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

Shanliang Yao

Shanliang.Yao19@student.xjtlu.edu.cn

Xi'an Jiaotong-Liverpool University

August 31, 2021

Table of Contents

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

Introduction

Background

Since typical point clouds obtained using LiDARs contain 100k points, training the architectures as in results in **high computational and memory requirements**.

VoxelNet

- a novel end-to-end trainable deep architecture for point-cloud-based 3D detection that directly operates on **sparse 3D points** and avoids information bottlenecks introduced by manual feature engineering.
- an efficient method to implement VoxelNet which benefits both from the sparse point structure and **efficient parallel processing** on the voxel grid.
- state-of-the-art results in LiDAR-based car, pedestrian, and cyclist detection benchmarks.

Architecture

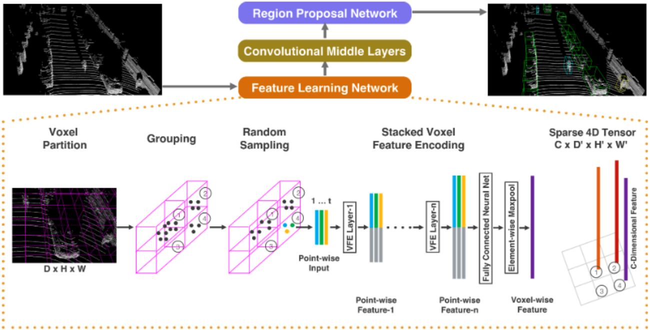
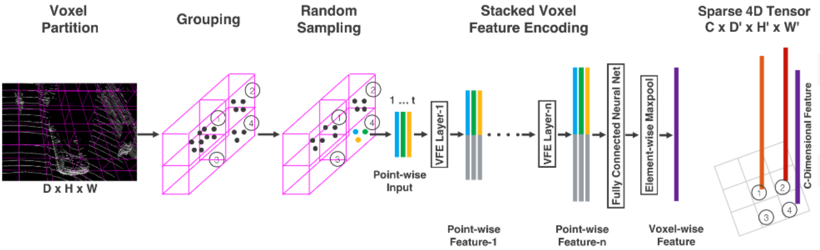


Figure 2. VoxelNet architecture. The feature learning network takes a raw point cloud as input, partitions the space into voxels, and transforms points within each voxel to a vector representation characterizing the shape information. The space is represented as a sparse 4D tensor. The convolutional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

Feature learning network



- **Voxel Partition:** subdivide the 3D space into equally spaced voxels
- **Grouping:** group the points according to the voxel they reside in
- **Random Sampling:** randomly sample $1 \sim T$ points
- **Stacked Voxel Feature Encoding:** Point-wise feature -> Voxel-wise Feature
- **Sparse Tensor Representation:** 4D tensor $C \times D' \times H' \times W'$

Region Proposal Network

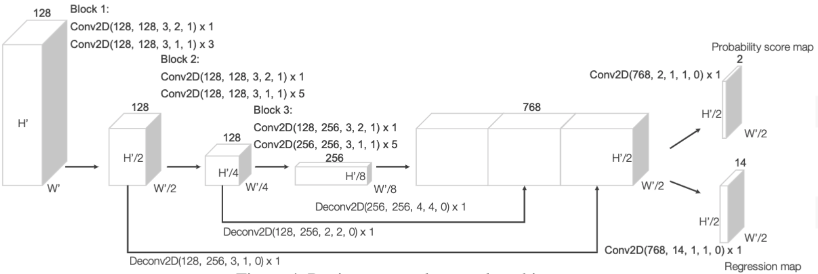


Figure 4. Region proposal network architecture.

- 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.
- After each convolution layer, **BN** and **ReLU** operations are applied.
- Then **upsample** the output of every block to a fixed size and **concatenate** 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**.

Efficient Implementation

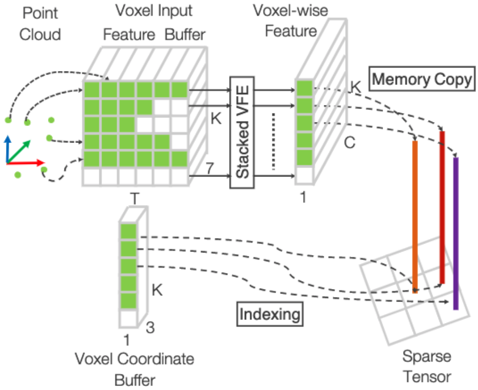


Figure 5. Illustration of efficient implementation.

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**.

- initialize a K × T × 7 dimensional tensor structure to store the voxel input feature buffer
- lookup operation is done efficiently in O(1) using a hash table where the voxel coordinate is used as the hash key
- the stacked VFE only involves point level and voxel level dense operations which can be computed on a GPU in parallel.

Loss Function

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

- $(x_c^g, y_c^g, z_c^g, l^g, w^g, h^g, \theta^g)$: 3D ground truth box
- $(x_c^a, y_c^a, z_c^a, l^a, w^a, h^a, \theta^a)$: a matching positive anchor
- $d^a = \sqrt{(l^a)^2 + (w^a)^2}$: the diagonal of the base of the anchor box
- $\{a_i^{\text{pos}}\}_{i=1\ldots N_{\text{pos}}}$: the set of N_{pos} positive anchors
- $\left\{a_j^{\text{neg}}\right\}_{j=1\ldots N_{\text{neg}}}$: the set of N_{neg} negative anchors
- $\mathbf{u}^* \in \mathbb{R}^7$: residual vector containing the 7 regression targets $\Delta x, \Delta y, \Delta z, \Delta l, \Delta w, \Delta h, \Delta\theta$

Loss Function

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

- p_i^{pos}: softmax output for positive anchor a_i^{pos}
- p_j^{neg}: softmax output for negative anchor a_j^{neg}
- L_{cls}: binary cross entropy loss
- $\mathbf{u}^* \in \mathbb{R}^7$: residual vector containing the 7 regression targets $\Delta x, \Delta y, \Delta z, \Delta l, \Delta w, \Delta h, \Delta\theta$
- $\mathbf{u}_i^* \in \mathbb{R}^7$: ground truth for positive anchor a_i^{pos}
- $\alpha = 1.5, \beta = 1$: positive constants balancing the relative importance
- L_{reg}: regression SmoothL1 loss function

Training

KITTI
Car Detection

- point clouds within $x \in [0, 70.4m], y \in [-40m, 40m], x \in [-3m, 1m]$
- a voxel size of $v_D = 0.4$, $v_H = 0.2$, $v_W = 0.2$ meters
- $D' = 10, H' = 400, W' = 352$
- $T = 35$
- anchor size $l = 3.9, w = 1.6, h = 1.56$

Pedestrian and Cyclist Detection

- point clouds within $x \in [0, 48m], y \in [-40, 40], x \in [-3m, 1m]$
- a voxel size of $v_D = 0.4$, $v_H = 0.2$, $v_W = 0.2$ meters
- $D' = 10, H' = 200, W' = 240$
- $T = 45$
- anchor size $l = 0.8, w = 0.6, h = 1.73$ for Pedestrian,
  $l = 1.76, w = 0.6, h = 1.73$ for Cyclist

Performance Comparison

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.

Performance Comparison

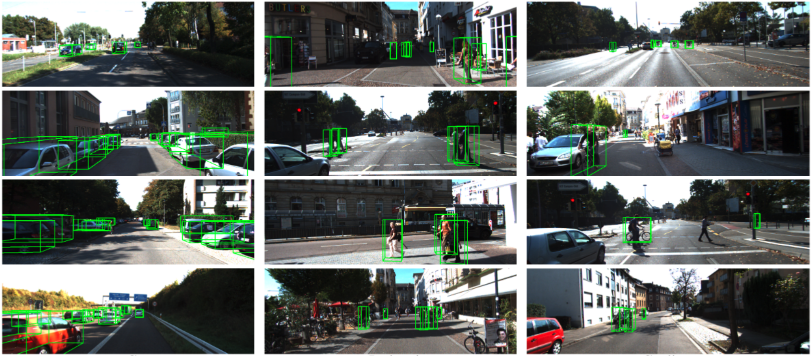
Benchmark	Easy	Moderate	Hard
Car (3D Detection)	77.47	65.11	57.73
Car (Bird's Eye View)	89.35	79.26	77.39
Pedestrian (3D Detection)	39.48	33.69	31.51
Pedestrian (Bird's Eye View)	46.13	40.74	38.11
Cyclist (3D Detection)	61.22	48.36	44.37
Cyclist (Bird's Eye View)	66.70	54.76	50.55

Table 3. Performance evaluation on KITTI test set.

Inference Time

- 225ms on a TitanX GPU and 1.7Ghz CPU
- input feature computation takes 5ms
- feature learning net takes 20ms
- convolutional middle layers take 170ms
- region proposal net takes 30ms

Visualization



Car Pedestrian Cyclist

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

Conclusion

Contributions

- remove the bottleneck of **manual feature** engineering and propose VoxelNet, a novel **end-to-end trainable** deep architecture for point cloud based 3D detection
- an efficient implementation of VoxelNet that benefits from **point cloud sparsity** and **parallel processing** on a voxel grid
- outperforms **state-of-the-art** LiDAR based 3D detection methods by a large margin

Future work

- extending VoxelNet for joint LiDAR and image based end-to-end 3D detection to further improve detection and localization accuracy.

Thank You !