VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection

Yin Zhou
Apple Inc
yzhou3@apple.com

Oncel Tuzel
Apple Inc
otuzel@apple.com

Abstract

Accurate detection of objects in 3D point clouds is a central problem in many applications, such as autonomous navigation, housekeeping robots, and augmented/virtual reality. To interface a highly sparse LiDAR point cloud with a region proposal network (RPN), most existing efforts have focused on hand-crafted feature representations, for example, a bird's eye view projection. In this work, we remove the need of manual feature engineering for 3D point clouds and propose VoxelNet, a generic 3D detection network that unifies feature extraction and bounding box prediction into a single stage, end-to-end trainable deep network. Specifically, VoxelNet divides a point cloud into equally spaced 3D voxels and transforms a group of points within each voxel into a unified feature representation through the newly introduced voxel feature encoding (VFE) layer. In this way, the point cloud is encoded as a descriptive volumetric representation, which is then connected to a RPN to generate detections. Experiments on the KITTI car detection benchmark show that VoxelNet outperforms the state-of-the-art LiDAR based 3D detection methods by a large margin. Furthermore, our network learns an effective discriminative representation of objects with various geometries, leading to encouraging results in 3D detection of pedestrians and cyclists, based on only LiDAR.

1. Introduction

Point cloud based 3D object detection is an important component of a variety of real-world applications, such as autonomous navigation [11, 14], housekeeping robots [28], and augmented/virtual reality [29]. Compared to image-based detection, LiDAR provides reliable depth information that can be used to accurately localize objects and characterize their shapes [21, 5]. However, unlike images, LiDAR point clouds are sparse and have highly variable point density, due to factors such as non-uniform sampling of the 3D space, effective range of the sensors, occlusion, and the relative pose. To handle these challenges, many approaches manually crafted feature represen-
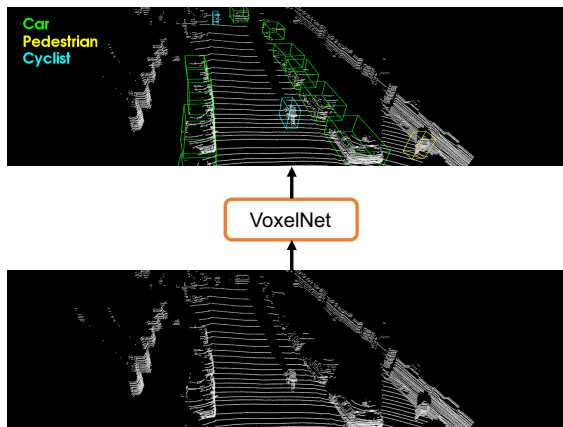

Figure 1. VoxelNet directly operates on the raw point cloud (no need for feature engineering) and produces the 3D detection results using a single end-to-end trainable network.

tations for point clouds that are tuned for 3D object detection. Several methods project point clouds into a perspective view and apply image-based feature extraction techniques [30, 15, 22]. Other approaches rasterize point clouds into a 3D voxel grid and encode each voxel with hand-crafted features [43, 9, 39, 40, 21, 5]. However, these manual design choices introduce an information bottleneck that prevents these approaches from effectively exploiting 3D shape information and the required invariances for the detection task. A major breakthrough in recognition [20] and detection [13] tasks on images was due to moving from hand-crafted features to machine-learned features.

Recently, Qi *et al.*[31] proposed PointNet, an end-to-end deep neural network that learns point-wise features directly from point clouds. This approach demonstrated impressive results on 3D object recognition, 3D object part segmentation, and point-wise semantic segmentation tasks. In [32], an improved version of PointNet was introduced which enabled the network to learn local structures at different scales. To achieve satisfactory results, these two approaches trained feature transformer networks on all input points (∼1k points). Since typical point clouds obtained using LiDARs contain ∼100k points, training the architec-
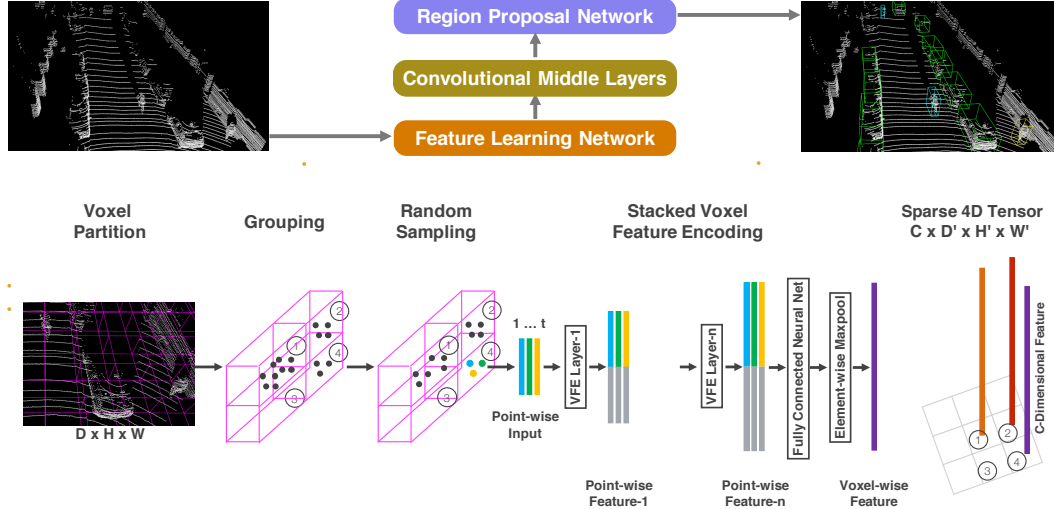
1

Figure 2. VoxelNet architecture. The feature learning network takes a raw point cloud as input, partitions the space into voxels, and transforms points within each voxel to a vector representation characterizing the shape information. The space is represented as a sparse 4D tensor. The convolutional middle layers processes the 4D tensor to aggregate spatial context. Finally, a RPN generates the 3D detection.

tures as in [31, 32] results in high computational and memory requirements. Scaling up 3D feature learning networks to orders of magnitude more points and to 3D detection tasks are the main challenges that we address in this paper.

Region proposal network (RPN) [34] is a highly optimized algorithm for efficient object detection [17, 5, 33, 24]. However, this approach requires data to be dense and organized in a tensor structure (e.g. image, video) which is not the case for typical LiDAR point clouds. In this paper, we close the gap between point set feature learning and RPN for 3D detection task.

We present VoxelNet, a generic 3D detection framework that simultaneously learns a discriminative feature representation from point clouds and predicts accurate 3D bounding boxes, in an end-to-end fashion, as shown in Figure 2. We design a novel voxel feature encoding (VFE) layer, which enables inter-point interaction within a voxel, by combining point-wise features with a locally aggregated feature. Stacking multiple VFE layers allows learning complex features for characterizing local 3D shape information. Specifically, VoxelNet divides the point cloud into equally spaced 3D voxels, encodes each voxel via stacked VFE layers, and then 3D convolution further aggregates local voxel features, transforming the point cloud into a high-dimensional volumetric representation. Finally, a RPN consumes the volumetric representation and yields the detection result. This efficient algorithm benefits both from the sparse point structure and efficient parallel processing on the voxel grid.

We evaluate VoxelNet on the bird's eye view detection and the full 3D detection tasks, provided by the KITTI benchmark [11]. Experimental results show that VoxelNet outperforms the state-of-the-art LiDAR based 3D detection

methods by a large margin. We also demonstrate that VoxelNet achieves highly encouraging results in detecting pedestrians and cyclists from LiDAR point cloud.

1.1. Related Work

Rapid development of 3D sensor technology has motivated researchers to develop efficient representations to detect and localize objects in point clouds. Some of the earlier methods for feature representation are [41, 8, 7, 19, 42, 35, 6, 27, 1, 36, 2, 25, 26]. These hand-crafted features yield satisfactory results when rich and detailed 3D shape information is available. However their inability to adapt to more complex shapes and scenes, and learn required invariances from data resulted in limited success for uncontrolled scenarios such as autonomous navigation.

Given that images provide detailed texture information, many algorithms inferred the 3D bounding boxes from 2D images [4, 3, 44, 45, 46, 38]. However, the accuracy of image-based 3D detection approaches are bounded by the accuracy of the depth estimation.

Several LIDAR based 3D object detection techniques utilize a voxel grid representation. [43, 9] encode each nonempty voxel with 6 statistical quantities that are derived from all the points contained within the voxel. [39] fuses multiple local statistics to represent each voxel. [40] computes the truncated signed distance on the voxel grid. [21] uses binary encoding for the 3D voxel grid. [5] introduces a multi-view representation for a LiDAR point cloud by computing a multi-channel feature map in the bird's eye view and the cylindral coordinates in the frontal view. Several other studies project point clouds onto a perspective view and then use image-based feature encoding

schemes [30, 15, 22].

There are also several multi-modal fusion methods that combine images and LiDAR to improve detection accuracy [10, 16, 5]. These methods provide improved performance compared to LiDAR-only 3D detection, particularly for small objects (pedestrians, cyclists) or when the objects are far, since cameras provide an order of magnitude more measurements than LiDAR. However the need for an additional camera that is time synchronized and calibrated with the LiDAR restricts their use and makes the solution more sensitive to sensor failure modes. In this work we focus on LiDAR-only detection.

1.2. Contributions

- We propose a novel end-to-end trainable deep architecture for point-cloud-based 3D detection, VoxelNet, that directly operates on sparse 3D points and avoids information bottlenecks introduced by manual feature engineering.

- We present an efficient method to implement VoxelNet which benefits both from the sparse point structure and efficient parallel processing on the voxel grid.

- We conduct experiments on KITTI benchmark and show that VoxelNet produces state-of-the-art results in LiDAR-based car, pedestrian, and cyclist detection benchmarks.

2. VoxelNet

In this section we explain the architecture of VoxelNet, the loss function used for training, and an efficient algorithm to implement the network.

2.1. VoxelNet Architecture

The proposed VoxelNet consists of three functional blocks: (1) Feature learning network, (2) Convolutional middle layers, and (3) Region proposal network [34], as illustrated in Figure 2. We provide a detailed introduction of VoxelNet in the following sections.

2.1.1 Feature Learning Network

Voxel Partition Given a point cloud, we subdivide the 3D space into equally spaced voxels as shown in Figure 2. Suppose the point cloud encompasses 3D space with range D, H, W along the Z, Y, X axes respectively. We define each voxel of size v_D, v_H, and v_W accordingly. The resulting 3D voxel grid is of size $D' = D/v_D, H' = H/v_H, W' = W/v_W$. Here, for simplicity, we assume D, H, W are a multiple of v_D, v_H, v_W.

Grouping We group the points according to the voxel they reside in. Due to factors such as distance, occlusion, object's relative pose, and non-uniform sampling, the LiDAR
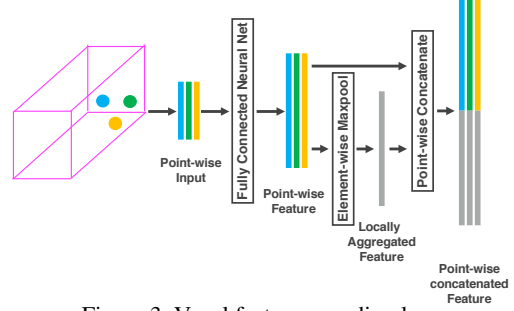

Figure 3. Voxel feature encoding layer.

point cloud is sparse and has highly variable point density throughout the space. Therefore, after grouping, a voxel will contain a variable number of points. An illustration is shown in Figure 2, where Voxel-1 has significantly more points than Voxel-2 and Voxel-4, while Voxel-3 contains no point.

Random Sampling Typically a high-definition LiDAR point cloud is composed of ∼100k points. Directly processing all the points not only imposes increased memory/efficiency burdens on the computing platform, but also highly variable point density throughout the space might bias the detection. To this end, we randomly sample a fixed number, T, of points from those voxels containing more than T points. This sampling strategy has two purposes, (1) computational savings (see Section 2.3 for details); and (2) decreases the imbalance of points between the voxels which reduces the sampling bias, and adds more variation to training.

Stacked Voxel Feature Encoding The key innovation is the chain of VFE layers. For simplicity, Figure 2 illustrates the hierarchical feature encoding process for one voxel. Without loss of generality, we use VFE Layer-1 to describe the details in the following paragraph. Figure 3 shows the architecture for VFE Layer-1.

Denote $\mathbf{V} = \{\mathbf{p}_i = [x_i, y_i, z_i, r_i]^T \in \mathbb{R}^4\}_{i=1...t}$ as a non-empty voxel containing $t \leq T$ LiDAR points, where $\mathbf{p}_i$ contains XYZ coordinates for the i-th point and r_i is the received reflectance. We first compute the local mean as the centroid of all the points in $\mathbf{V}$, denoted as (v_x, v_y, v_z). Then we augment each point $\mathbf{p}_i$ with the relative offset w.r.t. the centroid and obtain the input feature set $\mathbf{V}_{\text{in}} = \{\hat{\mathbf{p}}_i = [x_i, y_i, z_i, r_i, x_i - v_x, y_i - v_y, z_i - v_z]^T \in \mathbb{R}^7\}_{i=1...t}$. Next, each $\hat{\mathbf{p}}_i$ is transformed through the fully connected network (FCN) into a feature space, where we can aggregate information from the point features $\mathbf{f}_i \in \mathbb{R}^m$ to encode the shape of the surface contained within the voxel. The FCN is composed of a linear layer, a batch normalization (BN) layer, and a rectified linear unit (ReLU) layer. After obtaining point-wise feature representations, we use element-wise MaxPooling across all $\mathbf{f}_i$ associated to $\mathbf{V}$ to get the locally aggregated feature $\tilde{\mathbf{f}} \in \mathbb{R}^m$ for $\mathbf{V}$. Finally, we augment
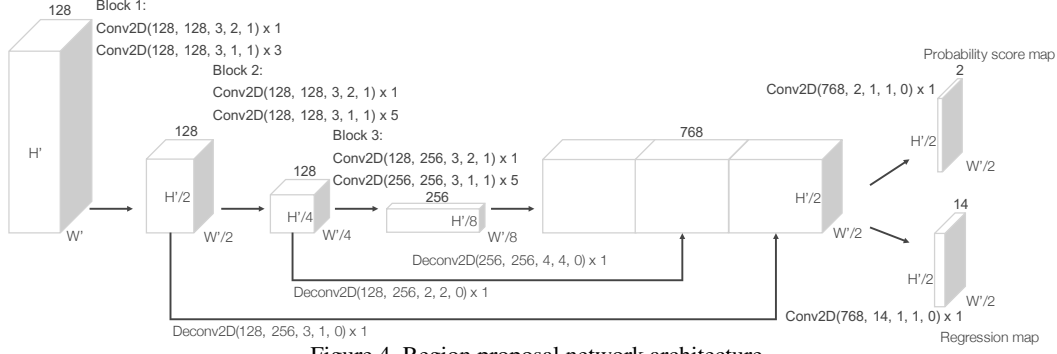
Figure 4. Region proposal network architecture.

each $\mathbf{f}_i$ with $\tilde{\mathbf{f}}$ to form the point-wise concatenated feature as $\mathbf{f}_i^{out} = [\mathbf{f}_i^T, \tilde{\mathbf{f}}^T]^T \in \mathbb{R}^{2m}$. Thus we obtain the output feature set $\mathbf{V}_{out} = \{\mathbf{f}_i^{out}\}_{i...t}$. All non-empty voxels are encoded in the same way and they share the same set of parameters in FCN.

We use VFE-i(c_{in}, c_{out}) to represent the i-th VFE layer that transforms input features of dimension c_{in} into output features of dimension c_{out}. The linear layer learns a matrix of size $c_{in} \times (c_{out}/2)$, and the point-wise concatenation yields the output of dimension c_{out}.

Because the output feature combines both point-wise features and locally aggregated feature, stacking VFE layers encodes point interactions within a voxel and enables the final feature representation to learn descriptive shape information. The voxel-wise feature is obtained by transforming the output of VFE-n into $\mathbb{R}^C$ via FCN and applying element-wise Maxpool where C is the dimension of the voxel-wise feature, as shown in Figure 2.

Sparse Tensor Representation By processing only the non-empty voxels, we obtain a list of voxel features, each uniquely associated to the spatial coordinates of a particular non-empty voxel. The obtained list of voxel-wise features can be represented as a sparse 4D tensor, of size $C \times D' \times H' \times W'$ as shown in Figure 2. Although the point cloud contains $\sim$100k points, more than 90% of voxels typically are empty. Representing non-empty voxel features as a sparse tensor greatly reduces the memory usage and computation cost during backpropagation, and it is a critical step in our efficient implementation.

2.1.2 Convolutional Middle Layers

We use ConvMD($c_{in}, c_{out}, \mathbf{k}, \mathbf{s}, \mathbf{p}$) to represent an M-dimensional convolution operator where c_{in} and c_{out} are the number of input and output channels, $\mathbf{k}$, $\mathbf{s}$, and $\mathbf{p}$ are the M-dimensional vectors corresponding to kernel size, stride size and padding size respectively. When the size across the M-dimensions are the same, we use a scalar to represent the size e.g. k for $\mathbf{k} = (k, k, k)$.

Each convolutional middle layer applies 3D convolution,

BN layer, and ReLU layer sequentially. The convolutional middle layers aggregate voxel-wise features within a progressively expanding receptive field, adding more context to the shape description. The detailed sizes of the filters in the convolutional middle layers are explained in Section 3.

2.1.3 Region Proposal Network

Recently, region proposal networks [34] have become an important building block of top-performing object detection frameworks [40, 5, 23]. In this work, we make several key modifications to the RPN architecture proposed in [34], and combine it with the feature learning network and convolutional middle layers to form an end-to-end trainable pipeline.

The input to our RPN is the feature map provided by the convolutional middle layers. The architecture of this network is illustrated in Figure 4. The network has three blocks of fully convolutional layers. The first layer of each block downsamples the feature map by half via a convolution with a stride size of 2, followed by a sequence of convolutions of stride 1 ($\times q$ means q applications of the filter). After each convolution layer, BN and ReLU operations are applied. We then upsample the output of every block to a fixed size and concatanate to construct the high resolution feature map. Finally, this feature map is mapped to the desired learning targets: (1) a probability score map and (2) a regression map.

2.2. Loss Function

Let $\{a_i^{\text{pos}}\}_{i=1...N_{\text{pos}}}$ be the set of N_{pos} positive anchors and $\{a_j^{\text{neg}}\}_{j=1...N_{\text{neg}}}$ be the set of N_{neg} negative anchors. We parameterize a 3D ground truth box as $(x_c^g, y_c^g, z_c^g, l^g, w^g, h^g, \theta^g)$, where x_c^g, y_c^g, z_c^g represent the center location, l^g, w^g, h^g are length, width, height of the box, and θ^g is the yaw rotation around Z-axis. To retrieve the ground truth box from a matching positive anchor parameterized as $(x_c^a, y_c^a, z_c^a, l^a, w^a, h^a, \theta^a)$, we define the residual vector $\mathbf{u}^* \in \mathbb{R}^7$ containing the 7 regression targets corresponding to center location $\Delta x, \Delta y, \Delta z$, three di-
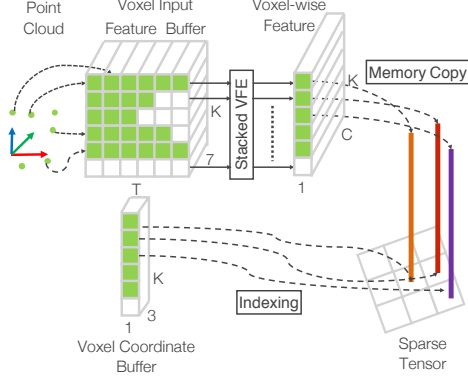
Figure 5. Illustration of efficient implementation.

mensions $\Delta l, \Delta w, \Delta h$, and the rotation $\Delta \theta$, which are computed as:

$$\Delta x = \frac{x_c^g - x_c^a}{d^a}, \Delta y = \frac{y_c^g - y_c^a}{d^a}, \Delta z = \frac{z_c^g - z_c^a}{h^a},$$
$$\Delta l = \log(\frac{l^g}{l^a}), \Delta w = \log(\frac{w^g}{w^a}), \Delta h = \log(\frac{h^g}{h^a}), \quad (1)$$
$$\Delta \theta = \theta^g - \theta^a$$

where $d^a = \sqrt{(l^a)^2 + (w^a)^2}$ is the diagonal of the base of the anchor box. Here, we aim to directly estimate the oriented 3D box and normalize Δx and Δy homogeneously with the diagonal d^a, which is different from [34, 40, 22, 21, 4, 3, 5]. We define the loss function as follows:

$$
\begin{aligned}
L &= \alpha \frac{1}{N_{\text{pos}}} \sum_i L_{\text{cls}}(p_i^{\text{pos}}, 1) + \beta \frac{1}{N_{\text{neg}}} \sum_j L_{\text{cls}}(p_j^{\text{neg}}, 0) \\
&+ \frac{1}{N_{\text{pos}}} \sum_i L_{\text{reg}}(\mathbf{u}_i, \mathbf{u}_i^*)
\end{aligned}
\quad (2)
$$

where p_i^{pos} and p_j^{neg} represent the softmax output for positive anchor a_i^{pos} and negative anchor a_j^{neg} respectively, while $\mathbf{u}_i \in \mathbb{R}^7$ and $\mathbf{u}_i^* \in \mathbb{R}^7$ are the regression output and ground truth for positive anchor a_i^{pos}. The first two terms are the normalized classification loss for $\{a_i^{\text{pos}}\}_{i=1...N_{\text{pos}}}$ and $\{a_j^{\text{neg}}\}_{j=1...N_{\text{neg}}}$, where the L_{cls} stands for binary cross entropy loss and α, β are postive constants balancing the relative importance. The last term L_{reg} is the regression loss, where we use the SmoothL1 function [12, 34].

2.3. Efficient Implementation

GPUs are optimized for processing dense tensor structures. The problem with working directly with the point cloud is that the points are sparsely distributed across space and each voxel has a variable number of points. We devised a method that converts the point cloud into a dense tensor structure where stacked VFE operations can be processed in parallel across points and voxels.

The method is summarized in Figure 5. We initialize a $K \times T \times 7$ dimensional tensor structure to store the voxel input feature buffer where K is the maximum number of non-empty voxels, T is the maximum number of points per voxel, and 7 is the input encoding dimension for each point. The points are randomized before processing. For each point in the point cloud, we check if the corresponding voxel already exists. This lookup operation is done efficiently in $O(1)$ using a hash table where the voxel coordinate is used as the hash key. If the voxel is already initialized we insert the point to the voxel location if there are less than T points, otherwise the point is ignored. If the voxel is not initialized, we initialize a new voxel, store its coordinate in the voxel coordinate buffer, and insert the point to this voxel location. The voxel input feature and coordinate buffers can be constructed via a single pass over the point list, therefore its complexity is $O(n)$. To further improve the memory/compute efficiency it is possible to only store a limited number of voxels (K) and ignore points coming from voxels with few points.

After the voxel input buffer is constructed, the stacked VFE only involves point level and voxel level dense operations which can be computed on a GPU in parallel. Note that, after concatenation operations in VFE, we reset the features corresponding to empty points to zero such that they do not affect the computed voxel features. Finally, using the stored coordinate buffer we reorganize the computed sparse voxel-wise structures to the dense voxel grid. The following convolutional middle layers and RPN operations work on a dense voxel grid which can be efficiently implemented on a GPU.

3. Training Details

In this section, we explain the implementation details of the VoxelNet and the training procedure.

3.1. Network Details

Our experimental setup is based on the LiDAR specifications of the KITTI dataset [11].
Car Detection For this task, we consider point clouds within the range of $[-3, 1] \times [-40, 40] \times [0, 70.4]$ meters along Z, Y, X axis respectively. Points that are projected outside of image boundaries are removed [5]. We choose a voxel size of $v_D = 0.4, v_H = 0.2, v_W = 0.2$ meters, which leads to $D' = 10, H' = 400, W' = 352$. We set $T = 35$ as the maximum number of randomly sampled points in each non-empty voxel. We use two VFE layers VFE-1$(7, 32)$ and VFE-2$(32, 128)$. The final FCN maps VFE-2 output to $\mathbb{R}^{128}$. Thus our feature learning net generates a sparse tensor of shape $128 \times 10 \times 400 \times 352$. To aggregate voxel-wise features, we employ three convolution middle layers sequentially as Conv3D(128, 64, 3, (2,1,1), (1,1,1)), Conv3D(64, 64, 3, (1,1,1), (0,1,1)), and

Conv3D(64, 64, 3, (2,1,1), (1,1,1)), which yields a 4D tensor of size $64 \times 2 \times 400 \times 352$. After reshaping, the input to RPN is a feature map of size $128 \times 400 \times 352$, where the dimensions correspond to channel, height, and width of the 3D tensor. Figure 4 illustrates the detailed network architecture for this task. Unlike [5], we use only one anchor size, $l^a = 3.9, w^a = 1.6, h^a = 1.56$ meters, centered at $z_c^a = -1.0$ meters with two rotations, 0 and 90 degrees. Our anchor matching criteria is as follows: An anchor is considered as positive if it has the highest Intersection over Union (IoU) with a ground truth or its IoU with ground truth is above 0.6 (in bird's eye view). An anchor is considered as negative if the IoU between it and all ground truth boxes is less than 0.45. We treat anchors as don't care if they have $0.45 \leq \text{IoU} \leq 0.6$ with any ground truth. We set $\alpha = 1.5$ and $\beta = 1$ in Eqn. 2.

Pedestrian and Cyclist Detection The input range[1] is $[-3, 1] \times [-20, 20] \times [0, 48]$ meters along Z, Y, X axis respectively. We use the same voxel size as for car detection, which yields $D = 10, H = 200, W = 240$. We set $T = 45$ in order to obtain more LiDAR points for better capturing shape information. The feature learning network and convolutional middle layers are identical to the networks used in the car detection task. For the RPN, we make one modification to block 1 in Figure 4 by changing the stride size in the first 2D convolution from 2 to 1. This allows finer resolution in anchor matching, which is necessary for detecting pedestrians and cyclists. We use anchor size $l^a = 0.8, w^a = 0.6, h^a = 1.73$ meters centered at $z_c^a = -0.6$ meters with 0 and 90 degrees rotation for pedestrian detection and use anchor size $l^a = 1.76, w^a = 0.6, h^a = 1.73$ meters centered at $z_c^a = -0.6$ with 0 and 90 degrees rotation for cyclist detection. The specific anchor matching criteria is as follows: We assign an anchor as postive if it has the highest IoU with a ground truth, or its IoU with ground truth is above 0.5. An anchor is considered as negative if its IoU with every ground truth is less than 0.35. For anchors having $0.35 \leq \text{IoU} \leq 0.5$ with any ground truth, we treat them as don't care.

During training, we use stochastic gradient descent (SGD) with learning rate 0.01 for the first 150 epochs and decrease the learning rate to 0.001 for the last 10 epochs. We use a batchsize of 16 point clouds.

3.2. Data Augmentation

With less than 4000 training point clouds, training our network from scratch will inevitably suffer from overfitting. To reduce this issue, we introduce three different forms of data augmentation. The augmented training data are generated on-the-fly without the need to be stored on disk [20].

---

[1] Our empirical observation suggests that beyond this range, LiDAR returns from pedestrians and cyclists become very sparse and therefore detection results will be unreliable.

Define set $\mathbf{M} = \{\mathbf{p}_i = [x_i, y_i, z_i, r_i]^T \in \mathbb{R}^4\}_{i=1,...,N}$ as the whole point cloud, consisting of N points. We parameterize a 3D bouding box $\mathbf{b}_i$ as $(x_c, y_c, z_c, l, w, h, \theta)$, where x_c, y_c, z_c are center locations, l, w, h are length, width, height, and θ is the yaw rotation around Z-axis. We define $\Omega_i = \{\mathbf{p}|x \in [x_c - l/2, x_c + l/2], y \in [y_c - w/2, y_c + w/2], z \in [z_c - h/2, z_c + h/2], \mathbf{p} \in \mathbf{M}\}$ as the set containing all LiDAR points within $\mathbf{b}_i$, where $\mathbf{p} = [x, y, z, r]$ denotes a particular LiDAR point in the whole set $\mathbf{M}$.

The first form of data augmentation applies perturbation independently to each ground truth 3D bounding box together with those LiDAR points within the box. Specifically, around Z-axis we rotate $\mathbf{b}_i$ and the associated Ω_i with respect to (x_c, y_c, z_c) by a uniformally distributed random variable $\Delta\theta \in [-\pi/10, +\pi/10]$. Then we add a translation $(\Delta x, \Delta y, \Delta z)$ to the XYZ components of $\mathbf{b}_i$ and to each point in Ω_i, where $\Delta x, \Delta y, \Delta z$ are drawn independently from a Gaussian distribution with mean zero and standard deviation 1.0. To avoid physically impossible outcomes, we perform a collision test between any two boxes after the perturbation and revert to the original if a collision is detected. Since the perturbation is applied to each ground truth box and the associated LiDAR points independently, the network is able to learn from substantially more variations than from the original training data.

Secondly, we apply global scaling to all ground truth boxes $\mathbf{b}_i$ and to the whole point cloud $\mathbf{M}$. Specifically, we multiply the XYZ coordinates and the three dimensions of each $\mathbf{b}_i$, and the XYZ coordinates of all points in $\mathbf{M}$ with a random variable drawn from uniform distribution $[0.95, 1.05]$. Introducing global scale augmentation improves robustness of the network for detecting objects with various sizes and distances as shown in image-based classification [37, 18] and detection tasks [12, 17].

Finally, we apply global rotation to all ground truth boxes $\mathbf{b}_i$ and to the whole point cloud $\mathbf{M}$. The rotation is applied along Z-axis and around $(0, 0, 0)$. The global rotation offset is determined by sampling from uniform distribution $[-\pi/4, +\pi/4]$. By rotating the entire point cloud, we simulate the vehicle making a turn.

4. Experiments

We evaluate VoxelNet on the KITTI 3D object detection benchmark [11] which contains 7,481 training images/point clouds and 7,518 test images/point clouds, covering three categories: *Car*, *Pedestrian*, and *Cyclist*. For each class, detection outcomes are evaluated based on three difficulty levels: *easy*, *moderate*, and *hard*, which are determined according to the object size, occlusion state, and truncation level. Since the ground truth for the test set is not available and the access to the test server is limited, we conduct comprehensive evaluation using the protocol described in [4, 3, 5] and subdivide the training data into a training set

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

Table 1. Performance comparison in bird's eye view detection: average precision (in %) on KITTI validation set.

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

Table 2. Performance comparison in 3D detection: average precision (in %) on KITTI validation set.

and a validation set, which results in 3,712 data samples for training and 3,769 data samples for validation. The split avoids samples from the same sequence being included in both the training and the validation set [3]. Finally we also present the test results using the KITTI server.

For the *Car* category, we compare the proposed method with several top-performing algorithms, including image based approaches: Mono3D [3] and 3DOP [4]; LiDAR based approaches: VeloFCN [22] and 3D-FCN [21]; and a multi-modal approach MV [5]. Mono3D [3], 3DOP [4] and MV [5] use a pre-trained model for initialization whereas we train VoxelNet from scratch using only the LiDAR data provided in KITTI.

To analyze the importance of end-to-end learning, we implement a strong baseline that is derived from the VoxelNet architecture but uses hand-crafted features instead of the proposed feature learning network. We call this model the hand-crafted baseline (HC-baseline). HC-baseline uses the bird's eye view features described in [5] which are computed at 0.1m resolution. Different from [5], we increase the number of height channels from 4 to 16 to capture more detailed shape information– further increasing the number of height channels did not lead to performance improvement. We replace the convolutional middle layers of VoxelNet with similar size 2D convolutional layers, which are Conv2D(16, 32, 3, 1, 1), Conv2D(32, 64, 3, 2, 1), Conv2D(64, 128, 3, 1, 1). Finally RPN is identical in VoxelNet and HC-baseline. The total number of parameters in HC-baseline and VoxelNet are very similar. We train the HC-baseline using the same training procedure and data augmentation described in Section 3.

4.1. Evaluation on KITTI Validation Set

Metrics We follow the official KITTI evaluation protocol, where the IoU threshold is 0.7 for class *Car* and is 0.5 for class *Pedestrian* and *Cyclist*. The IoU threshold is the same for both bird's eye view and full 3D evaluation. We compare the methods using the average precision (AP) metric.

Evaluation in Bird's Eye View The evaluation result is presented in Table 1. VoxelNet consistently outperforms all the competing approaches across all three difficulty levels. HC-baseline also achieves satisfactory performance compared to the state-of-the-art [5], which shows that our base region proposal network (RPN) is effective. For *Pedestrian* and *Cyclist* detection tasks in bird's eye view, we compare the proposed VoxelNet with HC-baseline. VoxelNet yields substantially higher AP than the HC-baseline for these more challenging categories, which shows that end-to-end learning is essential for point-cloud based detection.

We would like to note that [21] reported 88.9%, 77.3%, and 72.7% for *easy*, *moderate*, and *hard* levels respectively, but these results are obtained based on a different split of 6,000 training frames and ∼1,500 validation frames, and they are not directly comparable with algorithms in Table 1. Therefore, we do not include these results in the table.

Evaluation in 3D Compared to the bird's eye view detection, which requires only accurate localization of objects in the 2D plane, 3D detection is a more challenging task as it requires finer localization of shapes in 3D space. Table 2 summarizes the comparison. For the class *Car*, VoxelNet significantly outperforms all other approaches in AP across all difficulty levels. Specifically, using only LiDAR, VoxelNet significantly outperforms the state-of-the-art method MV (BV+FV+RGB) [5] based on

<div align="center">Car   Pedestrian   Cyclist</div>

Figure 6. Qualitative results. For better visualization 3D boxes detected using LiDAR are projected on to the RGB images.

LiDAR+RGB, by 10.68%, 2.78% and 6.29% in *easy*, *moderate*, and *hard* levels respectively. HC-baseline achieves similar accuracy to the MV [5] method.

As in the bird's eye view evaluation, we also compare VoxelNet with HC-baseline on 3D *Pedestrian* and *Cyclist* detection. Due to the high variation in 3D poses and shapes, successful detection of these two categories requires better 3D shape representation. As shown in Table 2 the improved performance of VoxelNet is emphasized for more challenging 3D detection tasks (from ~8% improvement in bird's eye view to ~12% improvement on 3D detection) which suggests that VoxelNet is more effective in capturing 3D shape information than hand-crafted features.

4.2. Evaluation on KITTI Test Set

We evaluated VoxelNet on the KITTI test set by submitting detection results to the official server. The results are summarized in Table 3. VoxelNet, significantly outperforms the previously published state-of-the-art [5] in all the tasks (bird's eye view and 3D detection) and all difficulties. We would like to note that many of the other leading methods listed in KITTI benchmark use both RGB images and LiDAR point clouds whereas VoxelNet uses only LiDAR.

We present several 3D detection examples in Figure 6. For better visualization 3D boxes detected using LiDAR are projected on to the RGB images. As shown, VoxelNet provides highly accurate 3D bounding boxes in all categories.

The inference time for VoxelNet is 33ms: the voxel input feature computation takes 5ms, feature learning network takes 16ms, convolutional middle layers take 1ms, and region proposal network takes 11ms on a TitanX GPU and

Benchmark	Easy	Moderate	Hard
Car (3D Detection)	77.47	65.11	57.73
Car (Bird's Eye View)	89.35	79.26	77.39
Pedestrian (3D Detection)	39.48	33.69	31.51
Pedestrian (Bird's Eye View)	46.13	40.74	38.11
Cyclist (3D Detection)	61.22	48.36	44.37
Cyclist (Bird's Eye View)	66.70	54.76	50.55

Table 3. Performance evaluation on KITTI test set.

1.7Ghz CPU.

5. Conclusion

Most existing methods in LiDAR-based 3D detection rely on hand-crafted feature representations, for example, a bird's eye view projection. In this paper, we remove the bottleneck of manual feature engineering and propose VoxelNet, a novel end-to-end trainable deep architecture for point cloud based 3D detection. Our approach can operate directly on sparse 3D points and capture 3D shape information effectively. We also present an efficient implementation of VoxelNet that benefits from point cloud sparsity and parallel processing on a voxel grid. Our experiments on the KITTI car detection task show that VoxelNet outperforms state-of-the-art LiDAR based 3D detection methods by a large margin. On more challenging tasks, such as 3D detection of pedestrians and cyclists, VoxelNet also demonstrates encouraging results showing that it provides a better 3D representation. Future work includes extending VoxelNet for joint LiDAR and image based end-to-end 3D detection to further improve detection and localization accuracy.

References

[1] P. Bariya and K. Nishino. Scale-hierarchical 3d object recognition in cluttered scenes. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1657–1664, 2010. 2

[2] L. Bo, X. Ren, and D. Fox. Depth Kernel Descriptors for Object Recognition. In *IROS*, September 2011. 2

[3] X. Chen, K. Kundu, Z. Zhang, H. Ma, S. Fidler, and R. Urtasun. Monocular 3d object detection for autonomous driving. In *IEEE CVPR*, 2016. 2, 5, 6, 7

[4] X. Chen, K. Kundu, Y. Zhu, A. Berneshawi, H. Ma, S. Fidler, and R. Urtasun. 3d object proposals for accurate object class detection. In *NIPS*, 2015. 2, 5, 6, 7

[5] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia. Multi-view 3d object detection network for autonomous driving. In *IEEE CVPR*, 2017. 1, 2, 3, 4, 5, 6, 7, 8

[6] C. Choi, Y. Taguchi, O. Tuzel, M. Y. Liu, and S. Ramalingam. Voting-based pose estimation for robotic assembly using a 3d sensor. In *2012 IEEE International Conference on Robotics and Automation*, pages 1724–1731, 2012. 2

[7] C. S. Chua and R. Jarvis. Point signatures: A new representation for 3d object recognition. *International Journal of Computer Vision*, 25(1):63–85, Oct 1997. 2

[8] C. Dorai and A. K. Jain. Cosmos-a representation scheme for 3d free-form objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(10):1115–1130, 1997. 2

[9] M. Engelcke, D. Rao, D. Z. Wang, C. H. Tong, and I. Posner. Vote3deep: Fast object detection in 3d point clouds using efficient convolutional neural networks. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1355–1361, May 2017. 1, 2

[10] M. Enzweiler and D. M. Gavrila. A multilevel mixture-of-experts framework for pedestrian classification. *IEEE Transactions on Image Processing*, 20(10):2967–2979, Oct 2011. 3

[11] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012. 1, 2, 5, 6

[12] R. Girshick. Fast r-cnn. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, 2015. 5, 6

[13] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014. 1

[14] R. Gomez-Ojeda, J. Briales, and J. Gonzalez-Jimenez. Plsvo: Semi-direct monocular visual odometry by combining points and line segments. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4211–4216, Oct 2016. 1

[15] A. Gonzalez, G. Villalonga, J. Xu, D. Vazquez, J. Amores, and A. Lopez. Multiview random forest of local experts combining rgb and lidar data for pedestrian detection. In *IEEE Intelligent Vehicles Symposium (IV)*, 2015. 1, 3

[16] A. Gonzlez, D. Vzquez, A. M. Lpez, and J. Amores. Onboard object detection: Multicue, multimodal, and multiview random forest of local experts. *IEEE Transactions on Cybernetics*, 47(11):3980–3990, Nov 2017. 3

[17] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, June 2016. 2, 6

[18] A. G. Howard. Some improvements on deep convolutional neural network based image classification. *CoRR*, abs/1312.5402, 2013. 6

[19] A. E. Johnson and M. Hebert. Using spin images for efficient object recognition in cluttered 3d scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(5):433–449, 1999. 2

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. 1, 6

[21] B. Li. 3d fully convolutional network for vehicle detection in point cloud. In *IROS*, 2017. 1, 2, 5, 7

[22] B. Li, T. Zhang, and T. Xia. Vehicle detection from 3d lidar using fully convolutional network. In *Robotics: Science and Systems*, 2016. 1, 3, 5, 7

[23] T. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. *IEEE ICCV*, 2017. 4

[24] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *ECCV*, pages 21–37, 2016. 2

[25] D. Maturana and S. Scherer. 3D Convolutional Neural Networks for Landing Zone Detection from LiDAR. In *ICRA*, 2015. 2

[26] D. Maturana and S. Scherer. VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition. In *IROS*, 2015. 2

[27] A. Mian, M. Bennamoun, and R. Owens. On the repeatability and quality of keypoints for local feature-based 3d object retrieval from cluttered scenes. *International Journal of Computer Vision*, 89(2):348–361, Sep 2010. 2

[28] Y.-J. Oh and Y. Watanabe. Development of small robot for home floor cleaning. In *Proceedings of the 41st SICE Annual Conference. SICE 2002.*, volume 5, pages 3222–3223 vol.5, Aug 2002. 1

[29] Y. Park, V. Lepetit, and W. Woo. Multiple 3d object tracking for augmented reality. In *2008 7th IEEE/ACM International Symposium on Mixed and Augmented Reality*, pages 117–120, Sept 2008. 1

[30] C. Premebida, J. Carreira, J. Batista, and U. Nunes. Pedestrian detection combining RGB and dense LIDAR data. In *IROS*, pages 0–1. IEEE, Sep 2014. 1, 3

[31] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2017. 1, 2

[32] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *arXiv preprint arXiv:1706.02413*, 2017. 1, 2

[33] J. Redmon and A. Farhadi. YOLO9000: better, faster, stronger. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 2

[34] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems 28*, pages 91–99. 2015. 2, 3, 4, 5

[35] R. B. Rusu, N. Blodow, and M. Beetz. Fast point feature histograms (fpfh) for 3d registration. In *2009 IEEE International Conference on Robotics and Automation*, pages 3212–3217, 2009. 2

[36] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-time human pose recognition in parts from single depth images. In *CVPR 2011*, pages 1297–1304, 2011. 2

[37] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. 6

[38] S. Song and M. Chandraker. Joint sfm and detection cues for monocular 3d localization in road scenes. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3734–3742, June 2015. 2

[39] S. Song and J. Xiao. Sliding shapes for 3d object detection in depth images. In *European Conference on Computer Vision, Proceedings*, pages 634–651, Cham, 2014. Springer International Publishing. 1, 2

[40] S. Song and J. Xiao. Deep Sliding Shapes for amodal 3D object detection in RGB-D images. In *CVPR*, 2016. 1, 2, 4, 5

[41] F. Stein and G. Medioni. Structural indexing: efficient 3-d object recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):125–145, 1992. 2

[42] O. Tuzel, M.-Y. Liu, Y. Taguchi, and A. Raghunathan. Learning to rank 3d features. In *13th European Conference on Computer Vision, Proceedings, Part I*, pages 520–535, 2014. 2

[43] D. Z. Wang and I. Posner. Voting for voting in online point cloud object detection. In *Proceedings of Robotics: Science and Systems*, Rome, Italy, July 2015. 1, 2

[44] Y. Xiang, W. Choi, Y. Lin, and S. Savarese. Data-driven 3d voxel patterns for object category recognition. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*, 2015. 2

[45] M. Z. Zia, M. Stark, B. Schiele, and K. Schindler. Detailed 3d representations for object recognition and modeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(11):2608–2623, 2013. 2

[46] M. Z. Zia, M. Stark, and K. Schindler. Are cars just 3d boxes? jointly estimating the 3d shape of multiple objects. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3678–3685, June 2014. 2