor's coordinate frame, are calculated as:

$$\begin{aligned} x &= d * cos(\omega) * cos(\phi) \\ y &= d * cos(\omega) * sin(\phi) \\ z &= d * sin(\omega) \end{aligned} \tag{1}$$

where d is the measured distance, ϕ is the yaw angle around Z axis and ω is the fixed pitch angle of each laser emitter.

2.2. 3D BBox Encoding

A three-dimensional bounding box (3D BBox) is used to define the location, size and orientation of an object in 3D space. The 3D BBox has proper dimensions and orientation to bound tightly to an object. If the object is partially occluded or truncated, the 3D BBox is of proper size and placed accordingly to represent the object's full size. For autonomous driving applications all objects are assumed to be on the ground, therefore an angle prediction is needed only for yaw. The 3D BBox is encoded by its 3D center coordinates $[x, y, z]$, its dimensions $[l, w, h]$ and its yaw orientation θ [13].

3. Operational pipeline of LiDAR based 3D object detectors

LiDAR-based 3D object detectors, from now on mentioned as 3D object detectors, use solely the LiDAR sensor data, without utilizing information from other sensors such as RGB cameras. To establish a common ground among all 3D object detectors, we present an operational pipeline, composed of three distinct modules, namely LiDAR Sensor Data Representation (SDR), Feature Extraction and Core Object Detection.

In the first operational module of LiDAR SDR, the incoming LiDAR sensor data is transformed into a structured and compact representation. Based on this representation, high-dimensional and rich features are extracted in the second operational module of Feature Extraction. The learned high-dimensional features are then processed in the third and final operational module of Core Object Detection. Core Object Detection is composed of two stages, Detector Networks and Prediction Refinement. The first stage of Detector Networks receives as an input the high-dimensional features from Feature Extraction module and outputs a 3D BBox prediction concerning the class, the location and the size of the object. The second stage of Prediction Refinement re-samples the predicted 3D BBox from the first stage of Detector Networks, extracts local features and process them to output a final more accurate prediction of the object's location, size and class.

3D object detectors that utilize solely the first stage of Detector Networks in their Core Object Detection operational module are categorized as one-stage, while if they use both stages of Detector Networks and Prediction Refinement, they are categorized as two-stage. One-stage 3D object detectors tend to be faster at inference, however two-stage ones tend to be more accurate in their predictions. The block diagram for 3D object detectors is presented in Fig. 1 and in the following sections each operational module is presented and analyzed.

3.1. LiDAR sensor data representations

The incoming LiDAR sensor data from one or multiple LiDAR sensors, depending on the setup and number of sensors installed

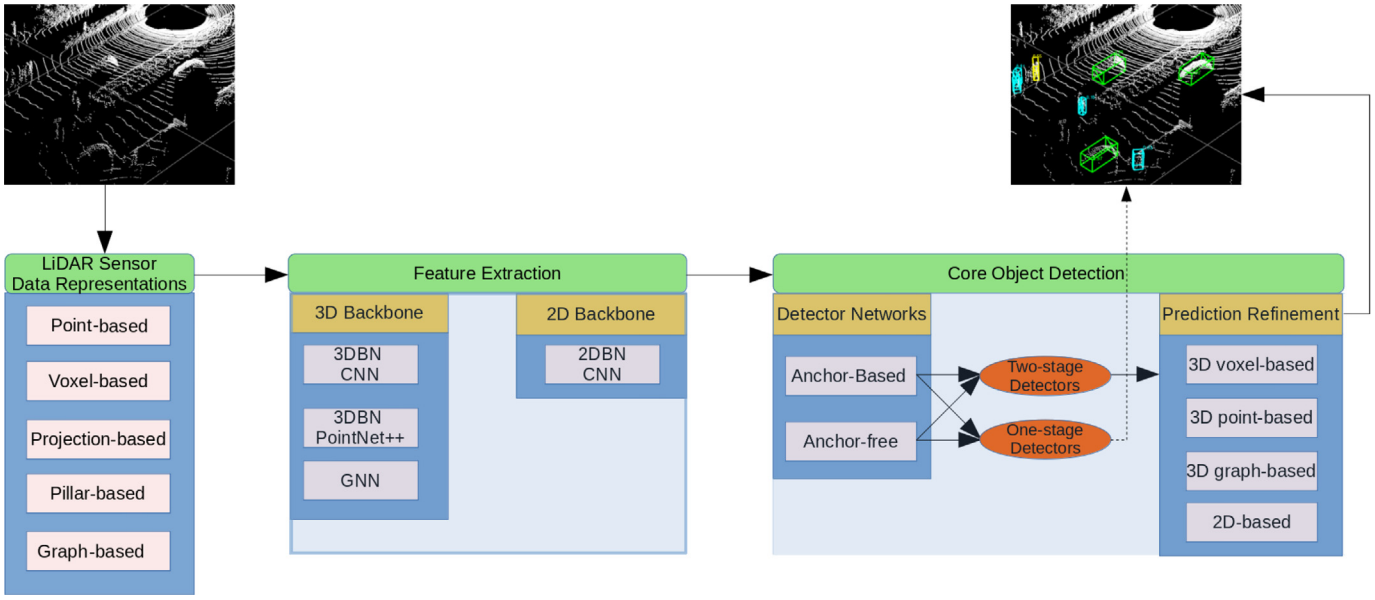


Fig. 1. Block diagram of the operational pipeline of LiDAR-based 3D object detectors with Deep Learning.

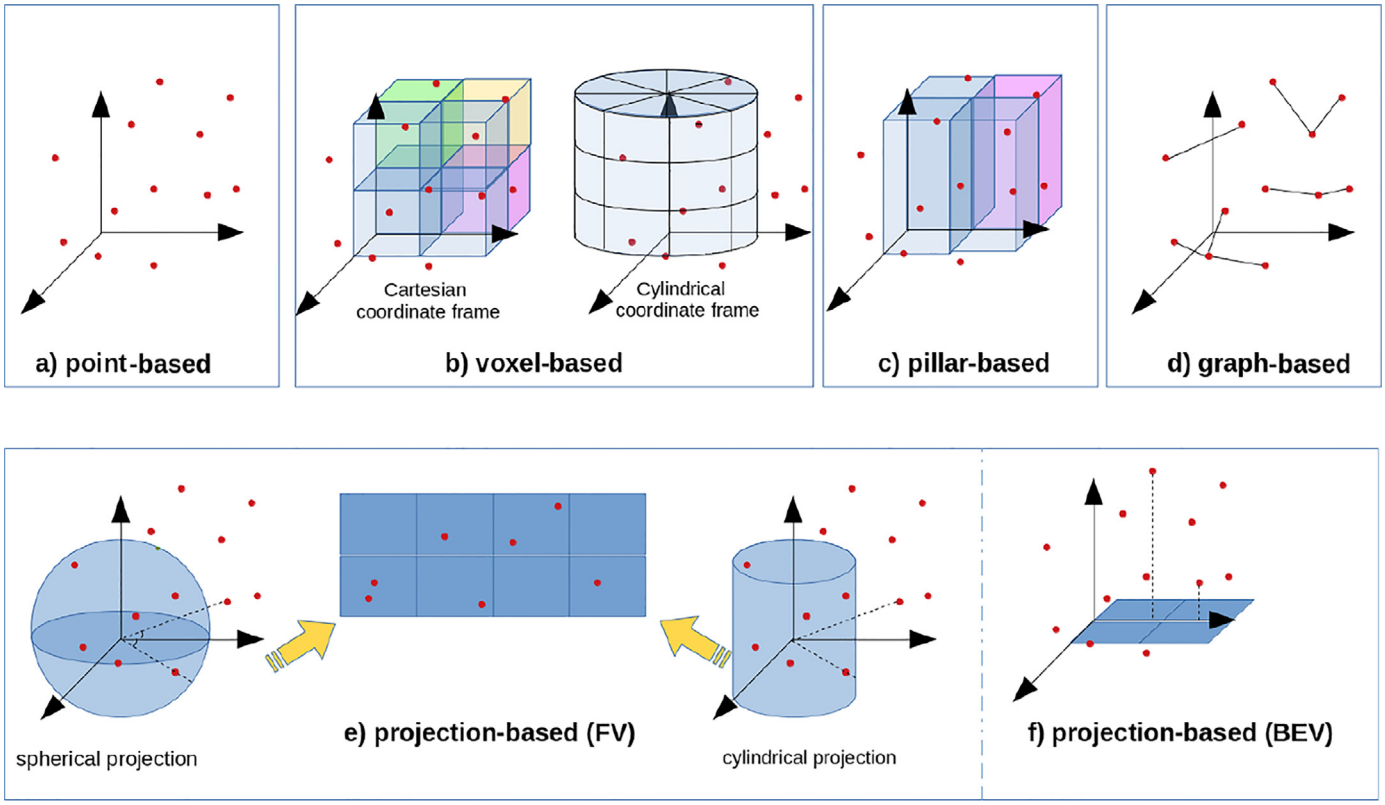


Fig. 2. Point cloud representations.

on the autonomous vehicle, is represented in the form of a point cloud as described in Section 2.1. However, it can not be processed directly by 3D object detectors due to its unstructured form and non-fixed size. First, it has to be encoded into a more compact structure by utilizing five distinct representations: point-based, voxel-based, pillar-based, projection-based and graph-based. Most recent architectures leverage up to two different representations, resulting in a dual representation category.

3.1.1. Point-based:

The incoming LiDAR point cloud retains the unstructured form of a point cloud, however, its representation becomes more compact when transformed into a fixed size. This is accomplished by sub-sampling the point cloud from its original size (i.e. of around 120k points) into a smaller fixed size of N points (i.e. N = 16k points) through two distinct methods of random sampling and Furthest Point Sampling (FPS). In random sampling, points are picked randomly until N points are selected. However, there is a bias in

that approach since points from densely populated areas are sampled more often compared to points from sparsely ones. FPS algorithm mitigates that bias, by picking points according to the farthest distance criterion through an iterative process. At each iteration, FPS first calculates the distances of all unselected points from the last selected point and then picks the point with the farthest distance. The result is a more representative point cloud compared to the incoming one, however, at an increased computational cost.

Every point, of the resulting sub-sampled point cloud, is encoded with the same features, similarly to the incoming LiDAR point cloud.

3.1.2. Voxel-based

Given an incoming LiDAR point cloud $P = \{p_1, p_2, \ldots, p_N\}$ and a 3D space $[L, W, H]$ partitioned into fixed size voxels $[u_L, u_W, u_H]$, voxelization is the process of assigning points to voxels. Partitioning the 3D space according to a Cartesian or cylindrical coordinate frame, results in voxel of a cuboid or of a cylindrical slice shape respectively. The voxelization according to a cylindrical coordinate frame is also referred as Hybrid-Cylindrical-Spherical (HCS) by [14]. A voxel is determined as a zero voxel if no points are assigned to it and as a non-zero voxel when at least one point is assigned to it. There are three distinct ways to perform the point to voxel assignment/mapping, namely fixed voxelization [13], dynamic voxelization [15] and hybrid scale voxelization [16].

Fixed Voxelization: It consists of two stages, the grouping stage and the sampling stage [13]. At first, a fixed buffer of size $VxOxF$ is constructed, where V is the maximum number of voxels, O is the maximum number of points per voxel and F is the size of the feature vector. In grouping stage, all points p_i are assigned to voxels v_j according to their 3D coordinates. Given that a voxel may contain more than O points, the sampling stage is used to randomly sub-sample O number of points from the voxel. If a number of K points, where $(K < O)$, are assigned to a voxel, then zero-padding is applied on the remaining $(O - K)$ buffer indices.

The same applies for voxels. Given that the total number of non-zero voxels is T, where $(T > V)$, then voxels are randomly sub-sampled, otherwise, if $(T < V)$, zero-padding is applied on the remaining $(V - T)$ buffer indices. During fixed voxelization, a fixed buffer size is constructed, however with a certain loss of information during point and voxel drop out.

Dynamic Voxelization [15]: In dynamic voxelization, the points to voxels assignment strategy is similar to the grouping stage of Fixed Voxelization where all points are assigned to voxels. However unlike Fixed voxelization, the sampling stage is not used, therefore the maximum number V of non-zero voxels in 3D space and the maximum number O of points per voxel are not predetermined but change dynamically. This results in a dynamic buffer size without any loss of information, since no voxels or points are dropped.

The point to voxel bi-directional relations are formulated as [15]:

$$\begin{aligned} &F_V(p_i) = v_j, \forall i \\ &F_P(v_j) = \{p_i | \forall p_i \in v_j\}, \forall j \\ &i = \{1, 2, \ldots, P\} \\ &j = \{1, 2, \ldots, V\} \end{aligned} \tag{2}$$

where $F_V(p_i)$ is the mapping function that assigns each point p_i to a voxel v_j and $F_P(v_j)$ is the mapping function that gathers p_i points within a voxel v_j [15].

Hybrid Scale Voxelization [16]: Similar to Dynamic Voxelization, the maximum number V of non-zero voxels in 3D space $[L, W, H]$ and the maximum number O of points per voxel are not predetermined but change dynamically. However, only the point to voxel mapping function is determined, therefore no bi-directional mapping is established. Furthermore, given a total number of different voxel scales S, points are assigned to multiple voxel scales $[u_L^s, u_W^s, u_H^s]$, where $s \in \{1, 2, \ldots, S\}$, i.e. $S = 2$, $u_L^1 = u_W^1 = 0.1m$, $u_L^2 = u_W^2 = 0.2m$. In some works [16], the height of the voxel is fixed for all scales. Point to voxel mapping is determined through an index c_i formulated as [16]:

$$c_i^{(s)} = \left\lfloor \frac{(x_i - x_{min})}{u_L^s} \right\rfloor \left\lfloor \frac{W}{u_W^s} \right\rfloor + \left\lfloor \frac{(y_i - y_{min})}{u_W^s} \right\rfloor \tag{3}$$

The aforementioned Fixed Voxelization, Dynamic Voxelization and Hybrid Scale Voxelization are used to transform the LiDAR point cloud in the spatial domain by assigning points to voxels according to their 3D coordinates. To complete the voxelization process, the next step is to transform the point cloud in the feature domain by extracting a voxel feature.

The most simple voxel feature is the binary occupancy encoding, as in [17], where a binary value of 0 and 1 is assigned to every zero and non-zero voxel, respectively. Another way of extracting a voxel feature is through statistical approaches, such as the mean value of 3D coordinates and reflection intensity [18], or even shape factors [19]. A voxel feature can be also extracted through Deep Learning models. Firstly, points are encoded with their canonical coordinates, reflection intensity values and other statistical based features such as $[x_c, y_c, z_c]$ distance from the arithmetic mean of all points inside the voxel etc. Next, points are introduced to a PointNet-based [20] architecture, resulting in an output feature vector. The most representative example of this architecture is the Voxel Feature Encoding (VFE) [13] module, shown in Fig. 3.

3.1.3. Pillar-based

Introduced by [21], this representation disregards partitioning along Z axis and divides the 3D space $[L, W, H]$ into fixed size pillars $[u_L, u_W, H]$. Intuitively the pillar is seen as an unbound voxel along the Z axis. Similar to voxelization, the assignment of points to pillars is made through three distinct ways of hard voxelization [13], dynamic voxelization [15] and hybrid scale voxelization [16] as described in Section 3.1.2.

Pillar-based features are extracted through Deep Learning models inspired by PointNet [20] following a structure as in VFE module [13]. Since pillars are not partitioned along Z axis, a pillar-based representation of a point cloud is seen as a BEV image of multiple channels. The most representative module for pillar-based feature extraction is presented by PointPillars [21].

3.1.4. Projection-based

In this representation, points from 3D space are projected into a 2D plane under a perspective transformation. Given the perspective view of the 2D plane, projection-based representations are split into Front-View (FV) and Bird's Eye View (BEV) representations.

Front-View projection: The point cloud is projected into a spherical or cylindrical surface with its origin located at the LiDAR sensor. Given a point p_i with 3D coordinates $[x_i, y_i, z_i]$, its spherical coordinate representation $[\phi_i, \theta_i, d_i]$ is computed as:

$$\phi_i = arctan(\frac{y_i}{x_i}), \theta_i = arccos(\frac{z_i}{d_i}), d_i = \sqrt{x_i^2 + y_i^2 + z_i^2} \tag{4}$$

and its cylindrical coordinate representation is computed as:

$$\rho_i = \sqrt{x_i^2 + y_i^2}, \quad \phi_i = arctan(\frac{y_i}{x_i}), \quad z_i = z_i \tag{5}$$

The surface is then unfolded into a 2D plane to form a dense and structured range image. Given a spherical and cylindrical coordinate system, the pixel coordinates $[r_{sph}, c_{sph}]$ and $[r_{cyl}, c_{cyl}]$ of the range image are calculated as:

$$\left[r_{sph} = \left\lfloor \frac{\theta}{\delta r} \right\rfloor, c_{sph} = \left\lfloor \frac{\phi}{\delta c} \right\rfloor \right] \quad , \quad \left[r_{cyl} = \left\lfloor \frac{z_i}{\delta r} \right\rfloor, c_{cyl} = \left\lfloor \frac{\phi}{\delta c} \right\rfloor \right] \tag{6}$$

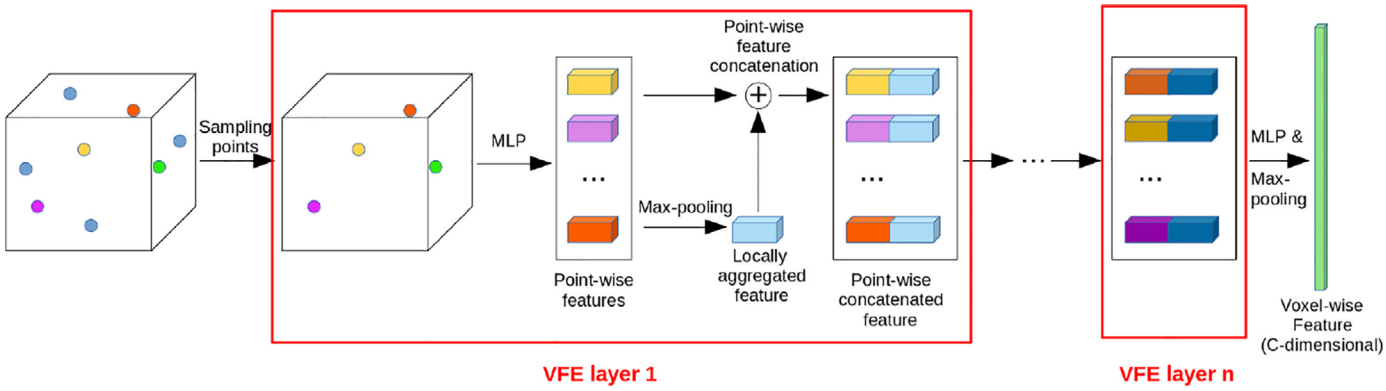


Fig. 3. Block Diagram of Voxel Feature Encoding (VFE) module by VoxelNet [13].

where δr, δc correspond to the range image vertical and horizontal resolutions, respectively.

The number of channels in the FV range image determines the number of initial features. A pixel in the FV range image is usually encoded by hand-crafted or predetermined features such as a binary occupancy encoding, 3D coordinates, distance from sensor, azimuth angle or reflection intensity [22,23] of its corresponding point.

BEV projection: The point cloud is projected into a BEV map of a certain grid resolution of λ meters, by assigning its points to the corresponding grid where the grid may contain more than one points. Given a point with 3D coordinates $[x_i, y_i, z_i]$, its grid coordinate $[r_{BEV}, c_{BEV}]$ in the BEV map is calculated as:

$$r_{BEV} = \left\lfloor \frac{x_i}{\lambda} \right\rfloor, \quad c_{BEV} = \left\lfloor \frac{y_i}{\lambda} \right\rfloor \tag{7}$$

Similar to FV range image, the BEV map is treated as a BEV image and the number of channels determine the number of initial features. A grid/pixel in the BEV image is usually encoded by hand-crafted or predetermined features such as a binary occupancy encoding along fixed size partitions in Z axis [24] and statistic values of points within the grid [25].

3.1.5. Graph-based:

In this representation, the point cloud is converted into a graph. Points are considered as nodes and the connections of a point with its neighbors that are located within a fixed radius d, as edges. Given a point cloud $P = \{p_1, \ldots, p_n\}$, where $p_i = (x_i, r_i)$ is a point with 3D coordinates $x_i \in \mathbb{R}^3$ and initial feature r_i, the graph construction is formulated as [26]:

$$G = (P, E), \quad E = \{(p_i, p_j) \mid \|x_i - x_j\|_2 < d\} \tag{8}$$

The construction of a graph based on raw point cloud is computationally inefficient, therefore a down sampled point cloud is used instead, usually after voxelization [26].

The initial feature for each node, is calculated in a similar way as in [13,21]. First a set of points, within a radius d around a node, are selected. Their canonical coordinates along with their reflection intensity values are processed by a Multi Layer Perceptron (MLP) and through Max operation, an initial feature vector for the node is calculated.

3.2. Feature extraction

Once the LiDAR point cloud is transformed to a structured and compact representation, both in spatial and feature domain, certain techniques and architectures are used to extract rich, high-dimensional features. These techniques are divided into two main categories, 3D Backbone Networks (3DBNs) and 2D Backbone Networks (2DBNs). The first category operates in 3D space, while the second category in 2D space. The selected LiDAR SDR has an influence on the selected technique, i.e. if the point cloud is projected into BEV, a 3DBN could not be used and instead a 2DBN will solely be used for high-dimension feature extraction.

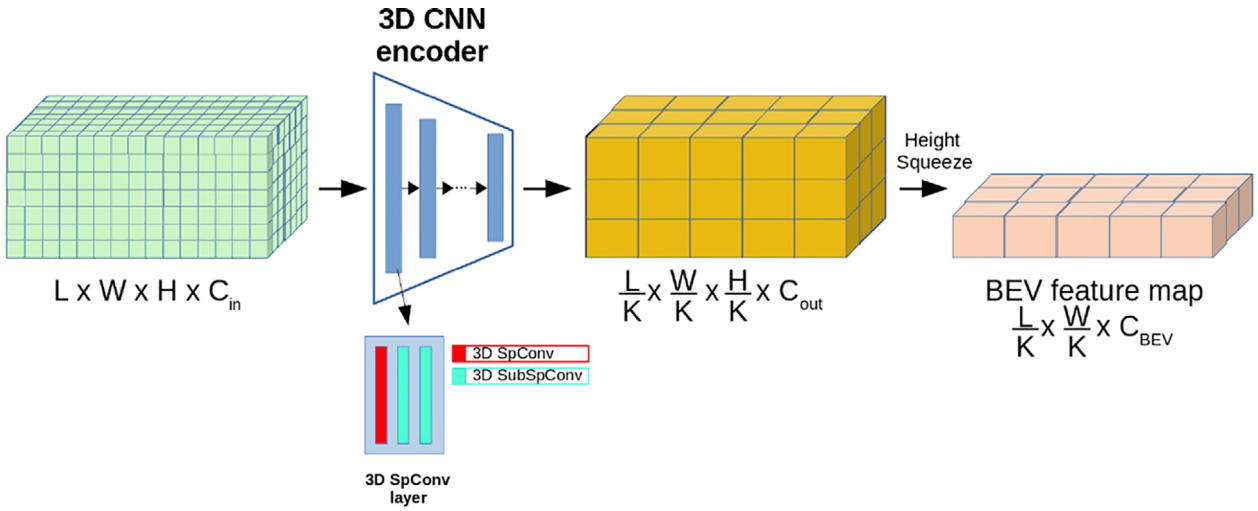
3.2.1. 3D Backbone networks

There are three distinct groups of 3DBNs, according to their building blocks. The first group utilizes Convolutional Neural Networks operating in 3D space (3D CNN), the second group PointNet/PointNet++ [20,27] networks and the third one Graph Neural Networks (GNNs) [28].

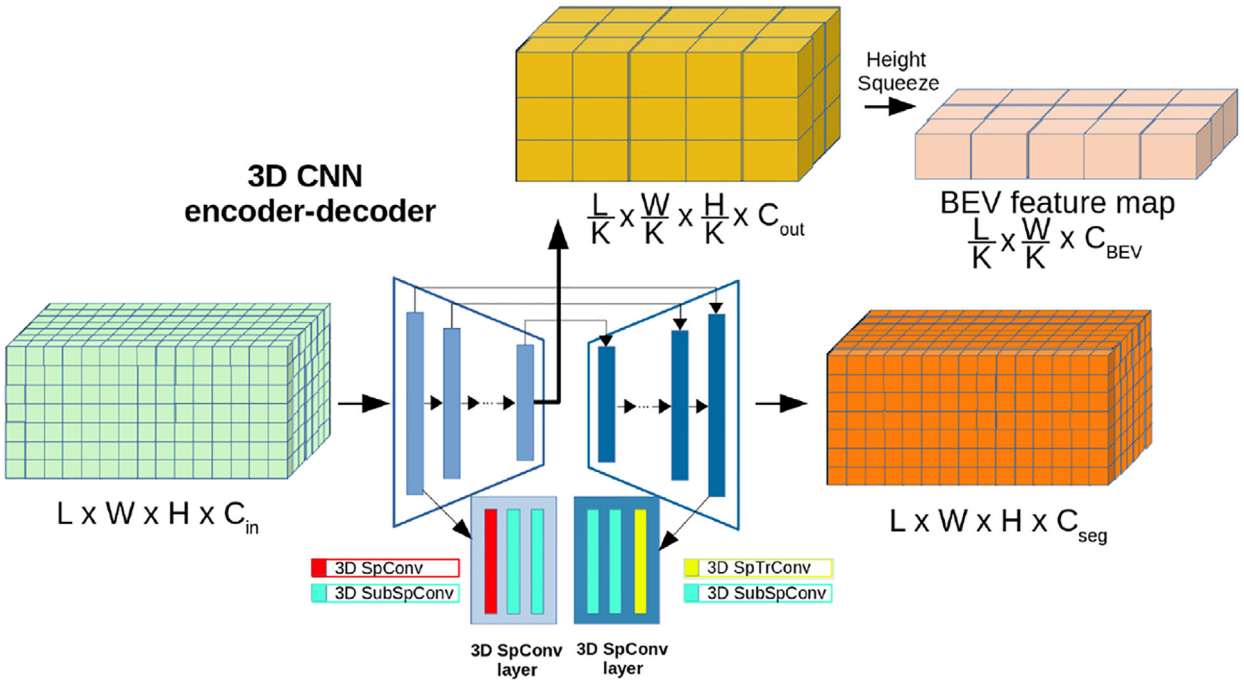
3DBN CNN: Equivalent to 2D convolution in 2D space, 3D convolution is applied on structured data in 3D space, such as a voxelized point cloud for autonomous driving applications [17]. However, the computational burden of 3D convolutions obstructs the usage of 3D CNN in autonomous driving application, where real time performance is non-negotiable. Efficient implementations of spatially sparse convolution [29,30] and submanifold sparse convolution networks [31] are designed and demonstrated to speed up the convolution operation, thus providing the building blocks for 3D Sparse (3D SpConv) and 3D submanifold sparse (3D SubSpConv) convolution networks. In a similar approach, an efficient voting scheme for applying a sliding window in 3D data is demonstrated by [32].

Attempting to presume upon the sparsity of LiDAR point clouds, since around 90% of voxels are empty of points [13], Vote3Deep [19] utilize the voting scheme by [32], while SECOND [33] utilize the forementioned 3D SpConv and 3D SubSpConv to construct a 3D sparse CNN-based backbone network for 3D object detection in LiDAR point clouds.

The design of the 3DBN sparse CNN follows an encoder or encoder/decoder architecture. An encoder architecture applies an adequate number of multiple consecutive layers of 3D SpConv with 3D SubSpConv followed by Batch Normalization [34] and Relu [35]. The result is a down-scaled, rich in features voxelized 3D space. Subsequently, a height compression is performed to concatenate features along Z axis and construct a high dimensional BEV feature map. An encoder/decoder architecture follows the same structure concerning the encoder part, while for the decoder part, an adequate number of multiple consecutive layers of 3D SubSpConv with 3D sparse transpose convolution (3D SpTrConv) followed by Batch Normalization [34] and Relu [35] are applied. The result is an upsampled voxelized 3D space of similar spatial dimensions as the initial one, with each voxel assigned to a semantic label. A block diagram for both encoder and encoder/decoder 3D CNN architectures is shown in Fig. 4.



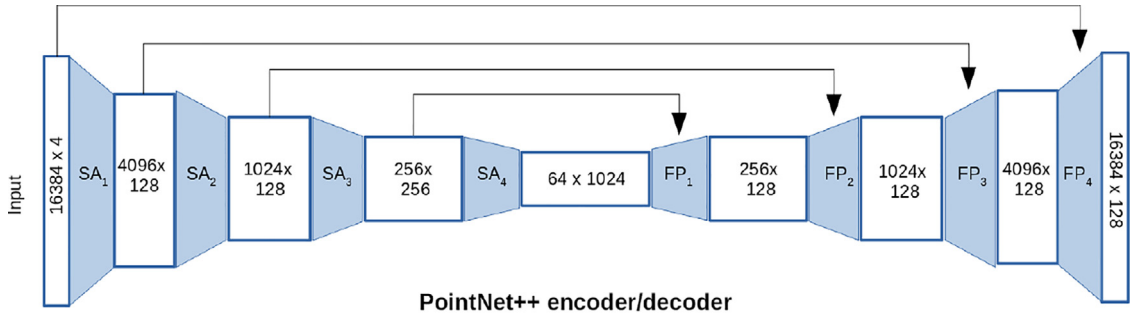
(a)



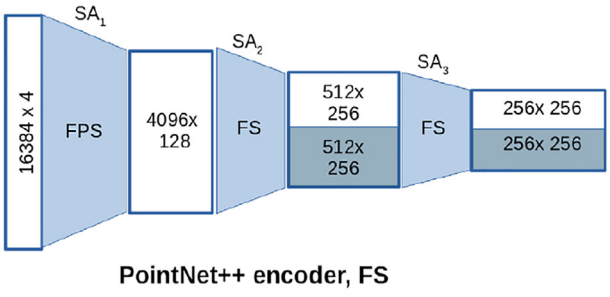
(b)

Fig. 4. Block diagrams for two different 3DBN CNN architectures. (a) A 3D sparse CNN encoder similar to SECOND [33]. (b) A 3D sparse CNN encoder-decoder architecture as in Parts A^2 Net [36].

3DBN PointNet++: Given a point cloud as an input, PointNet++ [27], in its segmentation form, is able to extract both semantic segmentation scores and global context features for each point. Therefore it is ideal for extracting features from LiDAR point clouds. In its semantic form, PointNet++ consists of several Set Abstraction (SA) layers following an encoder architecture, and of several Feature Propagation (FP) layers following a decoder architecture. Given an initial number of points as centroids, the SA layer utilize the FPS algorithm for selecting a fixed number of points within a ball query area (around a centroid point), and then applies a PointNet [20] to extract a local feature vector. Given the learned features from SA layers, the FP layers propagate the learned multi-scale features back to all points through interpolation. The usage of a semantic PointNet++ as a 3DBN for feature



(a)



(b)

Fig. 5. Block diagrams for two different Backbone 3D PointNet++ architectures. (a) A 3D PointNet++ encoder/decoder similar to STD [37]. (b) A 3D PointNet++ encoder architecture with Fusion Sampling as in 3DSSD [43].

extraction in autonomous driving application is demonstrated by many 3D object detectors [37–42].

In an attempt to remove the FP layers for saving computational resources, a related work [43] uses solely the encoder part of a PointNet++ [27]. To compensate the lack of the decoder part, a new point sampling algorithm is introduced in the set abstractions (SA) layers of PointNet++ [27], named as Fusion Sampling (FS). FS is a sampling strategy that utilizes two sampling algorithms, the FPS and the Feature-FPS (F-FPS). F-FPS calculates the feature distance between points at the end of each SA layer, and uses it as a criterion for selecting points in an attempt to increase the selection of foreground points over background ones. FS samples half of the points based on the euclidean distance (FPS) and the other half based on the feature distance (F-FPS) in an attempt to find a balance between the two FPS algorithms. A block diagram for both semantic 3DBN PointNet++ encoder/decoder [40] and 3DBN PointNet++ encoder FS [43] for 3D object detection, is shown in Fig. 5.

3DBN Graph: Given a graph point cloud representation, a GNN [28] is used to learn per node features, by aggregating features along the edges. The GNN is executed for a fixed number of T iterations where the node feature from t^{th} iteration is used as an input feature for the $(t+1)^{th}$ iteration. MLPs and Max operations are used to extract node features similar to PointNet [20]. The MLP weights are not shared among iterations but instead they are learned individually for each iteration $t = [1, 2, ., T]$.

3.2.2. 2D Backbone networks

The 2D Backbone Network (2DBN) is a Fully Convolutional Network (FCN) applied on structured data in 2D space, such as BEV and FV range images, to extract rich, high-dimensional feature maps.

There are different architectures for a 2DBN, most of them inspired by four main categories of Feature Pyramid Networks (FPN) [44], U-Net [45], ResNet [46] and VGG [47] networks. The building blocks of a 2DBN usually consist of 2D convolution (2D Conv) layers followed by Batch Normalization [34] and Relu [35], however, for certain 3D object detectors the 2D Deformable [48] (2D DefConv) or 2D Dilated Convolutions (2D DilConv) are also applied [22,49]. The 2D Transpose Convolution (2D TrConv) is used for upsampling the feature maps.

It is observed that in the implemented 2DBNs of 3D object detectors, the number of input and output feature maps, for a 2DBN following an FPN architecture, is not always the same. Therefore the following terms are introduced:

- SISO: Single Input feature map, Single Output feature map
- SIMO: Single Input feature map, Multiple Output feature maps
- MIMO: Multiple Input feature maps, Multiple Output feature maps
- MISO: Multiple Input feature maps, Single Output feature map

To provide a compact presentation of 2DBNs, their architectures along with the feature map that the 2DBNs are applied on, are en-

coded through a unified scheme, i.e. BEV(FPN SIMO). This refers to a 2DBN with FPN SISO architecture, applied on a BEV feature map.

In Fig. 6, the most representative architectures for 2DBNs, as implemented by 3D object detectors, are presented. A 2DBN of BEV (FPN SISO) architecture, as implemented by VoxelNet [13] and SECOND [33], performs upsampling to a fixed size followed by concatenation, to construct the high dimensional BEV feature map. In the implementation of RangeRCNN [22], the 2DBN follows a FV (FPN SISO) architecture, used to extract a high-dimensional FV feature image of same spatial dimensions as the input one. A 2DBN of BEV (FPN MIMO) architecture is implemented by Voxel-FPN [50] to extract high-dimensional BEV feature maps at multiple scales. It is divided into the early and later fusion stages.

3.3. Core object detection

The Core Object Detection module is used to process the extracted high-dimensional features from the Feature Extraction module and provide as an output classification confidence scores and regression values concerning the object class and the location and size of detected object in the form of a 3D BBox, respectively. It is split into two stages, Detector Networks and Prediction Refinement.

3.3.1. Detector networks

Detector Networks are categorized into two main approaches, namely Anchor-based and Anchor-free.

Anchor-based networks utilize multiple anchors of predefined sizes, to search through a high-dimensional feature map, output a classification score and regress an object's location and size relative to the anchor's location and size.

Anchor-free networks on the other hand, do not use predefined anchors, but instead utilize binary labelling information from the Feature Extraction module, to identify points or areas/parts that belong to an object. Then for each point or part of an object, an object prediction is made, consisting of a class confidence score and a 3D BBox.

Anchor-Based network: The most representative architecture of an Anchor-based network is the Region Proposal Network (RPN) [4]. In 3D object detectors, the RPN receives as an input one or more high-dimensional BEV feature maps and outputs a proposal concerning the class, 3D location, size and orientation of an object. Similar to predefined 2D anchors and regions of interest as in [4,5], 3D anchor boxes of a fixed 3D size for each object class, are used. The anchor box size is selected according to the natural dimensions of each class, e.g. in [13] for KITTI [51] dataset and for a 'car' class, an anchor size of length (l), width (w), height (h) l = 3.9m, w = 1.6m, h = 1.56m is selected while for 'pedestrian', an anchor size of l = 0.8m, w = 0.6m, h = 1.73m is used.

Efficient 1x1 convolutional layers are applied separately on the input high-dimensional BEV feature map (or maps) for each task of classification, object location/size, and direction, resulting in probability score, and regression maps. The output of an RPN is multiple 3D BBox proposals, many of which will overlap and correspond to the same object. To remove redundant 3D BBox proposals, Non Maximum Suppression (NMS) [52] is used as a post processing step.

In a different approach, instead of operating in BEV, the work in [37] introduces the concept of a spherical anchor in 3D space. For each point, a spherical anchor is created of predetermined dimensions and NMS is applied to reduce the total number of anchors. Next, a PointNet [20] network is applied on each spherical anchor to predict a 3D BBox. Finally an additional NMS step is applied to remove redundant 3D BBox proposals.

Anchor-Free network: A binary labelling information from Feature Extraction module is utilized by an Anchor-Free network, to identify points or parts of an object and predict a per point/part 3D BBox proposal. Anchor-Free networks are divided in those operating in the 3D or in the 2D space.

For networks operating in 3D space, a common trend is to learn a binary semantic label (background/foreground) for each point in the Feature Extraction module [36,37,39,40]. Then, through fully connected (FC) layers a 3D BBox prediction is made per point. For all aforementioned Anchor-Free networks, several proposals are calculated for each object consisting of multiple foreground points/voxels, therefore, Non Maximum Suppression (NMS) is still needed to remove redundant 3D BBox proposals. In an attempt to completely remove the NMS post processing step, the method by [38] feeds the per point binary semantic information along with the per point Spatial Embeddings (SE) information (both of them learned from 3DBN) to a simple clustering algorithm (i.e. K-means) to perform instance segmentation and therefore, make only one 3D BBox proposal per instance.

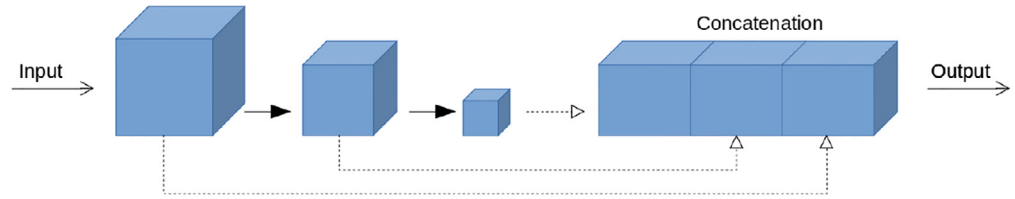
Anchor-Free networks can also operate in 2D space. In an attempt to leverage the intra-object location of every foreground grid, [53] predicts two separate BEV maps, one for foreground classification and one for 3D BBox regression. Each foreground is further encoded according to k classes which correspond to partitions of an object. Inspired by CenterNet [54] a 2D anchor-free object detector for 2D images, methods presented by [49,55,56] extend the same concept into a 3D object detector, by treating 3D objects as center points in BEV. An Anchor-free approach is utilized by [24], however objects are treated as 2D BBoxes in BEV instead. In all aforementioned Anchor-Free networks operating in 2D space, the network follows a FCN architecture. The output BEV feature map from 2DBN is processed through shared [24] or parallel FCN blocks [49], for classification, BBox regression etc. Each FCN block consists of sequential 2D Conv layers. Then, for each task, 1x1 convolutions are applied for generating the confidence scores and BBox regression maps. For [49,55] the NMS post processing step is no longer needed since for each object only one center is predicted and therefore, only one 3D BBox proposal is made, thus saving computational resources.

3.3.2. Prediction refinement

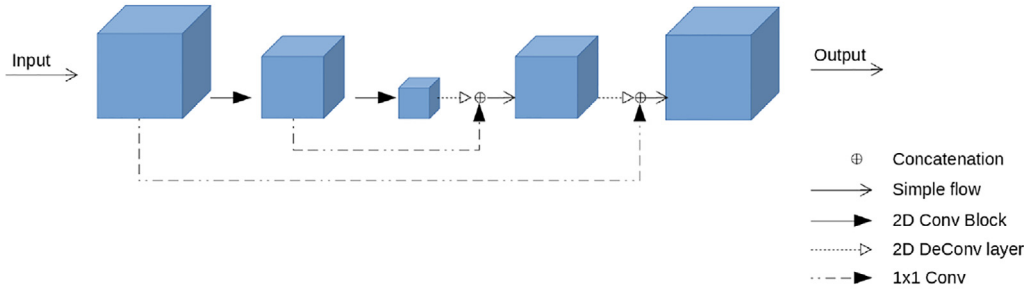
Prediction Refinement (PR) is used by two-stage 3D object detectors. A PR receives as input the proposed 3D BBox from Detector Networks, samples it at a fine-grained level and extracts features to improve the classification confidence score and the 3D BBox location, size and orientation. PRs are divided into two main categories, those operating in 3D space and those operating in 2D space. PRs that operate in 3D space are further divided into three sub categories concerning their sampling representation, into point-based, graph-based and voxel-based.

3D point-based PR utilize the natural representation of the point cloud and sample a fixed number of points from the proposed 3D BBox. Points are encoded with their canonical coordinates, reflection intensity and learned foreground score [42] while other architectures additionally encode points with their distance from the sensor and learned global context features [40]. The points are then introduced to a PointNet [20] or PointNet++ [27] architecture as demonstrated by [57] and [40] respectively, to refine the class confidence score and 3D BBox.

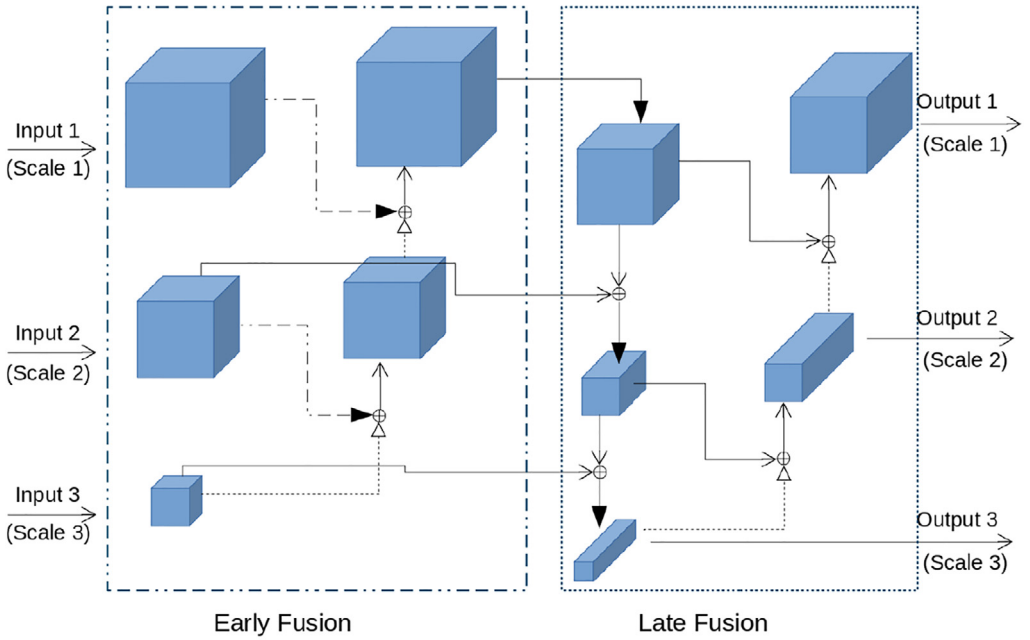
3D graph-based PR have been demonstrated by [41] and consist of two modules, one for extracting local features per proposal/object (R-GCN) and one for extracting global features per frame (C-GCN). The R-GCN module samples the proposed 3D BBox and encodes features to points as in [40]. Next, given the selected points, a graph is constructed and processed through MRGCN [58] layers to output a local feature vector. The C-GCN module constructs a graph where each proposal/object is considered a node and encoded with its local feature vector, learned from



(a)



(b)



(c)

Fig. 6. Block diagrams for three different 2DBNs. (a) BEV (FPN SISO) architecture, similar to VoxelNet [13], SECOND [33]. (b) FV (FPN SISO) architecture, similar to RangeR-CNN [22]. (c) BEV (FPN MIMO) architecture similar to Voxel-FPN [50].

R-GCN. Then, the graph is processed through EdgeConv [59] layers to learn a global feature for all proposals. For each proposal, the global feature from C-GCN is concatenated with the local feature from R-GCN and then introduced to two fully connected layers for refining the class confidence score and 3D BBox.

3D voxel-based PR partitions the proposed 3D BBox into a fixed number of voxels along X,Y and Z axis. Points within the proposed 3D BBox are converted into canonical coordinates and encoded with learned high-dimensional features [18] from Feature Extraction module and foreground scores if present [36]. The feature of each voxel is extracted either through VFE-based [13] architectures as in [37,60] or through specially designed feature fusion algorithms as in [36].

Voxels are processed through 3D SpConv layers resulting in a downscalled 3D space as in [36], converted to a BEV feature map through 3D Conv as in [60], or directly flattened to one-dimension as in [22]. Resulting representations in 3D or 2D space, are flattened to one dimension and fully connected layers are applied for refining the class confidence score and 3D BBox.

2D based PR operate in BEV to improve the 3D BBox proposal. In its early implementation, [25] receives a 2D BBox proposal and through a hand-crafted approach it extracts a Z value to convert the 2D BBox into a 3D BBox, by estimating a coarse ground plane and extracting the maximum height through the corresponding channel of the BEV feature map. In its latest implementation, [61] replaces the hand-crafted approach with fully connected layers. In a different approach, [62] splits every object into k^2 parts in BEV and adopts the Faster-RCNN [4] strategy to learn pose-sensitive feature maps, for refining the class confidence score and 3D BBox.

4. Analysis of 3D object detection methodologies

In this section, recent 3D object detectors are categorized according to their LiDAR SDR and presented individually, each of them in a single subsection. The methodology analysis and presentation for each 3D object detector is performed in relation to the operational pipeline modules, as discussed in Section 3. Furthermore, extra notice is given to the individual and more specific particularities of each methodology, which could not be thoroughly categorized and described through the common operational pipeline, along with the motivation of the method.

4.1. Voxel-based

4.1.1. Vote3Deep

Vote3Deep [19] is a one-stage detector that first utilizes the concept of sparse convolutional layers and L1 regularization on 3D data, for object detection in autonomous driving applications.

The LiDAR point cloud is voxelized and each voxel is encoded with a binary occupancy value and hand-crafted features. Inspired by Vote3D [32], Vote3Deep utilizes a voting scheme through sparse convolutional layers, thus forming its 3DBN module for feature extraction in 3D voxelized space.

The last layer of the 3DBN corresponds to a 3D sparse convolution of a proper class-dependent kernel size to output a classification score and 3D BBox regression values, thus it can be seen as an Anchor-based network. Finally NMS is applied for removing redundant proposals among intersected ones.

4.1.2. 3D FCN

3D FCN [17] is a one-stage detector that utilizes the concept of a FCN architecture for 3D data, motivated by the increased performance of FCN architectures for 2D object detection tasks. The first step of 3D FCN is to voxelize the LiDAR point cloud and encode each voxel with a binary occupancy value.

Then a 3DBN architecture, consisting of three sequential 3D Conv layers and two parallel 3D TrConv layers, are used to learn two 3D heatmaps, one for a confidence score and one for 3D BBox regression, through an Anchor-free approach.

4.1.3. PIXOR

PIXOR [24] is a one-stage detector focused in computational efficiency, that utilizes solely a 2D FCN architecture to detect 3D objects in BEV, without the computationally expensive 3D convolutions in its structure.

The LiDAR point cloud is voxelized and each voxel is encoded with a binary occupancy value. Given a 3D space of size $[L, W, H]$ and a voxel size of $[u_L, u_W, u_H]$, the voxelized space is then formed into a BEV image of spatial size $[\frac{L}{u_L}, \frac{W}{u_W}]$ and of $[\frac{H}{u_H}]$ channels as well as one more channel that corresponds to the normalized reflection intensity.

The BEV image is fed to a 2DBN following an FPN [44] SISO architecture, to construct a high-dimensional BEV feature map. An Anchor-free network operating in 2D, receives as an input the high-dimensional BEV feature map and outputs a classification score map and a 2D BBox regression map. Finally NMS is applied to remove redundant proposals.

4.1.4. HDNet

HDNet [63] is an extension of PIXOR [24] where semantic priors from High Definition (HD) maps are utilized for improving the detector's performance.

Every LiDAR point p_i with initial coordinates $[x_i, y_i, z_i]$, is encoded with new coordinates as $[x_i, y_i, z_i - z_o]$, where z_o is the height offset of the ground at the given $[x_i, y_i]$ location, obtained from the HD map. The rest of the structure is similar to PIXOR's [24].

If HD maps are not available, HDNet [63] proposes to construct the map priors, online from the LiDAR point cloud. This is accomplished through an extra network consisting of two separate 2DBN following a U-net [45] architecture, one for ground estimation and one for road segmentation.

4.1.5. VoxelNet

VoxelNet [13] is a one-stage detector that introduces the concept of VFE for voxel feature extraction, as discussed in Section 3.1.2, inspired by the ability of PointNet [20] to capture 3D shape information directly from points.

The LiDAR point cloud is voxelized through fixed voxelization and each non-zero voxel is encoded with a feature vector extracted from the VFE module, shown in Fig. 3.

A 3DBN following a 3D CNN encoder architecture, is applied on the voxelized 3D data to extract high-dimensional features. The resulting 3D feature map is concatenated along Z axis to construct a BEV feature map and fed to an 2DBN, following an FPN[44] SISO architecture, for further feature extraction. The resulting high-dimensional BEV feature map is then fed to an Anchor-based RPN inspired by [4] to construct probability score and 3D BBox regression maps.

4.1.6. SECOND

SECOND [33], which stands for Sparsely Embedded Convolutional Detection, is a one-stage detector that implements an efficient 3D sparse convolution backbone for 3D object detection in autonomous driving, motivated by the computational efficiency of 3D spatially sparse and submanifold convolutions [29–31]. Furthermore, it introduces a new form of angle loss regression along with a data augmentation technique for KITTI [51] dataset.

Similar to VoxelNet [13], SECOND voxelizes the LiDAR point cloud through fixed voxelization and extracts voxel features through the VFE module.

A 3DBN following a 3D sparse CNN encoder architecture is used to construct a high-dimension BEV feature map, as shown in Fig. 4. The BEV feature map is processed through a 2DBN following an FPN [44] SISO architecture and then fed to an Anchor-based RPN inspired by Single Shot Detector (SSD) [5] to construct three different maps, one for a probability score, one for 3D BBox regression and one for a direction score.

Concerning the yaw angle regression, SECOND [33] implements a sine-error loss instead of a radian-error loss, to remove the large loss found in the case of 0 and π angle since for both cases the 3D BBox is similar. To solve the direction problem that arises from that sine-error loss, a binary direction classifier is predicted by the network.

Concerning data augmentation for KITTI [51] dataset, SECOND [33] constructs a ground truth database by cropping the point cloud around objects, given a ground truth 3D BBox. During training, at each point cloud scene, objects from the ground truth database are randomly chosen and inserted via concatenation in the point cloud. At every insertion of an object, a collision check is performed to avoid spatially inconsistent outcomes, i.e. a car placed within another car.

4.1.7. 3D Backbone Network

3D Backbone Network (3D BN-Net) [64] is a one-stage detector that utilizes a 3DBN sparse CNN FPN-based [44] architecture for constructing BEV feature maps at multiple scales of the 3D voxelized space.

The LiDAR point cloud is voxelized with a fine voxel resolution $[u_L, u_W, u_H]$ of [0.025,0.025,0.0375] meters and each voxel is encoded with a binary occupancy value.

A 3DBN with 3D sparse CNN and an FPN-based[44] SIMO architecture is used to output four BEV feature maps at four different scales of the voxelized 3D space.

The four BEV feature maps are fed to a 2DBN, where they are first upsampled to a fixed size through 2D TrConv and then concatenated to produce a BEV feature map of fixed spatial size. The resulting BEV feature map is then fed to an Anchor-based RPN to output three different maps, one for classification score, one for 3D BBox regression and one for orientation.

4.1.8. Fast Point RCNN

Fast Point RCNN [57] is a two-stage detector that utilizes a voxel-based representation in its first stage for initial 3D BBox proposal, while switching to a point-based representation in its second stage for 3D BBox refinement. The reasoning behind this strategy is the computational efficiency of convolutions and the ability of point-based networks to capture fine-grained 3D shape information.

The LiDAR point cloud is voxelized through a simplified voxelization process of VoxelNet[13] where a maximum number of six points are selected per voxel, and an 8-channel MLP is used to extract the voxel feature vector.

A 3DBN following a 3D sparse CNN encoder architecture is used to extract high-dimension features and construct the BEV feature map. The BEV feature map is then fed to a 2DBN similar to [13] and then to an Anchor-based RPN as in [33], resulting in classification score and 3D BBox regression maps.

In its second stage, Fast Point RCNN utilizes a 3D point-based PR, named as RefinerNet. Each point that lies inside the proposed 3D BBox, is projected to BEV and decorated with the high-dimensional features from the 2DBN. The canonical point coordinates are fused with the encoded features through an attention mechanism. Finally a PointNet [20] network is used to refine the proposed 3D BBox.

4.1.9. MEGVII

MEGVII [65] is a one-stage detector that focuses on the class-imbalance problem, which is quite severe in nuScenes [66] dataset. To mitigate class imbalance MEGVII proposes a new dataset sampling strategy to provide a smoother class-balance, along with the construction of a new multi-head group network as the last part of an Anchor-based RPN. MEGVII ranked first in the 2019 nuScenes 3D Detection Challenge [67].

The LiDAR point cloud is voxelized and the voxel feature is calculated as the mean values of points within it. Next a 3DBN following a 3D sparse CNN encoder architecture as in [33], is used to construct a BEV feature map. A 2DBN and an Anchor-based RPN similar to [13] are used to further process the BEV feature map.

The last part of the RPN is the multi-group head network. Instead of predicting directly the 10 classes of nuScenes [66] dataset, it is decided through a manual selection to cluster classes of common shape and size while keeping the rest distinct to ease the class-imbalance problem. Therefore, classification is split into a two-step approach. In the first step, 6 specific hyper-classes are predicted, namely [(Car), (Truck, Construction Vehicle), (Bus, Trailer), (Barrier), (Motorcycle, Bicycle), (Pedestrian, Traffic Cone)]. Then for each of the specified hyper-classes, a second step is performed to further classify the object.

4.1.10. Patch Refinement

Patch Refinement [60] is a two-stage detector composed of two independently trained VoxelNet [13] based networks. The motivation of this method is to save computational resources by discarding processing on the sparse and empty areas of a LiDAR point cloud. This is accomplished by using a first coarse grained network to provide a prediction/patch that will be processed by a second network at a finer detail.

The first stage of [60] is a light version of VoxelNet [13] and is used to voxelize the point cloud, extract voxel features through a VFE module, construct a BEV feature map through a 3DBN, process the BEV feature map through a 2DBN and provide a 3D BBox prediction through an Anchor-based RPN.

In its second stage, the proposed prediction/patch is fed to a similar but more detailed and fine-grained network as in its first stage.

4.1.11. SARPNET

SARPNET [68], which stands for Shape Attention Regional Proposal Network, is a one-stage detector that utilizes a shape attention branch as the last part of its RPN, to presume upon well-combined 3D shape priors.

The LiDAR point cloud is voxelized as in SECOND[33], however with two main differences. First, voxels are expanded in all three directions by a small offset, therefore, creating an overlapping area among them. Secondly, an algorithm similar to FPS is created, to avoid sampling points within a voxel that are too close to each other. A light version of a VFE module, similar to [21] is used for voxel feature extraction.

A 3DBN composed of a single 3D submanifold sparse convolution [31] layer is applied on the voxelized 3D data followed by a feature concatenation along Z axis to construct the BEV feature map. The BEV feature map is fed through a 2DBN as in [33] to extract high-dimension features. The output feature map from the 2DBN is fed to an Anchor-based shape attention RPN, consisting of three attention branches, namely the Encoding branch, the BEV attention branch and the Vertical attention branch. The encoding branch consists of a 1x1 convolution layer. The BEV attention branch consists of two convolutional layers, the BEV encoding layer used for feature learning, and the BEV attention layer for providing weights based on a predefined area that mimics an anchor. The Anchor-based shape attention RPN outputs three maps, one for a

Table 1
Characteristics of Autonomous Driving Public Datasets .

Name	Year	Scenes	Size (hours)	Annotated Frames	Annotated 3D BBoxes	Object Classes	LiDAR Sensors	Camera Sensors	Online Benchmark	Lighting Conditions	Weather Conditions
KITTI [51]	2012	22	1.5	15k	200k	8	1xVelodyne HDL-64	2xStereo	Yes	day	sunny, cloudy
ApolloScape [89]	2018	-	100	80k	70k	6	2xRiegl VMX-1HA	2	Yes	day	various weather
Lyft [90]	2019	366	2.5	46k	1.3M	4	1x64-beam roof, 2x40-beam bumper	7	No	day	various weather
H3D [91]	2019	160	0.77	27k	1.1M	8	1xVelodyne HDL-64S2	3	No	-	-
nuScenes [66]	2019	1k	5.5	40k	1.4M	23	1xVelodyne HDL-32	6	Yes	day, night	various weather
Waymo [12]	2019	1150	6.4	230k	12M	4	1x360° 75m range, 4xHoneyComb 20m range	5	Yes	day, night	various weather
A2D2 [92]	2020	18	-	41.2k	42.8k	14	5xVelodyne VLP-16	6	No	day	various weather
KITTI-360 [93]	2020	9	-	81.1k	63k	19	1xVelodyne HDL-64, 1xSICK LMS 200	1xStereo, 2x180°	Not Yet	-	-

probability score, one for 3D BBox regression and one for direction classification as in [33].

4.1.12. FoF-Net

FoF-Net [69], which stands for Fusion of Fusion Network, is a one-stage detector that utilizes an FPN[44] architecture in its 2DBN to fuse BEV feature maps at different scales, to extract more robust features.

To further improve the computational efficiency of voxelization, the VFE module is discarded and the voxel feature is calculated as the mean values of points within it. Next, a 3DBN with 3D sparse CNN following an encoder architecture as in [33] is used to construct a BEV feature map and feed it as an input to the 2DBN. The 2DBN module follows an encoder-decoder FPN[44] architecture, and named as FoF strategy. The resulting high-dimensional BEV map from 2DBN is fed to an Anchor-based RPN to generate probability score and 3D BBox regression maps.

4.1.13. Part A^2 net

Part A^2 Net [36], which stands for Part Aware and Part Aggregation Network, is a two-stage detector that is implemented under two versions, an Anchor-based, and an Anchor-free, which share the same architecture principles.

The LiDAR point cloud is voxelized and the voxel feature is calculated as the mean values of points within it. A 3DBN following a sparse 3D CNN U-Net[45] encoder-decoder architecture as in Fig. 4, is used to create a BEV feature map and also provide per voxel (and therefore, per point) intra-object part locations along with a foreground/background binary score. The architecture of the 3DBN aims to encode the 3D geometric features of an object in a more effective and robust way.

The Anchor-based version of Part A^2 Net utilizes a 2DBN and an Anchor-based RPN as in [33]. The Anchor-free version of Part A^2 Net utilizes the per point learned intra-object part locations and binary semantic score from the decoder part of 3DBN, and applies an extra branch of fully connected layers to generate a 3D BBox proposal for foreground points only, similar to [40].

For its second stage, Parts A^2 uses a 3D voxel-based prediction refinement (PR). The proposed 3D BBox is voxelized and points are assigned to each voxel up to a maximum number. Every point is then encoded with its canonical coordinates and features are extracted from the decoder part of 3DBN. The feature vector for each voxel is derived through a fusion algorithm among the individual point feature vectors. Then multiple layers of 3D SpConv are applied on the 3D voxels resulting in high-dimension voxel feature extraction along with 3D space downscaling. Finally, the downscaled 3D spaced is flattened to one-dimension space and fully connected layers are applied to predict a confidence score together with a 3D BBox refinement.

4.1.14. SegVoxelNet

SegVoxelNet [70] is a one-stage detector which incorporates a semantic context encoder in its 2DBN, to learn binary semantic masks of object parts, along with a depth aware detection head to presume upon the diminishing point distribution of the point cloud as the distance from LiDAR sensor increases.

The LiDAR point cloud is voxelized and the voxel feature is calculated as the mean values of points within it. Next, a 3DBN with 3D sparse CNN following an encoder architecture as in [33], is used to extract high dimensional features and construct the BEV feature map.

The BEV feature map is fed to a 2DBN consisting of two networks, one following a U-Net [45] architecture used to extract high-dimensional features, and one following an FPN [44] architecture used for predicting BEV semantic masks. The output BEV feature maps from the two networks are fused together through a residual attention re-weighting mechanism as in [71].

An Anchor-based network with a depth aware detection head receives the fused BEV feature map from 2DBN, splits the map in three sub-maps according to distance from LiDAR sensor and applies for each sub-map an extra 2D Conv block. This 2D Conv block is not shared among sub-maps. For each sub-map, three sibling 1x1 convolution layers are applied, resulting in probability score, 3D BBox regression and direction classification maps.

4.1.15. HotSpotNet

HotSpotNet [53] is a one-stage detector that leverages the concept of modelling an object as a composition of its sub-parts, defined as hotspots, to encode effectively the 3D shape and geometric information of objects. All non-empty voxels that are located within a ground truth 3D BBox are selected as hotspots. Then every hotspot is assigned to its proper class according to which object sub-part it belongs to.

The LiDAR point cloud is voxelized by encoding each voxel with the mean coordinates and mean reflection intensity of points within. Next, a 3DBN with 3D sparse CNN following an encoder architecture, as in [33], is used to extract high-dimensional features and construct the BEV feature map. In the implementation for KITTI [51] and nuScenes [66] a 2DBN following a FPN[44] SISO and ResNet [46] architecture, respectively, is used for further feature extraction on the BEV feature map.

The resulting high-dimensional BEV feature map from 2DBN is fed to an Anchor-free network for hotspot classification and 3D BBox regression. A 3D BBox is predicted for each hotspot, thus NMS is still needed to remove redundant proposals. A spatial relation encoding network is utilized during training to assist and supervise HotSpotNet for learning spatial relations between hotspots and is removed during inference.

Table 2
3D Object Detectors Evaluation on KITTI 3D and BEV Benchmarks.

	3D (mAP)									BEV (mAP)										
	Car			Pedestrian			Cyclist			Car			Pedestrian			Cyclist				
Name/Year	Easy	Mod.	Hard	Easy	Mod.	Hard	Easy	Mod.	Hard	Easy	Mod.	Hard	Easy	Mod.	Hard	Easy	Mod.	Hard	Time (s)	Hardware
Voxel-based:																				
✓Vote3Deep [19] (2017)	76.79	68.24	63.23	**68.39**	**55.37**	**52.59**	79.92	67.88	**62.98**	-	-	-	-	-	-	-	-	-	1.1	n/a
✓3D FCN [17] (2017)	84.20	75.30	68.00	-	-	-	-	-	-	70.62	61.67	55.61	-	-	-	-	-	-	1.0	n/a
PIXOR [24] (2018)	-	-	-	-	-	-	-	-	-	83.97	80.01	74.31	-	-	-	-	-	-	0.093	Titan XP
HDNet [63] (2018)	-	-	-	-	-	-	-	-	-	93.28	86.01	80.11	-	-	-	-	-	-	0.035	n/a
✓VoxelNet [13] (2018)	77.47	65.11	57.73	39.48	33.69	31.51	61.22	48.36	44.37	89.35	79.26	77.39	46.13	40.74	38.11	66.7	54.76	50.55	0.033	Titan X
✓SECOND [33] (2018)	83.13	73.66	66.2	51.07	42.56	37.29	70.51	53.85	46.90	88.07	79.37	77.95	55.10	46.27	44.76	73.67	56.04	48.78	0.05	GTX 1080Ti
✓3D BN-Net [64] (2019)	83.56	74.64	66.76	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0.13	n/a
Fast Point RCNN [57] (2019)	85.29	77.40	70.24		-	-	-	-	-	90.87	87.84	80.52	-	-	-	-	-	-	0.065	Tesla P40
Patch Refinement [60] (2019)	88.67	77.20	71.82	-	-	-	-	-	-	92.72	88.39	83.19	-	-	-	-	-	-	0.15	1080 Ti
✓SARPNET [68] (2020)	84.92	75.64	67.70	49.89	41.97	40.93	77.66	60.43	54.03	88.93	87.26	78.68	56.81	45.07	43.86	79.94	62.80	55.86	0.05	GTX 1080Ti
✓FoF Net [69] (2020)	84.15	74.45	66.97	49.44	41.21	36.42	75.36	59.65	53.03	-	-	-	-	-	-	-	-	-	0.05	Titan XP
Part A^2 Net-Anchor [36] (2020)	87.81	78.49	73.51	52.17	43.29	40.29	79.17	63.52	56.93	91.70	87.79	84.61	59.04	49.81	45.92	83.43	68.73	61.85	0.08	Tesla V100
SegVoxelNet [70] (2020)	86.04	76.13	70.76	-	-	-	-	-	-	91.62	86.37	83.04	-	-	-	-	-	-	0.04	1080 Ti
HotSpotNet [53] (2020)	87.60	78.31	73.34	53.1	**45.37**	**41.47**	**82.59**	65.95	59.00	94.06	88.09	83.24	57.39	50.53	46.65	83.29	68.51	61.84	0.04	Titan V100
SA-SSD [72] (2020)	88.75	79.79	74.16	-	-	-	-	-	-	**95.03**	**91.03**	85.96	-	-	-	-	-	-	0.04	2080 Ti
CIA-SSD [73] (2020)	89.59	80.28	72.87	-	-	-	-	-	-	93.74	89.84	82.39	-	-	-	-	-	-	0.03	Titan XP
CenterNet3D [49] (2020)	86.20	77.90	73.03	-	-	-	-	-	-	91.80	88.46	83.62	-	-	-	-	-	-	0.04	n/a
Assosiate-3Ddet [74] (2020)	85.99	77.40	70.53	-	-	-	-	-	-	91.40	88.09	82.96	-	-	-	-	-	-	0.06	GTX 1080Ti
Voxel-RCNN [77] (2020)	**90.90**	81.62	77.06	-	-	-	-	-	-	94.85	88.83	86.13	-	-	-	-	-	-	0.04	2080 Ti
Pillar-based:																				
PointPillars [21] (2019)	82.58	74.31	68.99	51.45	41.92	38.39	77.10	58.65	51.92	90.07	86.56	82.81	57.60	48.64	45.78	79.90	62.73	55.58	**0.016**	1080 Ti
TANet [79] (2020)	84.39	75.94	66.82	53.72	44.34	40.49	75.70	59.44	53.53	91.58	86.54	81.19	**60.85**	**51.38**	**47.54**	79.16	63.77	56.21	0.035	Titan V
Voxel-FPN [50] (2020)	85.64	76.7	69.44	-	-	-	-	-	-	92.75	87.21	79.82	-	-	-	-	-	-	0.02	GTX 1080Ti
HVNet [16] (2020)	-	-	-	-	-	-	-	-	-	92.83	88.82	83.38	54.84	48.86	46.33	83.97	71.17	63.65	0.032	2080 Ti
Projection in BEV:																				
RT3D [62] (2018)	23.74	19.14	18.86	-	-	-	-	-	-	56.44	44.00	42.34	-	-	-	-	-	-	0.09	Titan X
BirdNet [25] (2018)	40.99	27.26	25.32	22.04	17.08	15.82	43.98	30.25	27.21	84.17	59.83	57.35	28.20	23.06	21.65	58.64	41.56	36.94	0.11	Titan XP
BirdNet+ [61] (2020)	70.14	51.85	50.03	37.99	31.46	29.46	67.38	47.72	42.89	84.80	63.33	61.23	45.53	38.28	35.37	72.45	52.15	46.57	0.1	Titan XP
Projection in FV:																				
LaserNet [23] (2019)	-	-	-	-	-	-	-	-	-	79.19	74.52	68.45	-	-	-	-	-	-	0.03	1080 Ti
Range RCNN [22] (2020)	88.47	81.33	77.09	-	-	-	-	-	-	92.15	88.40	85.74	-	-	-	-	-	-	0.06	Tesla V100
Point-based:																				
PointRCNN [40] (2019)	86.96	75.64	70.70	47.98	39.37	36.01	74.96	58.82	52.53	92.13	87.39	82.72	54.77	46.13	42.84	82.56	67.24	60.28	0.1	n/a
STD [37] (2019)	87.95	79.71	75.09	53.29	42.47	38.35	78.69	61.59	55.30	94.74	89.19	**86.42**	60.02	48.72	44.55	81.36	67.23	59.35	0.08	Titan V
Point-RGCN [41] (2019)	85.97	75.73	70.6	-	-	-	-	-	-	91.63	87.49	80.73	-	-	-	-	-	-	0.262	1080 Ti
✓StarNet [82] (2019)	81.63	73.99	67.07	48.58	41.25	39.66	73.14	58.29	52.58	-	-	-	-	-	-	-	-	-	-	-
3DSSD [43] (2020)	88.36	79.57	74.55	**54.64**	44.27	40.23	82.48	64.10	56.90	92.66	89.02	85.86	60.54	49.94	45.73	**85.04**	67.62	61.14	0.038	Titan V
✓3D IOU Net [39] (2020)	87.96	79.03	72.78	-	-	-	-	-	-	94.76	88.38	81.93	-	-	-	-	-	-	0.1	n/a
SE-RCNN [38] (2020)	87.74	78.96	74.3	-	-	-	-	-	-	94.11	88.10	83.43	-	-	-	-	-	-	0.1	Tesla P40
WS3D [42] (2020)	80.99	70.59	64.23	-	-	-	-	-	-	90.96	84.93	77.96	-	-	-	-	-	-	0.2	n/a
Graph-based:																				
Point GNN [26] (2020)	88.33	79.47	72.29	51.92	43.77	40.14	78.60	63.48	57.08	93.11	89.17	83.90	55.36	47.07	44.61	81.17	67.28	59.67	0.643	GTX 1070
Dual Representation:																				
PV-RCNN v1 [18] (2020)	90.25	81.43	76.82	52.17	43.29	40.29	78.60	63.71	57.65	94.98	90.65	86.14	59.86	50.57	46.74	82.49	68.89	62.41	0.08	n/a
PV-RCNN v2 [86] (2021)	90.14	**81.88**	**77.15**	-	-	-	82.22	67.33	60.04	92.66	88.74	85.97	-	-	-	84.60	71.86	63.84	0.062	n/a
SA-Det3D-DSA* [87] (2021)	88.25	81.46	76.96	46.97	40.89	38.80	82.19	**68.54**	**61.33**	92.42	90.13	85.93	52.03	45.82	43.81	83.93	**72.61**	**65.82**	0.08	n/a

SA-Det3D-DSA*: As implemented for a PV-RCNN v1 [18] backbone. ✓: The performance results were not found on the online KITTI [51] benchmark and instead the results from the original paper of the method are presented. For a method, that the results from its paper are not identical to the ones found on the online KITTI [51] benchmark, the results from KITTI are presented.

4.1.16. SA-SSD

SA-SSD [72], which stands for Structure Aware Single Stage Detector, is a one-stage detector that leverages a computational efficient and simplified voxelization representation. Every point is represented as a non-zero entry in the input tensor according to the point's 3D coordinates and quantization step $[d_x, d_y, d_z]$ of the input tensor. The quantization step determines the voxel size. If multiple points lie in the same tensor index, the last point is used.

A 3DBN with 3D sparse CNN following an encoder architecture as in [33] is used to extract high-dimensional features and construct the BEV image. A 2DBN following a VGG-inspired [47] architecture is used to further extract BEV features and an Anchor-based RPN applies two sibling 1x1 convolutions to generate a 3D BBox proposal map and a BEV K-part classification map. A Part-sensitive warping module splits the 3D BBox proposals into K regions, where each region represents an object part in a similar concept as in [53], and applies spatial transformation operations on the K dimension feature map to provide a more accurate classification confidence map.

A Detachable Auxiliary Network (DAN) is implemented alongside with the 3DBN and 2DBN, to assist the two backbone networks into learning more structure aware features through point-wise supervision. At each stage of the 3DBN and 2DBN, the corresponding features are projected back to the corresponding 3D point, through a feature propagation layer. Given as ground-truth the binary foreground/background point-wise segmentation labels and the relative position of each point from the object's center, a point-wise supervision is performed to learn more discriminative features. DAN is used only for training and is removed during inference.

4.1.17. CIA-SSD

CIA SSD [73], which stands for Confident IoU Aware Single Stage Detector, is a one-stage detector that applies voxelization on LiDAR point cloud, by encoding each voxel with the mean coordinates and mean reflection intensity of points within.

A 3DBN with 3D sparse CNN following an encoder architecture as in [33] is used to extract high-dimensional features and construct the BEV image. A 2DBN, named as Spatial-Semantic Feature Aggregation module, comprises of two different groups of 2D convolutions, one for spatial and one for semantic features, to encode and learn high-dimensional features in BEV while retaining a high spatial resolution. At the end of 2DBN, the two BEV feature maps are fused through an attentional module. An Anchor-based RPN is used to predict a classification score, 3D BBox regression and direction classification score maps along with an extra IoU confidence score map. To compensate the lack of the 3D BBox prediction refinement stage of two-stage detectors, the IoU confidence score map is used to rectify the classification confidence by amplifying the effect of high IoU confidence scores, against low ones. Finally, the 3D BBox proposals are filtered through a novel Distance-Variant IoU-Weighted NMS algorithm. The NMS algorithm aims in compensating the high uncertainties in the 3D BBox prediction, for objects located at a large distance from the LiDAR sensor.

4.1.18. CenterNet3D

Focused in computational efficiency and inspired by CenterNet [54], an object detector in 2D images, CenterNet3D [49] is a one-stage anchor-free and NMS-free detector, that predicts a single 3D BBox per object by modelling 3D objects as single center points in BEV.

First, the LiDAR point cloud is voxelized and the voxel feature is extracted through a VFE module as in [13].

A 3DBN with 3D sparse CNN following an encoder architecture as in [33] is used to extract high-dimensional features and construct the BEV image. A 2DBN composed of sequential layers of 2D convolution, 2D DefConv and 2D TrConv, is used to generate a high-dimension BEV feature map of larger spatial dimensions than the input one.

An Anchor-free network receives the high-dimensional BEV feature map as an input, and applies simultaneously three separate blocks of 2D Conv followed by 1x1 convolution, for producing three different 2D heatmaps, one for object center, one for 3D BBox regression and one for corner classification. Object centers are selected through a 3x3 kernel max pooling operation, from the resulting object center heatmap, thus one 3D BBox proposal is made per object without the need of NMS.

4.1.19. Associate-3Ddet

Associate-3Ddet [74] is a one-stage detector that focuses on learning robust features for partially visible or occluded objects. This is achieved through a domain adaptation-like strategy by constructing two siamese networks, the perceptual feature extraction (PFE) and the conceptual feature generator (CFG).

An augmented point cloud is constructed from the LiDAR point cloud by filling and completing the 3D objects. Then the CFG is end-to-end trained on those augmented scenes, separately from PFE. Once CFG training is complete, the weights are locked and the trained CFG model is used in parallel to help the PFE learn more robust features.

To train the PFE model, the LiDAR point cloud is introduced to the PFE, and its augmented corresponding point cloud is fed to the CFG. For both PFE and CFG, the point cloud is voxelized and a light-VFE as in [13] is used to extract the voxel feature. A 3DBN with 3D sparse CNN following an encoder architecture as in [33] is used to extract high-dimensional features and construct the BEV image. Next, two different 2DBNs are applied to PFE and CFG, resulting in two high dimensional BEV feature maps. For the PFE network, 2DBN comprises of 2D DefConv and 2D Conv layers. The perceptual to conceptual module takes as an input the two spatially equal and aligned BEV feature maps, from PFE and CFG, and learns how to make correspondences between the perceptual and conceptual features by optimizing the L2 distance between them for domain adaptation. An Anchor-based RPN receives the high-dimensional BEV feature map from PFE and outputs the confidence score and 3D BBox regression maps.

4.1.20. CenterPoint

CenterPoint [56] is a two-stage detector that detects 3D objects through their center points, inspired by similar work of CenterNet [54] object detector in 2D images. It is implemented for two different point cloud representations, namely for a voxel-based and a pillar-based. CenterPoint ranked first in the 2020 nuScenes 3D Detection Challenge [75].

In its voxel-based implementation, CenterPoint leverages previous work of VoxelNet [13] to structure the LiDAR point cloud, apply a 3DBN, then a 2DBN and construct a high-dimensional BEV feature map. The same applies for its pillar-based implementation where previous work of PointPillars [21] is used to construct the high-dimensional BEV feature map.

For both CenterPoint implementations of voxel and pillar based, the high-dimensional BEV feature map is fed to an Anchor-free network where three different BEV heatmaps are predicted for the object center, the 3D BBox regression and for the velocity and tracking of the object.

In its second stage, a point-based PR is utilized. First, four discriminative points are selected to encode the object's geometric structure. Each of the four points corresponds to the 3D center of each of the four faces of the predicted 3D BBox, projected in BEV. For each of these four points a feature vector is extracted from the high-dimensional BEV map from 2DBN through bilinear interpolation. These four feature vectors along with the feature vector of

the center point, are concatenated and fed through a computationally efficient point-based network consisting of MLPs, to output a classification score and refine the 3D BBox. In our paper, the performance of CenterPoint is reported for its voxel-based implementation only due to its superior performance compared to the pillar-based counterpart.

4.1.21. CVCNet

CVC-Net [14], which stands for Cross-View Consistency Network, is a one-stage detector that designed and introduced a new voxel representation, namely the HCS, to extract features from two different views through a shared 3DBN, in an effort to learn more discriminative features while saving computational resources.

The LiDAR point cloud is voxelized through the HCS voxelization and each voxel is encoded with the mean coordinates and mean reflection intensity of points within.

A 3DBN with 3D sparse CNN architecture as in [33] is used to extract high-dimensional features in the 3D HCS voxelized space. Next, the 3D HCS voxelized space is squeezed along the vertical and horizontal axis, resulting in two distinct multi-channel feature maps, a BEV and a FV respectively.

Two identical 2DBNs following a U-Net [45] architecture, are applied for each of the BEV and FV feature maps. The resulting high-dimensional BEV feature map is fed to an Anchor-free network as in [53] to output multi-head classification maps as in [65], along with 3D BBox regression maps. The resulting high-dimensional FV feature map is fed to a second Anchor-free network to output only a multi-head classification map.

The two resulting classification maps from BEV and FV are fed to a Cross-View Transformers (CVT) module, to align the features from both BEV and FV views. This alignment is performed by transforming the classification confidences from BEV to FV and vice versa through a voting scheme, inspired by Hough Transform.

4.1.22. Range GC-Net

Range-GC Net [76], which stands for Range-Guided Cylindrical Network, is a one-stage detector that also utilizes the HCS voxel representation. The range information is used to guide the 3DBN into learning more robust features, by adjusting the receptive field according to the object's scale and distance from sensor.

The LiDAR point cloud is voxelized according to HCS voxel representation. The HCS representation is split into two types, namely the guided and the guiding. For the guided HCS representation, the voxel feature is calculated with the mean coordinates and mean reflection intensity of points within, while for the second guiding HCS representation, the voxel feature is the range/distance of the voxel from the LiDAR sensor.

Two distinct 3DBNs with a 3D sparse CNN architecture as in [33], are applied on the guided and guiding HCS representations. At each stage of the 3DBNs, the output from guiding 3DBN is fed through a 1x1 convolution and then multiplied with the corresponding output from the guided 3DBN, thus forming a fused 3D feature map at each stage. The resulting fused 3D maps from multiple stages, are concatenated and fed through a 1x1 convolution layer to output the final 3D feature map.

The final 3D feature map is concatenated along Z axis to form a BEV feature map. A 2DBN as in VoxelNet [13] processes the BEV feature map. Next, an Anchor-free network as in CenterPoint [56] is utilized to process the high-dimensional BEV map from 2DBN, and to output three BEV heatmaps for the object center, the 3D BBox regression and for the velocity and tracking of the object.

4.1.23. Voxel-RCNN

Voxel-RCNN [77] is a two-stage detector that introduces a new computational efficient operation for grouping voxels, named as Voxel Query.

In its first stage, Voxel-RCNN utilizes previous work as in [18,33,72], to voxelize the LiDAR point cloud, apply a 3DBN, then a 2DBN and extract 3D BBox proposals through an Anchor-based RPN.

In its second stage, a hybrid point/voxel PR is utilized. A Region of Interest (RoI) of certain size is selected around the proposed 3D BBox and voxelized into a 6x6x6 grid. Each resulting voxel is encoded with its canonical coordinates along with the feature vector extracted from 3DBN. Then for each voxel, its center point is selected and categorized as a grid point. Given a grid point, Voxel Query is applied to group voxels around the grid point. Some of those grouped voxels may reside outside the proposed 3D BBox. Voxel Query is inspired by the Ball Query as in [27] but it utilizes the structured form of a voxelized 3D space, thus it is more computationally efficient. Next an accelerated PointNet [20] module, as implemented in the work of [78], is used to process the voxel features gathered by the Voxel Query and output the grid point feature vector. Finally, the grid point feature vectors are fed to a shared 2-layers MLP and then to two sibling FC layers to predict a confidence score and perform 3D BBox refinement.

4.2. Pillar-based

4.2.1. PointPillars

Driven by computational efficiency, PointPillars (PP) [21] is a one-stage detector and the first one that utilizes the concept of segmenting 3D space in pillars for 3D object detection in autonomous driving applications.

The LiDAR point cloud is segmented into pillars and a maximum number of points per pillar is selected. Each point inside a pillar is encoded by a 9-dimensional vector consisting of its original location $[x, y, z]$, its reflection intensity r, its offset distance from the pillar center $[x_p, y_p]$, and its distance from the arithmetic mean of all points within the pillar $[x_c, y_c, z_c]$. The 9-dimensional vector is fed through a simplified version of VFE [13], to extract the feature of the pillar. The result is a BEV feature map.

The BEV feature map is fed to a 2DBN as in [33], resulting in a high-dimensional BEV feature map. The resulting high-dimensional BEV feature map is then fed through an Anchor-based RPN inspired by SSD [5], to output classification score and 3D BBox regression maps.

4.2.2. TANet

TANet [79], which stands for Triple Attention Network, is a one-stage detector that utilizes a triple attention module to extract a voxel/pillar feature, more robust to noisy data. Furthermore it employs a coarse to fine regression concept in its 2DBN, to increase the localization accuracy without adding a significant computational cost.

The LiDAR point cloud is segmented into pillars and two sequential Triple Attention (TA) modules, composed of a point-wise, channel-wise and voxel-wise attention groups, are used to extract the pillar feature. The Excitation operation [80] is used to compute the point-wise and channel-wise feature vector and fully connected layers are used for computing the voxel-wise feature. All these three vectors are fused together to form the TA feature vector.

The resulting BEV feature map is fed to a 2DBN consisting of two networks, a Coarse Regression (CR) and a Fine Regression (FR) one. The CR network is of similar architecture as the 2DBN of [33]. The resulting high-dimensional BEV map from CR is fed to an Anchor-based RPN for a coarse BBox proposal. The Fine Regression (FR) network, follows an FPN [44] Multiple Input Multiple Output (MIMO) architecture to provide cross-layer feature maps of rich features. The cross-layer feature maps are upsampled to a fixed size and concatenated to construct the Fine Regression (FR) BEV

feature map. The resulting FR BEV feature map is fed to an Anchor-based RPN where the coarse BBox proposals from CR branch are used as the new anchors. This second Anchor-based RPN outputs the final prediction for TANet, consisting of a classification score and a 3D BBox proposal.

4.2.3. Voxel-FPN

Voxel-FPN [50], which stands for Voxel Feature Pyramid Network, is a one-stage detector that proposes to extract a per voxel/pillar feature in multiple scales, and then fuse the multi-scale features through a FPN [44] architecture in its 2DBN. The objective behind this strategy is to discard the need to manually fine tune the pillar dimensions and furthermore to capture more robust multi-scale features.

The LiDAR point cloud is segmented into pillars according to three different pillar scales. Then a VFE module as in [13] is used to extract pillar features for each scale, resulting in three multi-scale BEV feature maps. A 2DBN module based on an FPN [44] MIMO architecture as shown in Fig. 6, receives the multi-scale feature maps, processes them and outputs three high-dimensional BEV maps at different scales. An Anchor-based RPN receives each of the three BEV maps and outputs for each input BEV map a prediction score and a 3D BBox regression map.

4.2.4. HVNet

Inspired by the same objective as in [50], HVNet [16], which stands for Hybrid Voxel Network, is a one-stage detector that learns multi-scale voxel/pillar features by introducing the hybrid scale voxelization. The advantage of the method is that all points are retained during hybrid scale voxelization and that the multi-scale voxel features are projected back to points for memory and computational efficiency.

The LiDAR point cloud is segmented into multiple pillar scales through hybrid scale voxelization. A Hybrid Voxel Feature Extractor module is used to compute the pillar feature at each scale, and construct multi-scale BEV feature maps. Next, a 2DBN module following an FPN [44] MIMO architecture is used to construct three high-dimensional BEV feature maps for the 'car', 'pedestrian' and 'cyclist' classes. The three high-dimensional BEV feature maps are fed to an Anchor-based RPN inspired by [5], resulting in a classification score, location corner offset and height regression maps. The location corner offset map encodes the location of the 8 corners of the 3D BBox and the height repression map encodes the Z center coordinate and height of the 3D BBox.

4.2.5. AFDet

AFDet [55], which stands for Anchor Free Detector, is a one-stage anchor-free detector that utilizes a pillar-based point cloud representation. It aims in constructing a 3D object detector friendly to embedded systems by removing the computational CPU expensive post processing step of NMS and the excessive convolution operations of Anchor-based RPNs.

Leveraging the previous work of PointPillars [21], the LiDAR point cloud is segmented into pillars, a feature vector is extracted per pillar and a BEV feature map is constructed.

The BEV feature map is fed through a 2DBN, following an FPN [44] SISO architecture, to construct a high dimensional BEV feature map of the same spatial dimensions as the input one. The resulting high-dimensional BEV feature map is fed to an Anchor-free network where five individual maps are predicted, namely for center object localization in BEV, for center offset regression in BEV, for Z-axis location regression, for 3D BBox size regression and for orientation prediction. For each object only one center is selected through max pooling on the corresponding heatmap. Therefore, only one 3D BBox is predicted per object, thus NMS is no longer needed.

4.3. Projection in BEV

4.3.1. RT3D

RT3D [62], which stands for Real Time 3D, is a two-stage detector that utilizes a point cloud projection in BEV plane and follows an architecture similar of a 2D object detector [4], motivated by its computational efficiency.

The LiDAR point cloud is projected in a BEV image of certain x-y grid resolution. Each grid is encoded by three features, namely the minimum, mean and maximum height value of points located within. Next, a 2DBN following a ResNet-50 [46] architecture, is used to construct a high-dimensional BEV feature map. An Anchor-based RPN, as in Faster-RCNN [4], receives the high-dimensional BEV feature map as an input and outputs RoIs.

In its second stage, a 2D PR inspired by Faster-RCNN [4], is used to learn pose-sensitive feature maps by splitting every object into k^2 parts in the BEV. Based on those learned pose-sensitive maps and the given RoI from first stage, a confidence score prediction and 3D BBox refinement is made.

4.3.2. BirdNet

BirdNet [25] is a two-stage detector that utilizes a point cloud projection in BEV plane and follows an architecture similar of a 2D object detector [4].

The LiDAR point cloud is projected in a BEV image of certain x-y grid resolution. Each grid is encoded by three features, namely the maximum height, mean reflection intensity and a normalized point density value of points within. Next, a 2DBN following a VGG-16 [47] architecture, is used to construct a high-dimensional BEV feature map. An Anchor-based RPN, as in Faster-RCNN [4], receives the resulting BEV feature map from 2DBN, and outputs a RoI in BEV. In its second stage, the RoI is classified as in [4] and a 2D BBox in BEV is predicted.

To convert the 2D BBox into a 3D BBox, a post processing handcrafted approach is used to extract a Z value by estimating a coarse ground plane and extracting the maximum height through the corresponding channel of the BEV image.

4.3.3. BirdNet+

BirdNet+ [61] is a two-stage detector and an improved version of BirdNet [25].

Similar to Birdnet [25], the LiDAR point cloud is projected in a BEV image of certain x-y grid resolution and each grid is encoded by three features, namely the maximum height, mean reflection intensity and a normalized point density value of points within. Next, a 2DBN following a ResNet-50 [46] architecture, is used to construct a high-dimensional BEV feature map. An Anchor-based RPN, as in Faster-RCNN [4], receives the BEV feature map and outputs a predicted 2D BBox in BEV.

In its second stage, the predicted 2D BBox is fed as an input to two fully connected layers to perform classification and 3D BBox regression.

4.4. Projection in a perspective front-view range image

4.4.1. LaserNet

LaserNet [23] is a one-stage detector that utilizes a point cloud projection in a Front View (FV) range image, to benefit from the computational efficient and compact representation of a 2D grid.

The LiDAR point cloud is projected to a perspective FV range image and each range image pixel is encoded by a 5-dimensional feature vector $[d, z, \theta, r, b]$, where d is distance from sensor, z is the height distance from sensor, θ is the azimuth angle, r is the reflection intensity and b is a binary value that determines if the range image pixel contains a point or not.

The constructed FV range image is fed to 2DBN of similar architecture as a deep aggregation network [81]. The output high-dimensional FV range image from 2DBN is fed to an Anchor-free proposal network where 1x1 convolution layers are applied to predict a per pixel class probability score, along with a probability distribution over 2D BBoxes in BEV.

Given the predicted class probabilities and 2D BBox probability distributions from the Anchor-free proposal network, two post processing steps are applied to remove redundant proposals. First, the per pixel class probabilities are grouped through a Mean Shift clustering algorithm and then an Adaptive NMS operation utilizes the predicted 2D BBox probability distributions, to remove redundant 2D BBoxes. The predicted 2D BBox in BEV is converted to a 3D BBox assuming a certain ground plane and a fixed object height.

4.4.2. Range-RCNN

Range-RCNN [22] is a two-stage detector that utilizes a point cloud projection in a Front View (FV) range image.

The LiDAR point cloud is projected to a perspective FV range image and each pixel is encoded by a 5-dimensional vector $[x, y, z, d, r]$ where $[x, y, z]$ are the point coordinates, r is the point's reflection intensity and d is the point's distance from the LiDAR sensor.

A 2DBN following an FPN [44] SISO architecture with 2D DilConv as shown in Fig. 6, is used to learn per pixel features. However, to address the problem of occlusions and scale variation of objects in FV range images, a RV-PV-BEV module is implemented for transferring features from FV range image pixels to 3D points and then from 3D points to a BEV feature map. The BEV feature map is processed through a 2DBN and Anchor-based RPN as in [33] to generate classification score and 3D BBox regression maps.

In its second stage, Range RCNN utilizes a 3D voxel-based PR similar to [36] and voxelize the 3D BBox proposal from first stage. However unlike the implementation of [36], 3D convolutions are not applied. Instead the resulting voxelized 3D BBox is flattened to one dimension and fed through FC layers to refine the confidence score and proposed 3D BBox.

4.5. Point-based

4.5.1. Point-RCNN

Point-RCNN [40] is a two-stage detector that retains the natural and unstructured form of a point cloud both in its first and second stage, to avoid the information loss caused during quantization of the LiDAR point cloud.

The LiDAR point cloud is sub-sampled to a fixed size to obtain a more compact representation. A 3DBN Pointnet++ [27] encoder/decoder architecture is used to extract high-dimensional point-wise features and perform foreground/background binary point segmentation. Given the high-dimensional point features and binary segmentation information, an Anchor-free network composed of FC layers, utilizes a bin-based box regression approach, to predict 3D RoIs. NMS is applied to remove redundant RoI and keep only those of high quality.

To further encode the local structural information of an object, a 3D-point based PR is used by Point-RCNN in its second stage. Given the RoI from first stage, the proposed 3D BBox is enlarged to increase its receptive field and a fixed number of points within the enlarged 3D BBox are randomly selected. Each point is encoded with their canonical coordinates, reflection intensity r, distance from sensor d, and predicted binary segmentation information/mask m from stage-1. FC layers are used to convert those local features into the same dimensions as the stage-1 high-dimensional features. Once in the same dimensions, local features and stage-1 high-dimensional features are concatenated and fed to a PointNet++ [27] encoder architecture for confidence prediction and 3D BBox refinement.

4.5.2. STD

STD [37], which stands for Sparse To Dense, is a two-stage detector that uses the natural, unstructured form of a point cloud in its first stage, while switching to a voxel-based representation of the point cloud for its second stage of proposal refinement, to exploit the efficiency of the VFE module.

The LiDAR point cloud is sub-sampled to a fixed size to obtain a more compact representation. A 3DBN Pointnet++ [27] encoder/decoder architecture is used to extract high-dimensional point-wise features and perform foreground/background binary point segmentation. Given the high-dimensional point features and binary segmentation information, an Anchor-based network, inspired by PointNet [20], utilizes a fixed size spherical anchor to predict 3D RoIs. NMS is applied to remove redundant RoIs and keep only those of high quality.

A 3D-voxel based PR is used by STD in its second stage. The RoI from first stage is voxelized and points are assigned to each voxel as in VoxelNet [13]. Each point is encoded with their canonical coordinates and the extracted high-dimensional features from stage-1. Next, a VFE module as in [13] is applied to extract the voxel feature. The voxelized 3D ROI is flattened to one-dimension and fed to two parallel branches of FC layers. The first branch is used for confidence prediction and 3D BBox refinement, while the second branch is used for IoU estimation for predicting the 3D IoU between ground truth boxes and predicted ones. The IoU estimation is multiplied with the confidence prediction and fed as a criterion to an IoU-based NMS for removing redundant proposals.

4.5.3. Point-RGCN

Point-RGCN [41] is a two stage detector that uses the natural, unstructured form of a point cloud in its first stage, while switching to a graph-based representation of the point cloud for its second stage of proposal refinement, inspired by the advances in graph convolution networks [58].

For its first stage from point cloud representation till region proposal, Point-RGCN leverages the first stage of PointRCNN [40] as described in Section 4.5.1.

Point-RGCN for its second stage utilizes a 3D graph-based PR consisting of two modules, the Residual Graph Convolutional Network (R-GCN) and the Contextual Graph Convolutional Network (C-GCN). R-GCN is used for classifying and refining the 3D ROI based on their local characteristics, while C-GCN is used to further refine the 3D ROI by sharing contextual information between multiple 3D ROI.

Starting with R-GCN, the proposed 3D BBox is expanded and a fixed number of points is selected. Each point is encoded with a feature vector, learned through fusion of its canonical coordinates with its feature vector from stage-1. Next, a graph is constructed and MRGCN [58] layers are applied on the points. The C-GCN gathers for each proposal the output features of the R-GCN module and constructs a new graph where each proposal/object is considered a node. The new graph is processed through EdgeConv [59] layers to learn a global feature for all proposals. That global feature is concatenated with the local features on each proposal and then fed to two FC layers for refining the class confidence and predicted 3D BBox.

4.5.4. StarNet

StarNet [82] is a one-stage detector that uses the natural, unstructured form of a point cloud and proposes a sparse center sampling strategy to save computational resources.

At first, the point cloud is cropped on its Z axis $[-1.35, inf]$ so that it does not contain ground points. Next, a fixed number

of points C is selected from the cropped point cloud based on Furthest Point Sampling (FPS) algorithm. These C points are considered as center proposals. For each center proposal, K points are selected from raw point cloud at a radius R around the center point, thus forming a cropped point cloud around a center proposal which is considered as a Region of Interest (RoI). Every cropped point cloud/RoI is processed independently. Given a RoI, each point is encoded by a 4-dimensional vector of its location $[x, y, z]$ and its reflection intensity r. Then, FC layers are used to transform the 4-dimensional vector into a 64-dimensional one.

For feature extraction, StarNet does not leverage any previous work but instead proposes its own StarNet point cloud featurizer. The StarNet featurizer consists of many sequential StarNet blocks. Given K points within a RoI as an input, each StarNet block initially performs a feature max operation, feature concatenation and then applies two blocks of Batch Normalization [34], linear projection and Relu activation [35], to output a 64-dimension feature vector. The output of each StarNet block is averaged and then the outputs from all StarNet blocks are concatenated to form a 384-dimensional vector.

An Anchor-based RPN receives the 384-dimensional vector and places anchors at different grid offsets from the center proposal point. For each grid, a D dimensional feature vector is computed through a learned linear projection and used for prediction score and 3D BBox regression. Finally, NMS is applied to remove redundant proposals.

4.5.5. 3DSSD

3DSSD [43], which stands for 3D Single Stage Detector, is a one-stage detector that uses the natural, unstructured form of a point cloud. The architecture of 3DSSD [43] is designed to speed up the inference time of a point-based detector by discarding the upsampling layers of a semantic PointNet++ [27] in the 3DBN module, along with the removal of the second stage of 3D BBox refinement. This is achieved through a newly introduced FS strategy and a specially designed 3DBN following a PointNet++ [27] encoder architecture as shown in Fig. 5.

The LiDAR point cloud is sub-sampled to a fixed size through random sampling, to obtain a more compact representation. A 3DBN Pointnet++ [27] encoder architecture is used to extract a subset of points, containing high-dimensional features, through FS strategy as described in Section 3.2.1. As a result, a subset of points selected from Distance-FPS and Feature-FPS are received as an output from 3DBN.

An Anchor-free network receives the subset of points as an input, however only points selected through Feature-FPS are treated as initial center points. Those center points are shifted towards their corresponding instance center with extra supervision similar to VoteNet [83]. For each of the shifted center points, a 3D BBox prediction is made for its corresponding instance.

4.5.6. 3D IOU-Net

3D IoU-Net [39], which stands for 3D Intersection over Union Network, is a two-stage detector that uses the natural, unstructured form of a point cloud.

For its first stage from point cloud representation till region proposal, 3D IOU-Net leverages the first stage of PointRCNN [40] as described in Section 4.5.1.

In its second stage, 3D IOU-Net aims in improving its predicted IoU. This is achieved by learning a perspective-invariant per object feature representation and by encoding the geometric information of the predicted 3D BBox through two distinct modules, namely the Attentive Corner Aggregation (ACA) and Corner Geometry Encoding (CGE). The ACA module attempts to become aware of the full view of the object by learning a local feature per 3D BBox proposal from the eight corner perspectives through PointNet++ [27] and PointNet [20] architectures. Each corner perspective is weighted through a Perspective-channel attention mechanism inspired by TANet [79]. This attention mechanism is implemented as part of the ACA module. The CGE module feeds the absolute corner coordinates of the proposed 3D BBox through an MLP and 1D convolution, to extract a geometric feature from the corners of the proposed 3D BBox. Features from ACA and CGE are concatenated through MLPs to learn the IoU sensitive proposal feature.

An IoU alignment operation inspired by [84], is used to remove redundant results and to re-feed the output proposals again to the ACA and CGE modules for a second inference. Finally an IoU-guided NMS is used for post processing the resulting 3D BBox proposals.

4.5.7. SE-RCNN

SE-RCNN [38], which stands for Spatial Embeddings RCNN, is a two-stage detector that uses the natural, unstructured form of a point cloud and performs joint 3D BBox prediction along with instance segmentation. The intuition is that instance segmentation will encode more accurately the geometric information of an object, especially in cases of occlusions or when points from other objects or background are located within the object's bounding box.

The LiDAR point cloud is subsampled to a fixed size through random sampling. A 3DBN PointNet++ encoder/decoder architecture is used for learning per point high-dimensional features, foreground/background binary semantic information and instance-aware spatial embeddings (SE).

The binary semantic information and SE are fed to a deep clustering layer. Utilizing simple clustering algorithms (i.e. K-means) instance segmentation is performed and each point is assigned to an instance id. The point-wise instance ids along with the point-wise high-dimensional features from 3DBN, are used by an Anchor-free network to generate only one 3D BBox proposal per object, thus NMS is no longer needed.

In the second stage, a 3D point-based PR is employed. Given a 3D BBox proposal with an instance id, all points that are classified with the same instance id are selected, even if some of the points are located outside the proposed 3D BBox. The same way, points that are located in the proposed 3D BBox but do not have the same instance id, as the proposed 3D BBox, are removed. Out of all selected points, a fixed number of points is chosen through random sampling. These points are encoded with their canonical coordinates and high-dimensional features from first stage, and introduced to a PointNet++ [27] for refining the 3D BBox.

4.5.8. WS3D

Following a different approach and motivated by the increased annotation cost in work hours for labeling LiDAR data, two-stage detector WS3D [42], which stands for Weakly Supervised 3D, focuses on learning from weakly supervised labelled BEV maps along with few well labelled object instances. Weakly supervised labels refer to labelling just the center of an object in BEV with a simple click without any further 3D BBox labels.

During its first stage, WS3D learns to predict a cylindrical proposal and during its second stage, the cylindrical proposal is used to predict a corresponding 3D BBox. The weakly supervised scenes are used to train the first stage and the well labelled object instances are used to train the second stage of 3D BBox proposal. As a pre-processing step, a Pseudo Groundtruth Generation (PGG) module is used to convert the weakly center-based BEV annotation into a per point pseudo foreground label through a simple algorithm according to each point's distance from the object's center.

The first stage of WS3D follows a similar architecture to PointRCNN [40] where the LiDAR point cloud is subsampled to a fixed size and introduced to a 3DBN PointNet++ encoder/decoder architecture for extracting point-wise high-dimensional features. The pseudo foreground labels from PGG are used to supervise the per point foreground/background binary segmentation. An Anchor-free bin-based classification as in [40] is used for predicting the object center and a fixed size radius (i.e. of 4m for the 'car' class) is used to generate the cylindrical proposal that extends unlimited in the Z axis.

A Center-Aware (CA) NMS is used to remove redundant proposals. The score of foreground points within the 4m radius cylindrical proposal, is used by the CA NMS as the confidence score and criterion.

The second stage of WS3D follows a 3D point-based PR architecture. Given the cylindrical proposal from first stage, a fixed number of points is sampled, encoded into a 5D vector of canonical coordinates $[x, y, z]$, reflection intensity and binary foreground score, and introduced to a PointNet++ [27] architecture for estimating the initial 3D BBox coordinates. Once the initial 3D BBox is predicted, a second refinement step is applied. Points within the initial 3D BBox are introduced again through a similar PointNet++ [27] architecture to further refine the initial 3D BBox, resulting in a second and final 3D BBox proposal.

4.6. Graph-based

4.6.1. Point-GNN

Inspired by the advances in graph convolutional networks and the research potential in using graph neural networks for 3D object detection in LiDAR point clouds, Point-GNN [26], which is a one-stage detector, utilizes a graph-based approach from start to end.

The LiDAR point cloud is down-sampled to a fixed size through voxelization and then a graph is constructed with voxels used as vertices. Each vertex is then connected to its neighbors within a fixed radius. The initial feature of each vertex is calculated through MLPs in a light version of PointNet [20] as in PointPillars [21].

A 3DBN following a graph neural network [28] (GNN) architecture, consisting of MLPs, is used for high-dimension feature extraction. Unlike common GNNs, Point-GNN is redesigned to encode spatial information along with learned high-dimensional features. The GNN is executed for a fixed number of iterations. For each iteration the MLPs are different, therefore, weights are not shared between iterations. At the end of GNN, in an Anchor-free approach, two different MLPs are used, one for classification and one for per class 3D BBox regression. Although no predefined anchors are used, multiple 3D BBox proposal are made per object since multiple vertices may correspond to a single object. A variation of standard NMS with Box Merging and Scoring is used to remove redundant proposals and improve the localization accuracy.

4.7. Dual point cloud representations

4.7.1. MVF

MVF [15], which stands for Multi View Fusion, is a one stage detector that utilizes a dual representation of the point cloud, of voxelization and of a projection in a front view (FV) range image, to learn view-dependent features. Furthermore, it introduces the dynamic voxelization of the LiDAR point cloud to diminish the drawbacks of fixed voxelization, namely point and voxel drop out.

Each point of the LiDAR point cloud is converted to a spherical coordinate representation (ϕ_i, θ_i, d_i). Then, each point is assigned to a 3D voxel according to its 3D coordinates and also to a frustum grid according to its spherical coordinates. The local coordinates from both representations are concatenated and through FC layers, they are transformed into an N-dimensional vector, called as feature-1.

The dynamic voxelization encoding is utilized to establish a bi-directional mapping between points and voxels/grids for a voxel and a FV range image representation respectively. Each point, for both representations, is encoded with feature-1. The voxelized 3D space is considered as a multi-channel BEV image and a 2DBN module is used to extract per voxel high dimensional features, called as feature-2. For the FV range image representation, a second 2DBN is used to extract per grid high dimensional features, called as feature-3. Both 2DBNs are of the same FPN [44] SISO architecture and use ResNet [46] blocks for downsampling and 2D TrConv layers for upsampling. The 2DBNs are two separate networks and therefore have different weights.

The extracted per voxels feature-2 and feature-3 are projected back to the corresponding points. A concatenation is made for feature-1, feature-2 and feature-3, thus forming the per point feature vector, called as feature-4. The rest of the network architecture is as described in PointPillars [21] in Section 4.2.1 starting from pillar feature extraction. Instead of encoding each point with a learned n-dimensional vector during pillar feature extraction as in [21], in MVF the point is encoded with the learned feature-4 vector.

4.7.2. Pillar based object detection

Inspired by MVF [15], Pillar based Object Detection (OD) [85] is a one-stage detector that utilizes a dual representation of the point cloud, namely pillar-based and projection in a FV range image. There are three main differences with [15], with the first one being the projection of the FV range image to a cylindrical coordinate frame instead of a spherical one, to remove the perspective distortions effect. The second one concerns the projection of the view-dependent learned features back to points through bilinear interpolation, to mitigate the spatial aliasing effect. The third difference is the replacement of the Anchor-based RPN with a more simple and computational efficient Anchor-Free network.

The representation of LiDAR point cloud, the per view point extraction and feature fusion follow the same architecture as described in MVF [15] in Section 4.7.1, where points are encoded with a multi-view extracted feature.

Then, points encoded with the multi-view extracted feature vector, are projected in pillars. A light-PointNet [20] network as in PointPillars [21], is used to extract a feature vector for each pillar and construct the BEV feature map. A 2DBN as in [21] receives the BEV feature map and further process it into a high-dimensional BEV feature map. An Anchor-free network receives the high-dimensional BEV map as an input and outputs confidence score and 3D BBox regression maps. Finally NMS is applied to remove redundant proposals.

4.7.3. PV-RCNN V1

PV-RCNN v1 [18], which stands for PointVoxel-RCNN, is a two-stage detector that utilizes a voxel representation along with the natural, unstructured form of a point cloud. This dual representation aims to benefit from the computational efficiency of convolutions and the increased receptive field of the set abstraction operation of PointNet++ [27]. The voxel representation is used in the first stage to propose a 3D BBox, while the point cloud representation is used to learn and aggregate multi-scale per point features, for the second stage of 3D BBox refinement.

For its first stage, PV-RCNN v1 utilizes previous work of SECOND [33] to voxelize the LiDAR point cloud, perform feature extraction through a 3DBN sparse CNN and 2DBN, and utilize an Anchor-based RPN to propose a 3D BBox as described in Section 4.1.6.

For the second stage of proposal refinement, a fixed number of points, from now on called keypoints, are sampled from the LiDAR point cloud through the FPS algorithm. A Voxel Set Abstraction (VSA) module utilizes a PointNet-based [20] network to aggregate on those keypoints, the learned multi scale voxel features from the 3DBN sparse CNN. Along with the multi scale voxel features, two extra keypoint-wise feature vectors are also aggregated. The first feature vector is extracted by feeding the LiDAR point cloud keypoints through a PointNet [20] network and the second feature vector is extracted from the 2DBN high-dimensional BEV feature map through bilinear interpolation. Since keypoints might be either foreground or background points a Predicted Keypoint Weighting (PKW) module is implemented to re-weight the learned VSA features in favour of the foreground points through extra supervision from point cloud foreground/background segmentation labels.

The proposed 3D BBox from the Anchor-based RPN is voxelized into a 6x6x6 voxel grid and 216 points are uniformly sampled. Then, a RoI-grid Pooling module utilizes a PointNet-based [20] set abstraction operation to transfer the learned and re-weighted features of the keypoints, to the sampled 216 grid-points. Finally, MLP layers are used to predict a confidence score and perform 3D BBox refinement.

4.7.4. PV-RCNN V2

PV-RCNN v2 [86] is a two-stage detector and an extension/improved version of PV-RCNN v1 [18], focused in computational efficiency. Therefore it has the same structure as in [18]. There are however two main differences. The first difference concerns the selection of keypoints in the second stage, by introducing the Sectorized Proposal Centric (SPC) keypoint sampling algorithm.

Given the proposed 3D BBoxes from first stage and the LiDAR point cloud, SPC utilizes the FPS algorithm to sample a fixed number of points that reside within a specified distance around the centers of the proposed 3D BBoxes. Compared to applying the FPS algorithm on the whole LiDAR point cloud as in [18], SPC samples more representative keypoints close to the proposed 3D BBoxes and not far away in the background. Furthermore, SPC is more computationally efficient compared to FPS.

The second difference concerns the replacement of the PointNet-based set abstraction operation, used in the VSA and RoI grid Pooling modules of [18] with the newly introduced and less computationally expensive VectorPool module. VectorPool is used to construct a feature vector for each keypoint, by aggregating features from local neighborhoods. Each local feature is encoded through separate kernel weights instead of MLPs as in the set abstraction operation.

4.7.5. SA-Det3D

SA-Det3D [87], which stands for Self Attention Detector 3D, is a self attention-based framework for 3D object detection in LiDAR point clouds. It aims in overcoming certain natural limitations of a LiDAR point cloud, namely the sparsity and uneven distribution of points, along with noisy and incomplete data. This is accomplished by employing a self-attention transformer network architecture [88] to learn more robust and discriminative global features and combine them with ones learned from convolution or point-based networks.

SA-Det3D [87] is implemented for SECOND [33], PointPillars [21], Point-RCNN [40] and PV-RCNN v1 [18] 3D object detectors. It is presented in the section of Dual representation since the performance of SA-Det3D is reported in KITTI[51] 3D and BEV Benchmarks, for the PV-RCNN v1 [18] implemented version.

SA-Det3D proposes two different self attention modules, the Full Self-Attention (FSA) and the Deformable Self Attention (DSA). The objective for both FSA and DSA modules is to exploit pairwise similarities among local feature vectors through a self-attention mechanism as in [88], and extract a context feature vector. Once calculated, the context feature vector is concatenated with the local feature, as it is extracted from standard 3DBN and 2DBN modules, thus forming the per point/voxel/pillar final feature vector. Only one module, either FSA or DSA is applied in the architecture of a 3D object detector.

An FSA or DSA module is applied after the per point/voxel/pillar (local) feature extraction is performed, i.e. for PointPillars [21] it is applied after the light-PointNet network, used to extract the per pillar feature vector. The main difference between DSA and FSA is that DSA is applied for a subset of points/voxels/pillars selected through FPS algorithm. The resulting global context is then up-sampled to the rest of the points/voxels/pillars through feature propagation (FP) layers as in PointNet++ [27].

5. Datasets and evaluation metrics

Publicly available datasets are employed for training the LiDAR-based 3D object detectors through detailed labels in the form of 3D BBoxes. These datasets are, KITTI [51] and the most recent KITTI-360 [93] datasets from Karlsruhe Institute of Technology, nuScenes [66] dataset from nuTonomy, Waymo Open Dataset [12] from Waymo, A2D2 [92] dataset from Audi, ApolloScape [89] dataset from Baidu Inc, H3D [91] dataset from Honda Research Institute and the Lyft [90] dataset from Lyft Inc. Albeit different in certain attributes, such as the number of annotated objects and object classes among others, all the aforementioned datasets contain LiDAR sensor data along with annotated 3D BBoxes. Three open datasets with online 3D and BEV benchmarks are presented below, along with their evaluation metrics.

5.1. KITTI Dataset

KITTI [51] dataset is one of the oldest available datasets for autonomous driving applications, created by Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago. It is recorded during day-time, on highways and rural areas of Karlsruhe city in Germany. It contains images from two facing forward high resolution stereo camera systems, one color and one grayscale, a 360° point cloud captured from a Velodyne HDL-64E LiDAR sensor mounted at the top of the car, as well as localization data. KITTI provides synchronized data, of images with LiDAR point clouds, at a rate of 10 frames per second (fps).

A point is encoded by a four dimensional vector $[x, y, z, r]$, of its 3D coordinates from the LiDAR sensor and its reflection intensity value.

For the 3D and BEV object detection tasks, a fixed train and test set are provided. Objects are labelled according to eight different classes, however only three of these, ‘car’, ‘pedestrian’ and ‘cyclist’, are used for evaluation. Furthermore, only the objects that are within the cameras’ field of view, and are therefore visible in the images, are used for the official evaluation. Each object is labelled in the form of a 3D BBox and encoded by its center coordinates $[x, y, z]$, its length, width, height $[l, w, h]$ in meters, its heading and observation angles (α, θ) in radians, its occlusion and truncation state and finally its corresponding class. An object is assigned to an easy, moderate or hard category, depending on its corresponding size, in pixels, in the 2D image in combination to its truncation state.

KITTI [51] uses as a metric the mean Average Precision (mAP) with a threshold on Intersection over Union (IoU) for the 3D and BEV object detection tasks. The IoU threshold for the class ‘car’ is set to 0.7 and for ‘pedestrian’ and ‘cyclist’ to 0.5. For the 3D object detection task the 3D IoU between the predicted and ground truth 3D BBoxes is used, while for the BEV detection task the 2D IoU

between the predicted and ground truth BEV BBoxes is used. If the IoU is above the threshold values, the prediction is considered as true positive (TP), otherwise as false positive (FP).

Precision (p) is defined as the fraction of all TP (N_{TP}) predictions to all predictions, both TP and FP, ($N_{TotalDetections}$) as:

$$p = \frac{N_{TP}}{N_{TotalDetections}} \tag{9}$$

and Recall (r) is defined as the fraction of all TP (N_{TP}) predictions to all ground truths ($N_{GroundTruth}$) as:

$$r = \frac{N_{TP}}{N_{GroundTruth}} \tag{10}$$

The Precision-Recall (P-R) curve is constructed by plotting the precision and recall (on y and x axis respectively) for different IoU threshold values.

Average Precision is the area formed by the P-R curve. KITTI used the mAP based on an 11 recall point interpolation P-R curve while after the third quarter of 2019 [94], a 40 recall point interpolation is used instead, to better approximate the P-R curve.

Mean Average Precision (mAP) is calculated by averaging the Average Precision (AP) on all classes. However that metric can not define the orientation similarity, i.e. the front and back part of a 3D BBox. Therefore KITTI also introduces the Average Orientation Similarity (AOS) defined as:

$$AOS = \frac{1}{N}\sum_{r\in K}\max_{\tilde{r}:\tilde{r}\geq r} s(\tilde{r}) \tag{11}$$

where N is the number of recall points, (was 11 and after the third quarter of 2019 is 40) and K is a recall subset (was $[\frac{0}{11}, \frac{1}{11}, \frac{2}{11}, \ldots, \frac{11}{11}]$ and after the third quarter of 2019 is $[\frac{0}{40}, \frac{1}{40}, \frac{2}{40}, \ldots, \frac{40}{40}]$)

The orientation similarity $s \in [0, 1]$ is a normalized cosine similarity defined as:

$$s(r) = \frac{1}{|D(r)|}\sum_{i\in D(r)}\frac{1+\cos\Delta_\theta^{(i)}}{2}\delta_i \tag{12}$$

where $D(r)$ is the set of all object detections at recall rate r, $\Delta_\theta^{(i)}$ is the angle difference between the predicted and ground truth orientation of i^{th} detection and δ_i is set to 1 if detection i is a TP, otherwise δ_i is set to 0.

5.2. NuScenes dataset

NuScenes [66] dataset, created by nuTonomy, is a newly introduced dataset for autonomous driving applications captured in four different areas, of Boston and Singapore, under various lighting and weather conditions. It provides LiDAR point cloud data from a 32 beam LiDAR sensor, radar data from 5 different radar sensors along with camera data from six different RGB cameras. Sensors are placed accordingly on the car so that a 360° field of view is covered around the car. Furthermore LiDAR and camera sensors are well synchronized to provide data alignment between LiDAR data, radar data and RGB images. Labelled data are provided at a rate of 2Hz, however intermediate sensor frames are also released at a rate of up to 20Hz.

NuScenes annotates a total of 23 different object classes, however there is a severe class imbalance among them. Each object is annotated with its class, attributes (visibility, activity, pose) and a 3D BBox encoded by its center coordinates $[x, y, z]$, its length, width, height $[l, w, h]$ and its yaw angle θ.

For the 3D object detection task, a model is evaluated in a total number of 10 classes out of the 23. A scalar score is introduced, namely the nuScenes Detection Score (NDS), given by the following equation:

$$NDS = \frac{1}{10}[5mAP + \sum_{mTP\in TP}(1 - min(1, mTP))] \tag{13}$$

The AP metric is using a match based on the 2D center distance d on the ground plane (over 0.5, 1.2 and 4 m threshold values) instead of the commonly used Intersection Over Union (IoU). AP is calculated as the normalized area under the P-R curve, however points where precision or recall is below 10% are discarded to confine the effect of noise, found in such low precision and recall areas [66]. Given a set of classes $\mathbb{C}$ and set of threshold distance values $\mathbb{D} = \{0.5, 1, 2, 4\}$ mAP is calculated as:

$$mAP = \frac{1}{|\mathbb{C}||\mathbb{D}|}\sum_{c\in\mathbb{C}}\sum_{d\in\mathbb{D}} AP_{c,d} \tag{14}$$

True Positive metrics are calculated for each true positive (TP) prediction. A prediction is considered as TP if its center distance is located within a $d = 2m$ distance threshold value from the corresponding ground truth 3D BBox's center in BEV. TP metrics are in native units, designed to be positive scalars and consist of:

- Average Translation Error (ATE), which is the euclidean center distance in BEV, in meters
- Average Scale Error (ASE), which is the 3D IoU difference after orientation and alignment given as $(1 - IoU)$
- Average Orientation Error (AOE), which is the smallest yaw angle difference between prediction and ground truth, in radians
- Average Velocity Error (AVE), which is the absolute velocity error as the L2 norm of the velocity differences in BEV, given in m/s
- Average Attribute Error (AAE), which is given as 1 minus attribute classification accuracy, $(1 - acc)$

For certain classes where certain metrics do not make sense such the AVE,AOE and AAE metrics for barriers and cones, these metrics are not considered. The mean True Positive (mTP) metrics is calculated as:

$$mTP = \frac{1}{\mathbb{C}}\sum_{c\in\mathbb{C}} TP_c \tag{15}$$

5.3. Waymo open dataset

Waymo Open Dataset (WOD) [12] is among the newest released datasets for autonomous driving. It consists of LiDAR data captured from 4 short-range LiDARs and one 1 mid-range LiDAR and camera data captured from five high resolution RGB cameras. The dataset is quite diverse since it was recorded in suburban and urban areas of San Francisco, Mountain View and Phoenix during different times of the day. High quality calibrations and transformations are provided between sensors and coordinate frames.

LiDAR data are provided in the form of range images. Each range image covers a 360° field of view around the car. A pixel in a range image corresponds to a LiDAR return and is encoded with its range, intensity, elongation, no label-zone, LiDAR to camera projection information and the vehicle pose at the time that the LiDAR point was captured. A "No label zone" feature indicates that the LiDAR point falls in an area ignored for labelling.

For the 3D object detection task, objects within a 75m radius around the car are labelled in the form of 3D bounding boxes for four classes, namely 'vehicle', 'pedestrian', 'cyclist' and 'sign'. Each 3D BBox is encoded by its center coordinates $[x, y, z]$, its length, width, height $[l, w, h]$ in meters, its heading angle in radians θ and its unique tracking id.

The commonly used AP metric is also used by WOD. However instead of the 40 point interpolated approach used by KITTI [51] to

Table 3
3D Object Detectors Evaluation on nuScenes 3D Benchmark.

Name/Year	NDS	mAP	mATE	mASE	mAOE	mAVE	AAE	mAP										Time (s)	Hardware
								Car	Pedestrian	Bus	Barrier	Traffic Cone	Truck	Trailer	Motorcycle	Construction Vehicle	Bicycle		
Voxel-based:																			
MEGVII [65] (2019)	63.30	52.80	0.300	0.247	0.38	0.245	0.14	81.10	80.10	54.90	65.70	70.90	48.50	42.90	51.50	10.50	22.30	-	-
SARPNET [68] (2020)	48.40	32.40	0.400	0.249	**0.763**	0.272	0.090	59.90	69.40	19.40	38.30	44.60	18.70	18.00	29.80	11.60	14.20	-	-
HotSpotNet [53] (2020)	66.00	59.30	0.274	0.239	0.384	**0.333**	0.133	83.10	81.30	56.40	**71.60**	73.00	50.90	53.30	**63.50**	**23.00**	36.60	-	-
Range-GC Net [76] (2020)	66.10	58.50	0.272	0.243	0.383	0.293	0.126	85.00	84.30	56.90	69.00	**79.10**	50.20	52.60	58.60	19.10	29.80	-	-
CenterPoint-single [56] (2020)	**67.30**	**60.30**	0.262	0.239	0.361	0.288	0.136	**85.20**	**84.60**	**63.60**	71.10	78.40	**53.50**	**56.00**	59.50	20.00	30.70	-	-
CVCNet-ensemble [14] (2020)	66.60	58.20	0.284	0.241	0.372	0.224	0.126	82.60	83.00	59.40	69.70	69.70	49.50	51.10	61.80	16.20	**38.80**	0.09	Tesla V100
Pillar-based:																			
PointPillars [21] (2019)	45.30	30.50	**0.517**	**0.290**	0.5	0.316	**0.368**	68.40	59.70	28.20	38.90	30.80	23.00	23.40	27.40	4.10	1.10	**0.016**	1080 Ti
SA-Det3D-DSA** [87] (2021)	59.20	47.00	0.317	0.248	0.438	0.300	0.129	81.20	73.30	57.20	55.30	60.60	43.80	47.80	32.10	11.30	7.90	-	-
Point-based:																			
✓3DSSD [43] (2020)	56.40	42.66	0.390	**0.290**	0.440	0.220	0.120	81.20	70.17	61.41	47.94	31.06	47.15	30.45	35.96	12.64	8.63	-	-

SA-Det3D-DSA**: As implemented for a PointPillars [21] backbone.
✓: The performance results were not found on the online nuScenes [66] benchmark and instead the results from the original paper of the method are presented.
For a method, that the results from its paper are not identical to the ones found on the online nuScenes [66] benchmark, the results from nuScenes are presented.

calculate AP, WOD calculates AP as the area under the P-R curve as:

$$AP = 100 \int_0^1 \max\{p(r') | r' >= r\} dr \tag{16}$$

where $p(r)$ is the P-R curve.

Along with the Average Precision (AP) metric, the Average Precision Heading (APH) metric is introduced to also include heading information. Similar to AP, the APH metric is calculated as:

$$APH = 100 \int_0^1 \max\{h(r') | r' >= r\} dr \tag{17}$$

where $h(r)$ is calculated similar to $p(r)$ by weighting each true positive by a heading accuracy $\min(|\Delta_\theta|, 2\pi - |\Delta_\theta|)/\pi$.

Concerning the difficulty of detecting a 3D object, two different levels namely LEVEL1 and LEVEL2 are defined. Objects assigned in LEVEL2 are considered the most difficult ones to detect. At first all objects without any LiDAR points are ignored and therefore are not assigned to either level. An object is assigned in LEVEL2 if it has less than 6 LiDAR points or if a human annotator assigns that object manually to that category. The rest of the objects are then assigned to LEVEL1.

For consistency in experiments, the Waymo team provides fixed sets for training, validation and testing.

6. Discussion

In this section, we present our findings and key insights for each class of methods, taking into account their operational pipeline, detection performance and efficiency. As defined in Section 4, the classification proposed in this work relies upon the input LiDAR SDR.

The detection performance is measured by evaluation metrics that have been detailed in Section 5. Although these metrics may slightly vary, depending on the dataset, they focus on measuring the precision and recall in detecting object classes in 3D space and in BEV. The efficiency is evaluated according to the reported inference time for each dataset. For the sake of clarity, the GPU hardware used for the evaluation of the detectors, is also reported. The performance evaluation for KITTI [51], nuScenes [66] and Waymo [12] datasets are given in Table 2, Table 3 and Tables 4 - 5 respectively.

Overall, in our comparative study, it is noticed that there is not a single 3D object detector that outperforms all the others for every single object class, for a given dataset. Some detectors perform better in detecting the 'car' class, while others are better at detecting the 'pedestrian' or 'cyclist' classes. Furthermore, the performance of most detectors is evaluated for KITTI [51] dataset. This is partially explained for 3D object detectors before 2019, since nuScenes [66] and Waymo [12] were not publicly available. On the other hand, for most recent 3D object detectors that are evaluated solely on KITTI [51] and only for the 'car' class, this raises questions for the overall performance and effectiveness of their methodology.

However, since many 3D object detectors provide their source code in public, as shown in Table 7, this enables the possibility to provide further insights on the effectiveness of their methodology by evaluating the methods on the remaining classes for KITTI [51] dataset as well as on other available datasets such as nuScenes [66] and Waymo [12].

In the sequel, we provide a detailed performance analysis for each class of methods based on a classification that relies upon the input LiDAR SDR.

6.1. Voxel-based detectors

From the description of the corresponding 3D object detectors given in Section 4.1, we can conclude that in the majority of the

Table 4
3D Object Detectors Evaluation on Waymo Validation Set for 3D Object Detection.

	LEVEL_1						LEVEL_2							
	Car		Pedestrian		Cyclist		Car		Pedestrian		Cyclist			
Name/Year	3D mAP	3D mAPH	3D mAP	3D mAPH	3D mAP	3D mAPH	3D mAP	3D mAPH	3D mAP	3D mAPH	3D mAP	3D mAPH	Time (s)	Hardware
Voxel-based:														
✓SECOND [33] (2018)	72.27	71.69	68.70	58.18	60.62	59.28	63.85	63.33	60.72	51.31	58.34	57.05	-	-
✓Part A^2 Net-Anchor [36] (2020)	77.05	76.51	75.24	66.87	68.60	67.36	68.47	67.97	66.18	58.62	66.13	64.93	-	-
CenterPoint [56] (2020)	76.70	76.20	**79.00**	**72.90**	-	-	68.80	68.30	**71.00**	**65.30**	-	-	0.077	Titan RTX
CVC-Net [14] (2020)	65.20	-	-	-	-	-	-	-	-	-	-	-	0.09	Tesla V100
Voxel-RCNN [77] (2020)	75.90	-	-	-	-	-	66.59	-	-	-	-	-	-	-
Pillar-based:														
PointPillars* [21] (2019)	56.62	-	59.25	-	-	-	-	-	-	-	-	-	**0.041**	-
AFDet [55] (2020)	63.69	-	-	-	-	-	-	-	-	-	-	-	-	-
Point-based:														
StarNet [82] (2019)	53.70	-	66.80	-	-	-	-	-	-	-	-	-	-	-
Dual Representation:														
MVF [15] (2020)	62.93	-	65.33	-	-	-	-	-	-	-	-	-	0.065	-
PV-RCNN v1 [18] (2020)	77.51	76.89	75.01	65.65	67.81	66.35	68.98	68.41	66.04	57.61	**65.39**	**63.98**	0.3	Titan RTX
Pillar OD [85] (2020)	69.8	-	72.51	-	-	-	-	-	-	-	-	-	0.066	Tesla V100
PV-RCNN v2 [86] (2021)	**78.79**	**78.21**	76.67	67.15	**68.98**	**67.63**	**70.26**	**69.71**	68.51	59.72	66.48	65.17	0.1	Titan RTX

✓: Re-implemented by [86], PointPillars*: Re-implemented by [15]

methods the voxel feature is extracted either through a VFE module and its variations, or as the mean values of points within the voxel. The binary occupancy value and hand-crafted voxel features are rarely used, usually by older methods. Furthermore, the voxelized data are processed by the 3DBN which has been proposed by SECOND [33] and its variations [36,64].

It is also noticed that after the 3DBN module, voxel-based detectors convert the data representation from 3D to a 2D, resulting in a BEV. This strategy results in a balanced architecture, between an increased 3D object detection performance and a low inference time. Methods that use solely a 3DBN module achieve decent detection performance but at the cost of a high inference time [17,19], while architectures that use solely a 2DBN demonstrate a low inference time and an increased detection performance in BEV but not in 3D [24,63]. Furthermore, all four aforementioned methods use either a binary occupancy value or a hand-crafted voxel feature. Concerning the corresponding 2DBN which is fed by a BEV feature map, the most common architecture used in the literature is a FPN [44] with variations that are mainly related to the number of input and output feature maps. For the Detector Networks, most recent voxel-based detectors are in favor of using an Anchor-free network compared to the Anchor-based RPN, due to its computational efficient structure and lack of defining and fine-tuning the anchor sizes for each object class.

According to the analysis of the operational pipeline as presented in Table 6, it is shown that exactly half of the 3D object detectors use a voxel-based LiDAR SDR. Many different architectures have been proposed, with most recent methods [36,53,56] achieving either state of the art or top detection performance for most datasets and object classes. The common strategy in the aforementioned three methods, is the effective encoding of the 3D geometric information of an object, through the prediction of its parts or discriminative points of it, along with their intra-object spatial locations. The majority of voxel-based detectors operate in one-stage, suggesting that a single inference pass results in an acceptable performance along with a low inference time, as shown in Table 2. There are also two-stage detectors that use either voxel or point based networks to extract a fine-grained representation of the 3D shape of objects and refine the predicted 3D BBox. As a result, an improved detection performance is observed at a slightly increased inference time. However, a recent detector [77] proposes a hybrid voxel/point based network for its second stage, achieving top ranking performance while retaining a low inference time usually found in one-stage detectors.

6.2. Pillar-based detectors

In the case of pillar-based 3D object detectors, the pillar feature is extracted entirely through point-based networks, like the VFE module [13] and its variations. Next, the construction of a BEV feature map is employed which sets the base for the use of a 2DBN module, thus discarding completely the computationally expensive 3DBN. In all cases, the 2DBN module consists of an FPN [44] architecture with variations that are mainly related to the number of input and output feature maps. Most methods use an Anchor-based RPN for object detection, however, similarly to voxel-based detectors, there is a trend towards an Anchor-free network for efficiency and simplicity [55]. Furthermore, all pillar-based methods are built to use one-stage.

Compared to all methods, pillar-based detectors achieve an overall satisfactory detection performance for the "easy" and "moderate" difficulty levels for all classes in KITTI [51] dataset, but slightly inferior for the hard difficulty level. Furthermore, in the more complex and demanding datasets of nuScenes [66] and Waymo [12], the performance of the pillar-based detector [21] is significantly inferior compared to the voxel-based detectors, even for the "easy" and "moderate" cases. We believe that this is due to the difficulty in encoding efficiently the 3D structural and geometric information of objects especially in far away or occluded cases. The reasons might be the limitation of the point-based network, namely the light-weight VFE which is used to extract the pillar feature vector, as well as the lack of a 3DBN module. We argue on our hypothesis by comparing the performance of PointPillars [21] and SECOND [33] in KITTI [51] and Waymo [12] datasets. Although PointPillars [21] achieves a slightly increased performance in KITTI [51] dataset, SECOND [33] outperforms PointPillars [21] by a large margin for Waymo [12] dataset. The structure of PointPillars [21] is similar to SECOND [33] with one main difference, the usage of a light-weight VFE module to extract a pillar feature and construct the BEV feature map, instead of constructing the BEV feature map through a 3DBN, as SECOND [33] does. However, when it comes to computational efficiency, pillar-based detectors are the most efficient ones since they achieve the lowest inference time among all detectors.

Table 5
3D Object Detectors Distance-based Evaluation on Waymo Validation Set for LEVEL_1 Objects.

	3D mAP								BEV mAP							
	Car			Pedestrian			Cyclist		Car			Pedestrian			Cyclist	
Name/Year	0-30m	30-50m	50m-∞	0-30m	30-50m	50m-∞	0-30m	30-50m	50m-∞	0-30m	30-50m	50m-∞	0-30m	30-50m	50m-∞	0-30m	30-50m	50m-∞
Voxel-based:																		
✓SECOND [33] (2018)	90.66	70.03	47.55	74.39	67.24	56.71	73.33	55.51	41.98	96.84	88.39	78.37	79.92	75.50	67.92	74.58	59.31	46.75
✓Part A^2 Net-Anchor [36] (2020)	92.35	75.91	54.06	81.87	73.65	62.34	**80.87**	62.57	45.04	97.52	90.35	80.12	84.49	79.22	70.34	**81.96**	**66.38**	48.15
CenterPoint [56] (2020)	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
CVC-Net [14] (2020)	86.30	63.84	36.65	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Voxel-RCNN [77] (2020)	92.49	74.09	53.15	-	-	-	-	-	-	97.62	87.34	77.7	-	-	-	-	-	-
Pillar-based:																		
PointPillars* [21] (2019)	81.01	51.75	27.24	67.99	57.01	41.29	-	-	-	92.1	74.06	55.47	75.02	67.11	53.86	-	-	-
AFDet [55] (2020)	87.38	62.19	29.27	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Point-based:																		
StarNet [82] (2019)	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Dual Representation:																		
MVF [15] (2020)	86.30	60.02	36.02	72.51	62.35	50.62	-	-	-	93.59	79.21	63.09	80.01	72.98	62.51	-	-	-
PV-RCNN v1 [18] (2020)	92.44	76.11	55.55	80.87	73.76	63.76	79.54	62.25	46.19	97.62	90.72	81.84	83.83	79.56	71.93	80.53	64.91	49.81
Pillar OD [85] (2020)	88.53	66.50	42.93	79.34	72.14	56.77	-	-	-	95.78	84.74	72.12	83.56	78.7	65.8	-	-	-
PV-RCNN v2 [86] (2021)	**93.05**	**77.70**	**57.38**	**82.41**	**75.42**	**66.01**	80.76	**63.10**	**47.40**	**97.98**	**91.21**	**82.56**	**85.10**	**80.59**	**73.30**	81.40	65.31	**50.64**

✓: Re-implemented by [86] PointPillars*: Re-implemented by [15]

6.3. Projection in BEV detectors

In the case of 3D object detectors that use a projection in BEV LiDAR SDR, hand-crafted features are employed to construct a BEV feature map and make use, for the rest of the architecture, of typical 2D object detection architectures, e.g. Faster-RCNN [4],with variations mostly observed in the architecture of the 2DBN.

Taking into account the overview of the methods, it turns out that this particular category demonstrate an overall reduced detection performance along with an increased inference time. Furthermore, it seems that this category became less appealing for being used by the research community since the appearance of pillar-based 3D object detectors.

6.4. Projection in a perspective front-view range image detectors

In the case of 3D object detectors that use solely a FV projection LiDAR SDR, the range image pixels are encoded with hand-crafted features and then introduced into a 2DBN for feature extraction. In this category there exist only two methods, as presented in Sections 4.4.1 and 4.4.2, respectively.

Both methods are evaluated solely in the 'car' class for KITTI [51] dataset, with the one of them [23] only for the BEV detection task. Despite of their low inference time, it seems that this particular category did not stimulate adequately the interest of the research community. A potential reason for that is the weakness of the FV range image to reflect the real scale of the objects. To remedy this, one of the two methods in this category [22] although it performs feature extraction in the FV range image, in the sequel it makes a correspondence to a BEV map for the detection part.

6.5. Point-based detectors

Point-based 3D object detectors operate solely in 3D space and they mostly utilize a PointNet++ [27] with encoder/decoder architecture as their 3DBN module. Along with the learned high-dimensional point-wise features, foreground/background point segmentation is performed. This segmentation information is used to guide an Anchor-free network, to output per point object proposals. The majority of point-based 3D object detectors consist of two-stages, with most of them following the point-based LiDAR SDR in their second stage. Nevertheless, it is demonstrated that using a voxel-based representation for the second stage of prediction refinement, results in an increased detection performance along with a reduced inference time [37]. On the contrary, for a graph-based representation, a marginal increase in the detection performance along with a remarkably increased inference time is observed [41]. However, a recent one-stage method [43] introduces a new 3DBN module of a PointNet++ [27] encoder architecture, resulting in a simplified and overall more promising structure concerning both the detection performance and computational efficiency, comparable with the top ranking voxel-based detectors.

The majority of point-based detectors achieve an overall satisfactory detection performance for KITTI [51] dataset, however they consist of two-stages, which implies an inadequate detection performance in a single inference pass. The extra second stage along with the computationally expensive 3DBN, results in a higher inference time. Compared to efficient detectors like voxel-based or pillar-based as shown in Table 2, point-based detectors are marginally suitable for a real time application. Due to this limitation, improvements of point-based detectors have been reported recently. In particular, the approach in [43], employs an efficient 3DBN and has been evaluated in nuScenes [66], while the approach in [82], evaluated in Waymo [12], remedies the disadvantage of point-based detectors by processing regions of interested instead of the whole point cloud at once.

Table 6
Operational Pipeline for LiDAR-based 3D Object Detectors.

Name/Year	Number of Stages	LiDAR SDR	Feature Extraction		Core Object Detection			
					Detector Networks		Prediction Refinement	
			3D Backbone	2D Backbone	Anchor-based	Anchor-free	3D-based	2D-Based
Voxel:								
Vote3Deep [19] (2017)	one	voxel	3D CNN (Voting Scheme)	-	-	3D CNN	-	-
3D FCN [17] (2017)	one	voxel	3D CNN	-	-	3D CNN	-	-
PIXOR [24] (2018)	one	voxel	-	BEV (FPN SISO)	-	1D/2D CNN	-	-
HDNet [63] (2018)	one	voxel	-	BEV (FPN SISO)	-	1D/2D CNN	-	-
VoxelNet [13] (2018)	one	voxel	3D CNN	BEV (FPN SISO)	2D RPN	-	-	-
SECOND [33] (2018)	one	voxel	3D sparse CNN	BEV (FPN SISO)	2D RPN	-	-	-
3D BN-Net [64] (2019)	one	voxel	3D sparse CNN (FPN SIMO)	BEV (FPN MISO)	2D RPN	-	-	-
Fast Point RCNN [57] (2019)	two	voxel	3D sparse CNN	BEV (FPN SISO)	2D RPN	-	point	-
MEGVII [65] (2019)	one	voxel	3D sparse CNN	BEV (FPN SISO)	2D RPN	-	-	-
Patch Refinement [60] (2019)	two	voxel	3D CNN	BEV (FPN SIMO)	2D RPN	-	voxel	-
SARPNET [68] (2020)	one	voxel	3D sparse CNN	BEV (FPN SISO)	2D RPN (Shape Attention)	-	-	-
FoF Net [69] (2020)	one	voxel	3D sparse CNN	BEV (FPN SISO)	2D RPN	-	-	-
Part A^2 Net-Anchor [36] (2020)	two	voxel	3D sparse CNN enc/dec	BEV (FPN SISO)	2D RPN	-	voxel	-
Part A^2 Net-free [36] (2020)	two	voxel	3D sparse CNN enc/dec	-	-	FC	voxel	
SegVoxelNet [70] (2020)	one	voxel	3D sparse CNN	BEV (FPN SISO + U-Net)	2D RPN (Depth Aware)	-	-	-
HotSpotNet [53] (2020)	one	voxel	3D sparse CNN	BEV (FPN SISO)	-	1D/2D CNN	-	-
SA-SSD [72] (2020)	one	voxel	3D sparse CNN	BEV (VGG)	2D RPN +PSWarp	-	-	-
CIA-SSD [73] (2020)	one	voxel	3D sparse CNN	BEV (FPN SISO)	2D RPN (IoU NMS)	-	-	-
CenterNet3D [49] (2020)	one	voxel	3D sparse CNN	BEV	-	1D/2D CNN	-	-
Assosiate-3Ddet [74] (2020)	one	voxel	3D sparse CNN	2xBEV	2D RPN	-	-	-
CenterPoint [56] (2020)	two	voxel	3D sparse CNN	BEV (FPN SISO)	-	1D/2D CNN	point	-
CVC-Net [14] (2020)	one	HCS voxel	3D sparse CNN	FV (U-Net) + BEV (U-Net)	-	1D/2D CNN	-	-
Range-GC Net [76] (2020)	one	HCS voxel	2 x 3D sparse CNN	BEV (FPN SISO)	-	1D/2D CNN	-	-
Voxel-RCNN [77] (2020)	two	voxel	3D sparse CNN	BEV (FPN SISO)	2D RPN	-	point/voxel	-
Pillar-based:								
PointPillars [21] (2019)	one	pillar	-	BEV (FPN SISO)	2D RPN	-	-	-
TANet [79] (2020)	one	pillar	-	BEV (FPN SIMO)	2D RPN	-	-	-
Voxel-FPN [50] (2020)	one	pillar	-	BEV (FPN MIMO)	2D RPN	-	-	-
HVNet [16] (2020)	one	pillar	-	BEV (FPN MIMO)	2D RPN	-	-	-
AFDet [55] (2020)	one	pillar	-	BEV (FPN SISO)	-	1D/2D CNN	-	-
Projection in BEV:								
RT3D [62] (2018)	two	proj.BEV	-	BEV (ResNet-50)	2D RPN	-		2D CNN
BirdNet [25] (2018)	two	proj.BEV	-	BEV (VGG-16)	2D RPN	-	-	2D CNN
BirdNet+ [61] (2020)	two	proj.BEV	-	BEV (ResNet-50)	2D RPN	-	-	FC
Projection in FV:								
LaserNet [23] (2019)	one	proj.FV	-	FV (DLA)	-	1D CNN	-	-
Range-RCNN [22] (2020)	two	proj.FV	-	FV (FPN SISO) + BEV (FPN SISO)	2D RPN	-	voxel	-
Point cloud:								
Point-RCNN [40] (2019)	two	point cloud	PointNet++ enc/dec	-	-	FC	point	-
STD [37] (2019)	two	point cloud	PointNet++ enc/dec	-	PointNet	-	voxel	-
Point-RGCN [41] (2019)	two	point cloud	PointNet++ enc/dec	-	-	FC	graph	-
StarNet [82] (2019)	one	point cloud	StarNet feature extractor (FC)	-	FC	-	-	-
3DSSD [43] (2020)	one	point cloud	PointNet++ enc	-	-	FC	-	-
3D IOU Net [39] (2020)	two	point cloud	PointNet++ enc/dec	-	-	FC	point	-
SE-RCNN [38] (2020)	two	point cloud	PointNet++ enc/dec	-	-	FC	point	-
WS3D [42] (2020)	two	point cloud	PointNet++ enc/dec	-	-	FC	point	-
Graph:								
Point-GNN [26] (2020)	one	graph	GNN (MLPs)	-	-	MLPs	-	-
Dual Representation:								
MVF [15] (2020)	one	pillar/proj.FV	-	FV (FPN SISO) + 2xBEV (FPN SISO)	2D RPN	-	-	-
Pillar OD [85] (2020)	one	pillar/proj.FV	-	FV (ResNet) + 2xBEV (ResNet, FPN SISO)	-	1D/2D CNN	-	-
PV-RCNN v1 [18] (2020)	two	voxel/point	3D sparse CNN/PointNet++ enc	BEV (FPN SISO)	2D RPN	-	point	-
PV-RCNN v2 [86] (2021)	two	voxel/point	3D sparse CNN/PointNet++ enc	BEV (FPN SISO)	2D RPN	-	point	-
SA-Det3D-DSA* [87] (2021)	two	voxel/point	3D sparse CNN/PointNet++ enc	BEV (FPN SISO) + DSA	2D RPN	-	point(+DSA)	-

SA-Det3D*: As implemented for a PV-RCNN v1 [18] backbone

6.6. Graph-based detectors

In the case of graph-based methods, only one work has been reported in the literature, namely Point-GNN [26]. Once the graph representation of the point cloud is constructed, graph neural networks are applied consecutively, resulting in a end-to-end graph-based 3D object detection without using other modules like 3DBN, 2DBN etc.

The reported high inference time of Point-GNN [26] for KITTI [51] dataset, as shown in Table 2, might be the main reason for being less appealing, in particular, for autonomous driving applications. However, the detection performance is satisfactory and overall promising, therefore demonstrating a research potential into implementing efficient graph-based detectors.

6.7. Dual point cloud representation detectors

Last but not least, in the case of dual representation detectors, there exist two approaches. In the first one, followed by [15,85] two distinct LiDAR SDRs are constructed to extract view-dependent feature vectors and subsequently fuse them, resulting in a point-wise multi-view feature representation. In the second approach followed by [18,86], the first LiDAR SDR, namely the voxel-based, is used to extract multi-scale feature vectors and predict 3D BBoxes, and the second LiDAR SDR, namely the point-based, is used to further process and aggregate the multi-scale voxel feature vectors onto a small number of keypoints to assist the 3D BBox refinement. Dual representation detectors don't follow a specific pattern concerning their architecture but instead use the corresponding operational modules, like 3DBN/2DBN etc, based on their corresponding LiDAR SDR as described in Section 4.7.

Both approaches aim into learning and encoding more efficiently the 3D structural information of objects. The first approach of learning multi-view features as implemented by [15,85] achieves an overall increased detection performance, however quite inferior when compared with the second approach of learning multi-scale features, since the detectors of [18,86] achieve state of the art detection performance for most classes and difficulty levels in KITTI [51] and Waymo [12] datasets. The inference time of all dual representation methods, with the exception of [18] for Waymo [12], is slightly increased compared to voxel-based detectors, however still suitable for a real time application.

6.8. The role of attention

Attention or self attention modules improve the overall performance of 3D object detectors. In particular, the work in TA-Net [79] implements a triple attention (TA) module as described in Section 4.2.2, to perform attention locally during the extraction of the pillar feature vector.

In an attempt to provide a global attention among feature vectors, a self attention module as introduced by [88] is implemented for 3D object detectors by the work of [87]. The attention module is implemented under two distinct networks, namely Full Self Attention (FSA) and Deformable Self Attention (DSA) which can be plugged into existing 3D object detectors and improve their overall performance. This improvement has been demonstrated for Point-Pillars [21], SECOND [33], Point-RCNN[40], PV-RCNN v1 [18], as described in [87].

7. Future research trends and open problems

In this section we present our insights concerning the future trends, research potential and open problems in 3D object detection for autonomous driving applications.

7.1. Application in embedded systems

In the literature, most methods report their inference time for GPU hardware which is usually found in desktop computers, omitting the case for edge computing devices, usually found in embedded systems, which is the hardware that an autonomous vehicle or mobile robot is naturally using. Furthermore, the complexity of the methods in terms of network size and memory allocation is rarely reported.

Certain detectors like the pillar-based ones [21,55], focus on becoming computationally friendly to embedded systems by adopting a simple and efficient architecture, resulting in a reduced number of network parameters, or by further optimizing their architectures through specially designed libraries for GPU inference, like TensorRT from NVIDIA, respectively. Nevertheless, even these two methods were not tested or evaluated on edge computing devices.

Furthermore, the energy consumption of the methods is never addressed as a critical constraint or reported as a benchmark, albeit being one of the most important factors for an embedded system, especially for electric autonomous vehicles and mobile robots.

Inspired by the recent experimental contributions of [95], performing a comparative study and benchmarking of Deep Learning methods for semantic segmentation inference on the Nvidia Jetson TX1, and of [96], evaluating the energy consumption of a Deep Learning 2D object detector as a function of its quality of service on the Nvidia Jetson Nano, we identify the research potential of similar comparative and energy profiling studies, for LiDAR-based 3D object detectors in embedded systems. We believe that the publicly available code of detectors as shown in Table 7 will further assist to realize this research attempt.

7.2. Towards efficient hybrid networks

Recent advances as in the work of PV-RCNN v2 [86] and Voxel-RCNN [77], focus on replacing certain computationally expensive operations of point-based networks, e.g. the set abstraction and ball query operations in PointNet++ [27], with more efficient voxel-inspired ones, while still retaining the rest of the point-based network unchanged. Such solutions lead to hybrid networks that utilize the best out of both, an increased detection performance and a low inference time. Similar approaches could also be applied in graph neural networks to reduce the computational burden of graph neural networks. There is therefore a research potential in identifying computationally inefficient bottlenecks in existing architectures and improving them through hybrid approaches.

Most 3D object detectors use a single LiDAR SDR, with recent methods [15,18,85,86] employing a dual LiDAR SDR to extract more rich and discriminative feature vectors. The only limiting constraint against using even more LiDAR SDRs in the same detector, is the inference time of the architecture which should be suitable for a real time application. Hybrid networks aim to further reduce the inference time of a detector, which facilitates the usage of an extra third or even a fourth LiDAR SDR, for learning more discriminative features.

7.3. Sparsity of LiDAR point clouds

The inherent sparsity of a point cloud has drawn interest in the research community, especially for 3D object completion and shape generation architectures [97,98]. In 3D object detection for autonomous driving, object and shape completion techniques have been used as a pre-processing step during training, to associate and learn robust features through a domain adaptation strategy [74]. However, object completion and shape generation architectures have not been used by 3D object detectors during the in-

Table 7
Links to the Official Code Repositories of LiDAR-based 3D Object Detectors.

Name/Year	Link
Voxel-based:	
SECOND [33] (2018)	https://github.com/traveller59/second.pytorch
Part A^2 Net-Anchor [36] (2020)	https://github.com/open-mmlab/OpenPCDet
SA-SSD [72] (2020)	https://github.com/skyhehe123/SA-SSD
CenterPoint [56] (2020)	https://github.com/tianweiy/CenterPoint
CIA-SSD [73] (2020)	https://github.com/Vegeta2020/CIA-SSD
CenterNet3D [49] (2020)	https://github.com/wangguojun2018/CenterNet3d
Assosiate-3Ddet [74] (2020)	https://github.com/dleam/Associate-3Ddet
Pillar-based:	
PointPillars [21] (2019)	https://github.com/nutonomy/second.pytorch
TANet [79] (2020)	https://github.com/happinesslz/TANet
Projection in BEV:	
BirdNet+ [61] (2020)	https://github.com/AlejandroBarrera/BirdNet2
Point-based:	
PointRCNN [40] (2019)	https://github.com/open-mmlab/OpenPCDet
3DSSD [43] (2020)	https://github.com/Jia-Research-Lab/3DSSD
WS3D [42] (2020)	https://github.com/hlesmqh/WS3D
Graph-based:	
Point GNN [26] (2020)	https://github.com/WeijingShi/Point-GNN
Dual Representation:	
PV-RCNN v1 [18] (2020)	https://github.com/open-mmlab/OpenPCDet
Pillar OD [85] (2020)	https://github.com/WangYueFt/pillar-od
SA-Det3D-DSA [87] (2021)	https://github.com/AutoVision-cloud/SA-Det3D

ference of the LiDAR point cloud, which could lead to an increased detection performance.

7.4. Towards self-Supervised learning

In the work of WS3D [42], an existing two-stage 3D object detector, namely Point-RCNN [40], is used as a backbone for implementing and testing a weakly supervised training approach. The authors of [42], construct manually weakly annotated data and use them for training the first stage of the detector while the detailed labelled data are used to train the second stage. So far, the work of [42] is the closest approximation towards self-supervised learning for LiDAR-based 3D object detectors in autonomous driving applications. Due to the difficulty of obtaining detailed labelled data, self-supervised and semi-supervised learning for 3D object detectors is of great research interest.

7.5. Performance evaluation for complex realistic scenarios

The presented 3D object detection methods are evaluated on the benchmarks of one, or more, of the three publicly available datasets, namely KITTI [51], nuScenes [66] and Waymo [12]. However, these three datasets, along with the rest shown in Table 1, exhibit limitations when compared to a more complex and realistic scenario for 3D object detection. These limitations include the reduced scene diversity, the lack of noisy data and the small number of object classes.

Concerning the scene diversity, all of the publicly available datasets contain scenes from urban or highway areas that are almost flat with a marginal ground inclination. As a result, all 3D object detectors aim to predict for each object a 3D BBox pose of 4 Degrees of Freedom (DoF), that stands for the x,y,z location and yaw angle orientation. For a more challenging case of 3D object detection in autonomous driving, it is essential to include scenes where the objects may not always be oriented in the same plane as the ego vehicle, i.e. in the case of crossing roads that have different ground inclination levels. Such cases may appear both in urban and rural areas. In such demanding scenarios, the current 3D BBox prediction of a 4 DoF pose may be insufficient and therefore may lead the 3D object detectors to extend the predicted pose from a 4-DoF to a 5-DoF, thus including the pitch angle of the 3D BBox.

Concerning noisy data, the adverse weather conditions such as fog, rain and snow are the main natural phenomena for introducing extra noise in the LiDAR sensor measurements, since the emitted laser beam from the LiDAR sensor may be refracted. However, the appearance of this type of noise in benchmarking datasets has been partially addressed by the nuScenes [66] and Waymo [12] datasets. Besides natural phenomena, extra interference noise may be introduced by the simultaneous emission of multiple LiDAR sensors. This is not referred to the multiple LiDAR sensors of a single vehicle, where the installed LiDAR sensors configuration is already known, but in the case where other vehicles in the scene also use LiDAR sensors. To the best of our knowledge, all of the publicly available datasets were recorded in scenes where the ego vehicle was the only one with installed LiDAR sensor/sensors. Therefore, in those cases, there is a potential of an extra introduced noise in the LiDAR sensor measurements. Both the adverse weather conditions and the interference from other LiDAR sensors demonstrate a research interest to compensate for noisy data i.e. by certain pre-processing algorithms to filter out noisy data before feeding the point cloud to existing 3D object detectors.

Current benchmarking datasets evaluate the performance of 3D object detectors for a certain amount of object classes. For most datasets the number of object classes equals to three, with nuScenes [66] containing ten classes. In a realistic scenario, the number of object classes to be detected may further increase to reflect a real context. Furthermore, the uneven class distribution of a dataset causes a class-imbalance problem during training, which becomes quite severe as the number of object classes increases. MEGVII detector [65] has addressed the class-imbalance for the nuScenes [66] dataset. However, there is still a research potential for more effective strategies, especially for ones using anchor-free networks due to their demonstrated efficiency. Additionally, transfer learning could also be used to re-train a 3D object detector into adapting to new object classes, based on a small and class-balanced amount of labelled data. This strategy could also be used in conjunction to self-supervised learning.

8. Conclusions

In the last few years, 3D object detection using solely LiDAR sensor data has become an active research field, mainly due to the

advance of Deep Learning architectures and available LiDAR sensors, along with its potential in commercial applications for autonomous driving.

In this paper, an analysis of the operational pipeline of LiDAR-based 3D object detectors is performed, and the individual operational modules are presented and discussed through a unified frame to establish a common ground among detectors. The detectors are classified and presented individually through the aforementioned unified frame. Furthermore, the detectors performance is reported according to the benchmarks and evaluation metrics of three open datasets for autonomous driving applications. An insightful discussion summarizes our findings and key elements that comprise a research potential for future work.

To the best of our knowledge, this work is the most complete in the literature that aims to provide a useful insight on the advantages and drawbacks of each method.

Disclosure of conflicts of interest

We declare that we have no conflict in our work with any companies or organizations.

Declaration of Competing Interest

None.

CRediT authorship contribution statement

Georgios Zamanakos: Conceptualization, Methodology, Validation, Investigation, Resources, Writing – review & editing. **Lazaros Tsochatzidis:** Conceptualization, Methodology, Validation, Investigation, Resources, Writing – review & editing, Funding acquisition. **Angelos Amanatiadis:** Conceptualization, Methodology, Validation, Investigation, Resources, Writing – review & editing, Funding acquisition. **Ioannis Pratikakis:** Conceptualization, Methodology, Validation, Investigation, Resources, Writing – review & editing, Supervision, Project administration, Funding acquisition.

Acknowledgments

This research has been co-financed by the European Union and Greek national funds through the Operational Program Competitiveness, Entrepreneurship and Innovation, under the call RESEARCH-CREATE-INNOVATE (project code:T2EDK-02743). We would also like to thank NVIDIA Corporation, which kindly donated the Titan X GPU, that has been used for this research.

References

[1] Urmson C, Anhalt J, Bagnell D, Baker C, Bittner R, Clark M, et al. Autonomous driving in urban environments: boss and the urban challenge. J Field Rob 2008;25(8):425–66.
[2] Levinson J, Askeland J, Becker J, Dolson J, Held D, Kammel S, et al. Towards fully autonomous driving: Systems and algorithms. In: IEEE Intelligent Vehicles Symposium; 2011. p. 163–8.
[3] Feng D, Haase-Schütz C, Rosenbaum L, Hertlein H, Glaeser C, Timm F, et al. Deep multi-modal object detection and semantic segmentation for autonomous driving: datasets, methods, and challenges. IEEE Trans Intell Transp Syst 2020.
[4] Ren S, He K, Girshick R, Sun J. Faster r-cnn: towards real-time object detection with region proposal networks. IEEE Trans Pattern Anal Mach Intell 2016;39(6):1137–49.
[5] Liu W, Anguelov D, Erhan D, Szegedy C, Reed S, Fu C-Y, et al. Ssd: Single shot multibox detector. In: European Conference on Computer Vision; 2016. p. 21–37.
[6] Bochkovskiy A., Wang C.-Y., Liao H.-Y. M.. Yolov4: Optimal speed and accuracy of object detection. arXiv preprint: 2004.10934, 2020.
[7] Guo Y, Wang H, Hu Q, Liu H, Liu L, Bennamoun M. Deep learning for 3D point clouds: a survey. IEEE Trans Pattern Anal Mach Intell 2020.
[8] Arnold E, Al-Jarrah OY, Dianati M, Fallah S, Oxtoby D, Mouzakitis A. A survey on 3d object detection methods for autonomous driving applications. IEEE Trans Intell Transp Syst 2019;20(10):3782–95.
[9] Li Y, Ma L, Zhong Z, Liu F, Chapman MA, Cao D, et al. Deep learning for liDAR point clouds in autonomous driving: a review. IEEE Trans Neural Netw Learn Syst 2020:1–21.
[10] Wu Y, Wang Y, Zhang S, Ogai H. Deep 3D object detection networks using lidar data: a review. IEEE Sens J 2020;21(2):1152–71.
[11] Fernandes D, Silva A, Névoa R, Simões C, Gonzalez D, Guevara M, et al. Point–cloud based 3D object detection and classification methods for self-driving applications: a survey and taxonomy. Information Fusion 2021;68:161–91.
[12] Sun P, Kretzschmar H, Dotiwalla X, Chouard A, Patnaik V, Tsui P, et al. Scalability in perception for autonomous driving: Waymo open dataset. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition; 2020. p. 2446–54.
[13] Zhou Y, Tuzel O. Voxelnet: End-to-end learning for point cloud based 3D object detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition; 2018. p. 4490–9.
[14] Chen Q, Sun L, Cheung E, Yuille AL. Every view counts: cross-view consistency in 3D object detection with hybrid-cylindrical-spherical voxelization. Adv Neural Inf Process Syst 2020a;33.
[15] Zhou Y, Sun P, Zhang Y, Anguelov D, Gao J, Ouyang T, et al. End-to-end multi-view fusion for 3D object detection in lidar point clouds. In: Conference on Robot Learning. PMLR; 2020a. p. 923–32.
[16] Ye M, Xu S, Cao T. Hvnet: Hybrid voxel network for lidar based 3D object detection. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition; 2020a. p. 1631–40.
[17] Li B. 3D fully convolutional network for vehicle detection in point cloud. In: IEEE/RSJ International Conference on Intelligent Robots and Systems; 2017. p. 1513–18.
[18] Shi S, Guo C, Jiang L, Wang Z, Shi J, Wang X, et al. Pv-rcnn: Point-voxel feature set abstraction for 3D object detection. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition; 2020a. p. 10529–38.
[19] Engelcke M, Rao D, Wang DZ, Tong CH, Posner I. Vote3deep: Fast object detection in 3D point clouds using efficient convolutional neural networks. In: IEEE International Conference on Robotics and Automation; 2017. p. 1355–61.
[20] Qi CR, Su H, Mo K, Guibas LJ. Pointnet: Deep learning on point sets for 3D classification and segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition; 2017a. p. 652–60.
[21] Lang AH, Vora S, Caesar H, Zhou L, Yang J, Beijbom O. Pointpillars: Fast encoders for object detection from point clouds. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition; 2019. p. 12697–705.
[22] Liang Z, Zhang M, Zhang Z, Zhao X, Pu S. Rangercnn: towards fast and accurate 3D object detection with range image representation. arXiv preprint arXiv:200900206 2020.
[23] Meyer GP, Laddha A, Kee E, Vallespi-Gonzalez C, Wellington CK. Lasernet: An efficient probabilistic 3D object detector for autonomous driving. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition; 2019. p. 12677–86.
[24] Yang B, Luo W, Urtasun R. Pixor: Real-time 3D object detection from point clouds. In: Proceedings of the IEEE conference on Computer Vision and Pattern Recognition; 2018a. p. 7652–60.
[25] Beltrán J, Guindel C, Moreno FM, Cruzado D, Garcia F, De La Escalera A. Birdnet: a 3D object detection framework from lidar information. In: IEEE International Conference on Intelligent Transportation Systems; 2018. p. 3517–23.
[26] Shi W, Rajkumar R. Point-gnn: Graph neural network for 3D object detection in a point cloud. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition; 2020. p. 1711–19.
[27] Qi CR, Yi L, Su H, Guibas LJ. Pointnet++: deep hierarchical feature learning on point sets in a metric space. arXiv preprint arXiv:170602413 2017b.
[28] Scarselli F, Gori M, Tsoi AC, Hagenbuchner M, Monfardini G. The graph neural network model. IEEE Trans Neural Networks 2008;20(1):61–80.
[29] Graham B. Spatially-sparse convolutional neural networks. arXiv preprint arXiv:14096070 2014.
[30] Graham B. Sparse 3D convolutional neural networks. arXiv preprint arXiv:150502890 2015.
[31] Graham B, van der Maaten L. Submanifold sparse convolutional networks. arXiv preprint arXiv:170601307 2017.
[32] Wang DZ, Posner I. Voting for voting in online point cloud object detection.. In: Robotics: Science and Systems, 1. Rome, Italy; 2015. p. 10–15607.
[33] Yan Y, Mao Y, Li B. Second: sparsely embedded convolutional detection. Sensors 2018;18(10):3337.
[34] Ioffe S, Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: International Conference on Machine Learning. PMLR; 2015. p. 448–56.
[35] Nair V, Hinton GE. Rectified linear units improve restricted boltzmann machines. In: International Conference on Machine Learning; 2010. p. 807–14.
[36] Shi S, Wang Z, Shi J, Wang X, Li H. From points to parts: 3D object detection from point cloud with part-aware and part-aggregation network. IEEE Trans Pattern Anal Mach Intell 2020b.
[37] Yang Z, Sun Y, Liu S, Shen X, Jia J. STD: Sparse-to-dense 3D object detector for point cloud. In: Proceedings of the IEEE/CVF International Conference on Computer Vision; 2019. p. 1951–60.
[38] Zhou D, Fang J, Song X, Liu L, Yin J, Dai Y, et al. Joint 3D instance segmentation and object detection for autonomous driving. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition; 2020b. p. 1839–49.

[39] Li J, Luo S, Zhu Z, Dai H, Krylov AS, Ding Y, et al. 3D Iou-net: iou guided 3D object detector for point clouds. arXiv preprint arXiv:200404962 2020.
[40] Shi S, Wang X, Li H. Pointrcnn: 3D object proposal generation and detection from point cloud. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition; 2019. p. 770–9.
[41] Zarzar J, Giancola S, Ghanem B. Pointrgcn: graph convolution networks for 3D vehicles detection refinement. arXiv preprint arXiv:191112236 2019.
[42] Meng Q, Wang W, Zhou T, Shen J, Van Gool L, Dai D. Weakly supervised 3D object detection from lidar point cloud. In: European Conference on Computer Vision; 2020. p. 515–31.
[43] Yang Z, Sun Y, Liu S, Jia J. 3DSSD: Point-based 3D single stage object detector. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition; 2020. p. 11040–8.
[44] Lin T-Y, Dollár P, Girshick R, He K, Hariharan B, Belongie S. Feature pyramid networks for object detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition; 2017. p. 2117–25.
[45] Ronneberger O, Fischer P, Brox T. U-net: Convolutional networks for biomedical image segmentation. In: International Conference on Medical Image Computing and Computer-assisted Intervention; 2015. p. 234–41.
[46] He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition; 2016. p. 770–8.
[47] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:14091556 2014.
[48] Dai J, Qi H, Xiong Y, Li Y, Zhang G, Hu H, et al. Deformable convolutional networks. In: Proceedings of the IEEE International Conference on Computer Vision; 2017. p. 764–73.
[49] Wang G, Tian B, Ai Y, Xu T, Chen L, Cao D. Centernet3d: an anchor free object detector for autonomous driving. arXiv preprint arXiv:200707214 2020a.
[50] Kuang H, Wang B, An J, Zhang M, Zhang Z. Voxel-FPN: multi-scale voxel feature aggregation for 3D object detection from lidar point clouds. Sensors 2020;20(3):704.
[51] Geiger A, Lenz P, Urtasun R. Are we ready for autonomous driving? the kitti vision benchmark suite. In: IEEE Conference on Computer Vision and Pattern Recognition; 2012. p. 3354–61.
[52] Girshick R, Donahue J, Darrell T, Malik J. Rich feature hierarchies for accurate object detection and semantic segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition; 2014. p. 580–7.
[53] Chen Q, Sun L, Wang Z, Jia K, Yuille A. Object as hotspots: An anchor-free 3D object detection approach via firing of hotspots. In: European Conference on Computer Vision; 2020b. p. 68–84.
[54] Zhou X, Wang D, Krähenbühl P. Objects as points. arXiv preprint arXiv:190407850 2019.
[55] Ge R, Ding Z, Hu Y, Wang Y, Chen S, Huang L, et al. Afdet: anchor free one stage 3D object detection. arXiv preprint arXiv:200612671 2020.
[56] Yin T, Zhou X, Krähenbühl P. Center-based 3D object detection and tracking. arXiv preprint arXiv:200611275 2020.
[57] Chen Y, Liu S, Shen X, Jia J. Fast point r-cnn. In: Proceedings of the IEEE/CVF International Conference on Computer Vision; 2019. p. 9775–84.
[58] Li G, Müller M, Qian G, Delgadillo IC, Abualshour A, Thabet A, et al. DeepGCNs: making GCNs go as deep as CNNs. arXiv preprint arXiv:191006849 2019a.
[59] Wang Y, Sun Y, Liu Z, Sarma SE, Bronstein MM, Solomon JM. Dynamic graph cnn for learning on point clouds. ACM Trans Graph 2019;38(5):1–12.
[60] Lehner J, Mitterecker A, Adler T, Hofmarcher M, Nessler B, Hochreiter S. Patch refinement–localized 3D object detection. arXiv preprint arXiv:191004093 2019.
[61] Barrera A, Guindel C, Beltrán J, García F. Birdnet+: End-to-end 3D object detection in lidar bird's eye view. In: IEEE International Conference on Intelligent Transportation Systems; 2020. p. 1–6.
[62] Zeng Y, Hu Y, Liu S, Ye J, Han Y, Li X, et al. Rt3d: real-time 3-d vehicle detection in lidar point cloud for autonomous driving. IEEE Rob Autom Lett 2018;3(4):3434–40.
[63] Yang B, Liang M, Urtasun R. Hdnet: Exploiting hd maps for 3D object detection. In: Conference on Robot Learning. PMLR; 2018b. p. 146–55.
[64] Li X, Guivant J, Kwok N, Xu Y, Li R, Wu H. Three-dimensional backbone network for 3D object detection in traffic scenes. arXiv preprint arXiv:190108373 2019b.
[65] Zhu B, Jiang Z, Zhou X, Li Z, Yu G. Class-balanced grouping and sampling for point cloud 3D object detection. arXiv preprint arXiv:190809492 2019.
[66] Caesar H, Bankiti V, Lang AH, Vora S, Liong VE, Xu Q, et al. nuscenes: A multimodal dataset for autonomous driving. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition; 2020. p. 11621–31.
[67] nuTonomy. nuscenes 3D object detection challenge, WAD, CVPR. 2019. URL https://www.nuscenes.org/object-detection?externalData=all&mapData=all&modalities=Any.
[68] Ye Y, Chen H, Zhang C, Hao X, Zhang Z. Sarpnet: shape attention regional proposal network for lidar-based 3D object detection. Neurocomputing 2020b;379:53–63.
[69] Wang L, Fan X, Chen J, Cheng J, Tan J, Ma X. 3D Object detection based on sparse convolution neural network and feature fusion for autonomous driving in smart cities. Sustainable Cities and Society 2020b;54:102002.
[70] Yi H, Shi S, Ding M, Sun J, Xu K, Zhou H, et al. Segvoxelnet: Exploring semantic context and depth-aware features for 3D vehicle detection from point cloud. In: IEEE International Conference on Robotics and Automation; 2020. p. 2274–80.
[71] Wang F, Jiang M, Qian C, Yang S, Li C, Zhang H, et al. Residual attention network for image classification. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition; 2017. p. 3156–64.
[72] He C, Zeng H, Huang J, Hua X-S, Zhang L. Structure aware single-stage 3D object detection from point cloud. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition; 2020. p. 11873–82.
[73] Zheng W, Tang W, Chen S, Jiang L, Fu C-W. Cia-ssd: confident iou-aware single-stage object detector from point cloud. arXiv preprint arXiv:201203015 2020.
[74] Du L, Ye X, Tan X, Feng J, Xu Z, Ding E, et al. Associate-3ddet: perceptual-to–conceptual association for 3D point cloud object detection. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition; 2020. p. 13329–38.
[75] nuTonomy. nuscenes 3D object detection challenge, 5th AI driving olympics, NeurIPS. 2020. URL https://www.nuscenes.org/object-detection?externalData=all&mapData=all&modalities=Any.
[76] Rapoport-Lavie M, Raviv D. It'S all around you: range-guided cylindrical network for 3D object detection. arXiv preprint arXiv:201203121 2020.
[77] Deng J, Shi S, Li P, Zhou W, Zhang Y, Li H. Voxel r-CNN: towards high performance voxel-based 3D object detection. arXiv preprint arXiv:201215712 2020.
[78] Hu Q, Yang B, Xie L, Rosa S, Guo Y, Wang Z, et al. Randla-net: Efficient semantic segmentation of large-scale point clouds. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition; 2020. p. 11108–17.
[79] Liu Z, Zhao X, Huang T, Hu R, Zhou Y, Bai X. Tanet: Robust 3D object detection from point clouds with triple attention. In: Proceedings of the AAAI Conference on Artificial Intelligence, 34; 2020. p. 11677–84.
[80] Hu J, Shen L, Sun G. Squeeze-and-excitation networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition; 2018. p. 7132–41.
[81] Yu F, Wang D, Shelhamer E, Darrell T. Deep layer aggregation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition; 2018. p. 2403–12.
[82] Ngiam J, Caine B, Han W, Yang B, Chai Y, Sun P, et al. Starnet: targeted computation for object detection in point clouds. arXiv preprint arXiv:190811069 2019.
[83] Qi CR, Litany O, He K, Guibas LJ. Deep hough voting for 3D object detection in point clouds. In: Proceedings of the IEEE/CVF International Conference on Computer Vision; 2019. p. 9277–86.
[84] Zhu L, Xie Z, Liu L, Tao B, Tao W. Iou-uniform r-cnn: breaking through the limitations of rpn. Pattern Recognit 2021;112:107816.
[85] Wang Y, Fathi A, Kundu A, Ross D, Pantofaru C, Funkhouser T, et al. Pillar-based object detection for autonomous driving. arXiv preprint arXiv:200710323 2020c.
[86] Shi S, Jiang L, Deng J, Wang Z, Guo C, Shi J, et al. Pv-rcnn++: point-voxel feature set abstraction with local vector representation for 3D object detection. arXiv preprint arXiv:210200463 2021.
[87] Bhattacharyya P, Huang C, Czarnecki K. Self-attention based context-aware 3D object detection. arXiv preprint arXiv:210102672 2021.
[88] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, et al. Attention is all you need. arXiv preprint arXiv:170603762 2017.
[89] Huang X, Cheng X, Geng Q, Cao B, Zhou D, Wang P, et al. The apolloscape dataset for autonomous driving. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops; 2018. p. 954–60.
[90] Kesten R., Usman M., Houston J., Pandya T., Nadhamuni K., Ferreira A., et al. Lyft level 5 perception dataset 2020. https://level5.lyft.com/dataset/; 2019.
[91] Patil A, Malla S, Gang H, Chen Y-T. The H3D dataset for full-surround 3D multi--object detection and tracking in crowded urban scenes. In: 2019 International Conference on Robotics and Automation (ICRA). IEEE; 2019. p. 9552–7.
[92] Geyer J, Kassahun Y, Mahmudi M, Ricou X, Durgesh R, Chung AS, et al. A2d2: Audi autonomous driving dataset. arXiv preprint arXiv:200406320 2020.
[93] Xie J, Kiefel M, Sun M-T, Geiger A. Semantic instance annotation of street scenes by 3D to 2D label transfer. In: Conference on Computer Vision and Pattern Recognition (CVPR); 2016.
[94] Simonelli A, Bulo SR, Porzi L, López-Antequera M, Kontschieder P. Disentangling monocular 3D object detection. In: Proceedings of the IEEE/CVF International Conference on Computer Vision; 2019. p. 1991–9.
[95] Faniadis E, Amanatiadis A. Deep learning inference at the edge for mobile and aerial robotics. In: 2020 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR). IEEE; 2020. p. 334–40.
[96] Zamanakos G, Seewald A, Midtiby HS, Schultz UP. Energy-aware design of vision-based autonomous tracking and landing of a uav. In: 2020 Fourth IEEE International Conference on Robotic Computing (IRC). IEEE; 2020. p. 294–7.
[97] Yuan W, Khot T, Held D, Mertz C, Hebert M. Pcn: Point completion network. In: 2018 International Conference on 3D Vision (3DV). IEEE; 2018. p. 728–37.
[98] Huang Z, Yu Y, Xu J, Ni F, Le X. Pf-net: Point fractal network for 3D point cloud completion. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition; 2020. p. 7662–70.