The Visual Computer
https://doi.org/10.1007/s00371-022-02682-0

ORIGINAL ARTICLE

RepPVConv: attentively fusing reparameterized voxel features for efficient 3D point cloud perception

Keke Tang[1] · Yuhong Chen[1] · Weilong Peng[1] · Yanling Zhang[1] · Meie Fang[1] · Zheng Wang[2] · Peng Song[3]

Accepted: 18 September 2022

Abstract
Designing efficient deep learning models for 3D point clouds is an important research topic. Point-voxel convolution (Liu et al. in NeurIPS, 2019) is a pioneering approach in this direction, but it still has considerable room for improvement in terms of performance, since it has quite a few layers of simple 3D convolutions and linear point-voxel feature fusion operations. To resolve these issues, we propose a novel reparameterizable point-voxel convolution (RepPVConv) block. First, RepPVConv adopts two reparameterizable 3D convolution modules to extract more informative voxel features without introducing any extra computational overhead for inference. The rationale is that the reparameterizable 3D convolution modules are trained in high-capacity modes but are reparameterized into low-capacity modes during inference while losslessly maintaining the original performance. Second, RepPVConv attentively fuses the reparameterized voxel features with those of points. Since the proposed approach operates in a nonlinear manner, descriptive reparameterized voxel features can be better utilized. Extensive experimental results show that RepPVConv-based networks are efficient in terms of both GPU memory consumption and computational complexity and significantly outperform the state-of-the-art methods.

Keywords Reparameterization · PVCNN · Point clouds · Attention · Efficient

1 Introduction

With the popularization of depth sensing devices, 3D point cloud data is becoming ubiquitous in various applications of computer vision, computer graphics and robotics, e.g., object recognition [13,38] and robotic grasping [20]. In the last few years, researchers adopt deep learning models to handle these applications, significantly refreshing the state-of-the-art records. Although high accuracy is an important factor, considering the complexity and variability of real-world scenarios, enabling 3D point cloud perception models to be executed with low computational resources in a timely fashion is worthy of study.

Early 3D deep learning models transform the irregular point clouds into structured representations, e.g., rasterized 3D voxel grids [53], such that conventional convolutions could be easily adapted. However, their high performance usually demands high-resolution representations of voxel data to preserve the fine details in the given input data, which makes them expensive in terms of both computation and memory. Later, point-based methods, e.g., the pioneering PointNet [25] and PointCNN [19], attempt to directly exploit raw point clouds by processing each individual point using multi-layer perceptrons (MLPs) or designing complex kernel operations. However, locally aggregating the features of irregularly scattered points requires time-consuming operations, e.g., nearest-neighbor searching and dynamic kernel computation, hindering the efficiency of these methods.

Based on analyzing the bottlenecks of the above two research directions, point-voxel convolution (PVConv) [21] takes advantage of voxel-based convolution to obtain contiguous memory access patterns by a voxel-based branch and sparsity points to reduce the memory footprint by a point-based branch and then uses trilinear interpolation to fuse two

Keke Tang and Yuhong Chen contributed equally to this work.

✉ Weilong Peng
wlpeng@gzhu.edu.cn

✉ Meie Fang
fme@gzhu.edu.cn

[1] Guangzhou University, Guangzhou, China

[2] Southern University of Science and Technology, Shenzhen, China

[3] Singapore University of Technology and Design, Singapore, Singapore

Published online: 19 October 2022

branches of features. Although this method achieves large improvements in terms of speedup and memory reduction, its accuracy is somewhat sacrificed. Delving into the structure of PVConv, we find two factors that limit its performance. (1) The voxel-based branch uses just two layers of simple 3D convolutions, which cannot fully utilize the informative local features encoded in voxels. (2) Trilinear interpolation introduces information loss when transforming the features of voxel grids to points for feature fusion.

To address the first issue, a straightforward method is to adopt more powerful 3D convolution networks in the voxel branch of PVConv (e.g., with higher capacity), which brings more computational overhead in both the training and testing phases. Considering that efficiency in the testing phase is more critical for real applications since training can be conducted offline, we propose utilizing reparameterization technology to decrease the overhead of powerful 3D convolution networks for efficient inference while maintaining their performance. To address the second issue, we propose adopting nonlinear fusion methods instead of linear fusion methods to better utilize the obtained voxel features.

In this paper, we propose a novel reparameterizable point-voxel convolution (RepPVConv) block with two reparameterizable 3D convolution modules to fully utilize features in voxels and one attentive feature fusion module to reduce information loss during feature fusion. Specifically, the reparameterizable multi-branch module and dense-kernel module of RepPVConv are trained in high-capacity modes and then reparameterized into low-capacity modes for efficient inference, i.e., from multiple branches to a single branch and from dense kernels to normal kernels respectively, while losslessly maintaining the original performance. After that, the attentive feature fusion module of RepPVConv learns which parts of the voxel features are relevant to the point features and thus fuses the most relevant information in a nonlinear manner for better utilizing the voxel features. Therefore, RepPVConv can perceive 3D point clouds both accurately and effectively.

Overall, our contribution is threefold:

- We design a lightweight 3D convolution block, RepPVConv, that can perform 3D point cloud perception accurately and efficiently.
- We devise two reparameterizable 3D convolution modules, which are the first to introduce reparameterization technology for improving the efficiency of 3D network inference.
- We propose an attentive module that enforces the effective fusion of features in 3D point clouds and voxels in a nonlinear manner to achieve higher accuracy.

We evaluate the effectiveness and efficiency of RepPVCNN, which is constructed by stacking multiple RepPVConv blocks, by testing it on various 3D perception tasks, e.g., indoor scene segmentation, object part segmentation, 3D object classification and 3D object detection on multiple challenging datasets. Extensive results validate that RepPVConv is superior to the state-of-the-art methods in terms of both accuracy and efficiency.

2 Related work

2.1 Deep learning with point clouds

Deep learning models for 3D point cloud perception have been extensively investigated [3,14]. Voxel-based methods rasterize 3D point clouds into 3D voxel grids and then extend 2D deep learning methods to 3D. Wu et al. [47], Maturana et al [22] and Li [18] used binary voxel grids by categorizing each voxel as free space, a surface or an occluded area and then fed the voxels into 3D CNNs. To better utilize voxels, later researchers fused multiple statistics to represent each input voxel [10,33,34,53], obtaining large improvements. However, the major drawback of voxel-based methods, in which better performance usually demands higher resolutions, remains unsolved. Recently, the pioneering point-based model, i.e., PointNet [25], was proposed to handle 3D point clouds directly instead by processing each individual point using multi-layer perceptrons (MLPs). Later researchers exploited local aggregation schemes, including hierarchical structure [26], point-based convolutions [19,39,46,48], graph CNNs [5,32,36,44,51], and transformer-based networks [52], etc., achieving significantly improved performance. However, as indicated by [21], since point clouds are irregularly structured, which hinders neighborhood search and kernel computations, point-based methods are usually less efficient. Since deep learning for 3D point cloud perception is now a large and very active research field, for more complete reviews on existing research directions, we refer the readers to the survey papers [14,16].

2.2 Efficiency-aware 3D network design

Efficiency-aware 3D network design is an important research topic, especially for the applications executed on low-resource devices. However, most current voxel-based and point-based methods are not efficient enough (see Sect. 2.1). To resolve the inefficiency issue with respect to voxel-based methods, researchers have exploited the sparsity in input data and hierarchically partitioned the given space using a set of unbalanced octrees, where each leaf node stored a pooled feature representation [28,29,42] or a planar patch [43], thereby significantly reducing the memory allocation and computational complexity at high resolutions. Later, submanifold sparse convolutional networks were proposed to enable efficient volumetric convolution at high resolutions

by skipping non-activated regions [7,12]. As for point-based methods, RandLA-Net [15] adopted random key point sampling and increased receptive fields to accelerate, which, however, is more suitable for large-scale scenes. By analyzing the bottlenecks of both point-based and voxel-based deep learning models, a branches of researches [6,21,23,45] proposed combining the two types of methods into point-voxel convolutions as separate branches. Some recent work adopted not only point-voxel convolutions but also sparse convolution operations to further improve the efficiency of 3D networks [30,31,37,50].

We also adopt the point-voxel convolutions to achieve efficient 3D point cloud perception. Particularly, we aim to alleviate the inefficiency issue of voxel-based models in the voxel branch. Compared with octree and sparse convolution approaches that require additional operations, e.g., building octrees and hash tables, even during inference, our method is free of any extra operations.

2.3 Reparameterization technology

Reparameterization is a useful trick that can be adopted for many different purposes. A variational autoencoder (VAE) utilizes it to externalize the randomness in the latent code drawn from a Gaussian distribution, which allows the gradient to propagate back through the whole network [17]. DiracNet parameterizes weights as residuals of the Dirac function instead of adding explicit skip connections to help propagate information deeper in the network and alleviate the vanishing gradient problem [49]. RepVGG reparameterizes the VGG-style convolutional networks that have a multi-branch topology at training time, into single-branch models for efficient inference [9]. Inspired by this, we adopt reparameterization technology to improve the efficiency of 3D voxel-based models. To the best of our knowledge, reparameterization for 3D deep learning is understudied, especially for general-purpose 3D deep learning, and our work is the first attempt.

3 Methods

In this section, we first review PVConv, discuss some potential drawbacks that limit its performance (Sect. 3.1) and give our brief solution, i.e., RepPVConv (Sect. 3.2). Then, we describe the two novel reparameterizable 3D convolution modules (Sect. 3.3) and an attentive voxel-point fusion module (Sect. 3.4) in RepPVConv.

3.1 Limitations of PVConv [21]

Given an unordered point set $\boldsymbol{P} = \{(p_k, f_k)\}$ with $\{p_k\}$ denoting the point coordinates and $\{f_k\}$ as the point features, PVConv first normalizes the point coordinates and then conducts voxelization to transform the normalized point cloud $\{(\hat{p}_k, f_k)\}$ into a volume $\boldsymbol{V}$ with a resolution of $K \times K \times K$ by averaging all features $\{f_k\}$ whose normalized coordinates $\{\hat{p}_k\}$ fall into the voxel grid.

With $\boldsymbol{P}$ and $\boldsymbol{V}$, PVConv breaks the feature extraction process into two separate branches: a **point-based feature branch** to extract individual point features using MLPs (similar to PointNet), and a **voxel-based feature branch** to extract voxel features via a stack of 3D volumetric convolutions. After that, PVConv adopts a linear fusion approach that first devoxelizes the volume $\boldsymbol{V}$ back into the domain of the input point cloud by trilinear interpolation and then adds them together to fuse the voxel features with the point features. See the classic PVConv in the left bottom of Fig. 1.

Discussion As indicated in PVConv, the MLPs in the point-based feature branch can already output distinct and discriminative features for each point; therefore, their main contribution is to leverage the voxel-based feature branch for fusing neighborhood information.

Indeed, derived from mature 2D convolution techniques, 3D convolutions are very powerful. However, in consideration of efficiency, very limited volume resolutions and 3D convolution layers are adopted in PVConv, making the voxel features insufficiently discriminative. In addition, simple trilinear interpolation cannot guarantee that the voxel features are transformed into point features properly for fusion.

3.2 Overview of RepPVConv

To resolve the drawbacks of the classic PVConv approach, we propose a novel RepPVConv block with two key updates: reparameterizable 3D convolution modules for the voxel branch and an attentive voxel-point fusion module for the linear fusion module. Please refer to Fig. 1 for a demonstration.

3.3 Reparameterizable 3D convolution modules

To fully utilize voxel features while ensuring high efficiency, we propose using reparameterizable 3D convolution modules that enlarge the network capacity to better capture discriminative features during the training phase but can be reparameterized into low-capacity convolutions for efficient inference. In this section, we introduce the reparameterizable multi-branch 3D convolution module and the reparameterizable dense-kernel 3D convolution module, and apply them to the volume $\boldsymbol{V}$ voxelized from the given point clouds.

3.3.1 Reparameterizable multi-branch Module

Since a multi-branch topology behaves as an ensemble of numerous models and thus is beneficial for training [41],

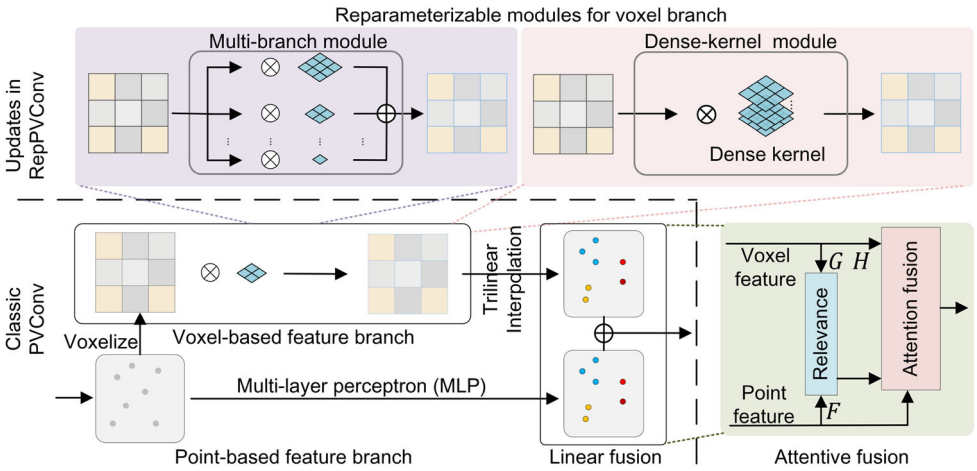

Fig. 1 Classic PVConv and the updates in our proposed RepPVConv block. Note that RepPVConv can utilize the multi-branch module and the dense-kernel module individually or together in the voxel-based feature branch

we propose a reparameterizable multi-branch module that enlarges the network capacity by adding more branches during training but can be reparameterized into a single branch for efficient inference.

Multi-branch training Denote $Conv3D(t, p, s)$ as a 3D convolution with a kernel size of $t \times t \times t$, a padding size of p and a stride of s, followed by a batch normalization (BN) layer. We use three branches of 3D convolutions: $Conv3D(1, 0, 1)$, $Conv3D(3, 1, 1)$ and $Conv3D(5, 2, 1)$, and then feed the summarization of their outputs to a nonlinear activation function. Specifically, we stack such modules with the same number as that of 3D convolutions in PVConv for fair comparisons.

Single-branch inferencing Although the multi-branch structure design can better extract voxel features, it significant increases the inference time. Therefore, we utilize reparameterization technology to fuse multiple 3D convolution branches into a single branch.

We first convert each convolution kernel W_t of size $t \times t \times t$ with its following BN layer into a new convolution kernel $W_t^{'}$ with a bias vector $b_t^{'}$:

$$W_t^{'} = \frac{W_t \gamma_t}{\sigma_t}, \quad t = 1, 3, 5, \tag{1}$$

$$b_t^{'} = \beta_t - \frac{\mu_t \gamma_t}{\sigma_t}, \quad t = 1, 3, 5, \tag{2}$$

where μ_t, σ_t, γ_t, and β_t represent the accumulated mean, standard deviation, learned scaling factor and bias of the BN layer respectively, which are uniformly denoted as $\mathbf{\Phi}_t$. We then add three new convolution layers and bias vectors separately to obtain the final single convolution kernel $W^{''}$ and bias $b^{''}$:

$$W^{''} = \sum_{t=\{1,3,5\}} Pad(W_t^{'}, 5), \tag{3}$$

$$b^{''} = \sum_{t=\{1,3,5\}} b_t^{'}, \tag{4}$$

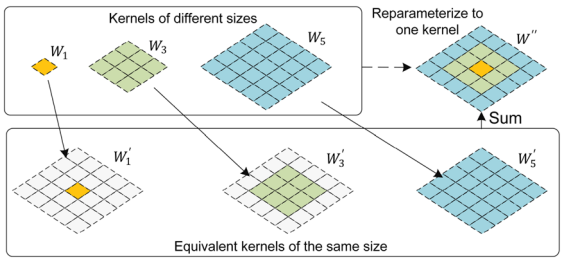

Fig. 2 Demonstration of the reparameterization of three 2D convolution kernels into a single kernel by converting them into equivalent kernels via applying zero padding and then conducting summarization. Note that the BN layers and bias vectors are not illustrated for clarity

where $Pad(W_t^{'}, 5)$ denotes the enlarged kernel obtained by applying zero padding to $W_t^{'}$ to make its size equal to $5 \times 5 \times 5$, if its original size is smaller than that. Please refer to Fig. 2 for a demonstration in 2D. Note that the equivalence of such transformations requires carefully designing the convolution kernels that need to be merged, e.g., providing them the same stride s and ensuring that they satisfy $t - 2p = 1$.

By applying the reparameterizable multi-branch module, multiple branches are occupied during network training, and only one branch with the largest kernel is used during inference.

3.3.2 Reparameterizable dense-kernel module

In this part, we describe the reparameterizable dense-kernel module that enlarges the network capacity by making kernel denser (i.e., with more parameters) for a fixed receptive field during training; this kernel can be reparameterized into a normal kernel for efficient inference.

Dense-kernel training Suppose that a normal kernel is of size $t \times t \times t$, and its dense version is of size $d \times t \times t \times t$ with a dense ratio of d. Therefore, for a fixed receptive field, denser kernels have d times the number of parameters

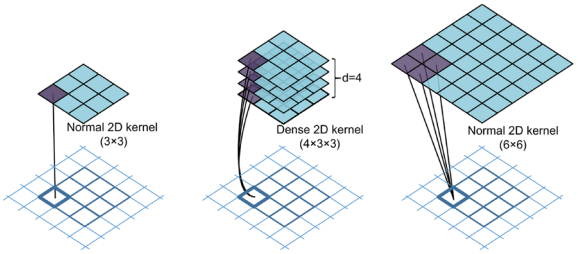

Fig. 3 Left: a normal 2D kernel; middle: a dense 2D kernel; right: an equivalent normal 2D kernel to the dense kernel

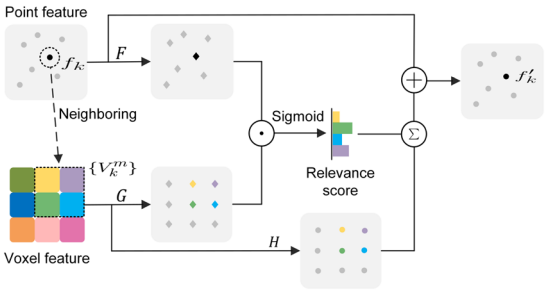

Fig. 4 Demonstration of the attentive voxel-point fusion module

that normal kernels have for capturing more discriminative features during training. By setting $d = u^3$, a 3D convolution with a dense kernel of size $d \times t \times t \times t$ over a volume $\boldsymbol{V}$ is equivalent to a convolution with an upsampled normal 3D kernel of size $ut \times ut \times ut$ over an upsampled volume $\boldsymbol{V^u}$. Here, $\boldsymbol{V}^u$ denotes the upsampled volume of $\boldsymbol{V}$ with a resolution of $uK \times uK \times uK$. To demonstrate a dense kernel more intuitively, we present a 2D case with $u = 2, d = u^2 = 4$ and $t = 3$ in Fig. 3. It can be seen that a dense 2D kernel of size $4 \times 3 \times 3$ has 4 times more parameters than a normal 2D kernel of size 3×3, and it is equivalent to a larger normal 2D kernel of size 6×6.

To facilitate easy training, we simply upsample the input volumes with the magnification ratio u by using nearest-neighbor interpolation, and we adopt normal kernels that are equivalent to the dense kernels by enlarging the strides and padding sizes with the same magnification ratio.

Normal-kernel inferencing Although the dense kernel structure design can better extract voxel features, it significantly increases the inference time. Therefore, we convert the learned kernels in the training phase to normal kernels by adopting reparameterization technology.

Formally, we use W_t^u to denote the learned kernel of size $ut \times ut \times ut$. By applying the average pooling operation $AvgPool_u$ with a pooling size and stride of u, we can obtain the following equation:

$$W_t^u \otimes \boldsymbol{V}^u = u^3 \cdot AvgPool_u(W_t^u) \otimes \boldsymbol{V}, \tag{5}$$

where $\cdot$ denotes the element-wise product and $\otimes$ denotes the convolution operation. Therefore, the final reparameterized kernel is as follows:

$$W_t^{u'} = u^3 \cdot AvgPool_u(W_t^u). \tag{6}$$

By applying the reparameterizable dense-kernel module, the upsampled normal kernels, i.e., dense kernels, are used during network training, and only smaller normal kernels are used during inference.

3.4 Attentive voxel-point fusion module

With the features of the voxel-based feature branch and point-based feature branch as inputs, the attentive voxel-point fusion module fuses them by first converting the voxel features into the point feature space and then summarizing them with the point features. Please refer to Fig. 4 for a demonstration.

For each normalized point represented by a coordinate and feature pair $(\hat{p}_k, f_k)$, we fuse the features of eight neighboring voxels $\{V_k^m\}$ with $m = 0...7$ (following PVCNN [21]) to it attentively. Specifically, we follow a similar attention process to that in [40]. First, we project the feature of each point f_k in addition to those of its eight neighboring voxels V_k^m into the same embedding feature space by using two separate networks F and G. Second, we calculate their relevance scores by calculating the dot product of the flattened features, followed by the Sigmoid operation:

$$Rel(f_k, V_k^m) = Sigmoid(F(f_k) \odot G(V_k^m)), \tag{7}$$

where $\odot$ denotes the operation of flattening into vectors and then conducting the dot product. Third, we convert the voxel features V_k^m into a summable space with the point features by feeding them into network H and then aggregate them with the relevance scores in a weighted manner,

$$f_k^{'} = f_k + \sum_m H(V_k^m) Rel(f_k, V_k^m). \tag{8}$$

Therefore, $f_k^{'}$ is the final fused feature that contains both point and voxel features. Compared with the trilinear interpolation procedure in PVConv, our module fuses the features in a more effective nonlinear manner.

4 Experiments

To validate the effectiveness and efficiency of RepPVConv, we construct RepPVCNN by stacking multiple RepPVConv blocks and evaluate its performance extensively on four

Table 1 Indoor scene segmentation performance on S3DIS with ablating the reparameterizable multi-branch module (Mul.), reparameterizable dense-kernel module (Den.) and attentive voxel-point fusion module (Att.)

Channel	Metric	Ours	w/o Mul	w/o Den	w/o Att
$0.125 \times C$	MIoU	**51.37**	50.08	50.01	49.83
	MAcc	84.13	84.17	83.06	**84.22**
$0.25 \times C$	MIoU	**55.88**	53.58	54.04	53.59
	MAcc	86.10	85.02	**86.26**	85.26
$0.5 \times C$	MIoU	**56.54**	54.26	56.38	55.73
	MAcc	**86.79**	85.15	86.35	86.43
$0.75 \times C$	MIoU	**56.81**	55.03	56.32	55.34
	MAcc	86.48	85.73	**86.60**	86.58
$1 \times C$	MIoU	**57.32**	55.40	56.47	56.74
	MAcc	**86.73**	86.29	86.66	86.54

Significant bold values are the best results for that sub-regions divided by lines

different tasks, i.e., indoor scene segmentation, object part segmentation, 3D object classification and detection.

4.1 Implementation of RepPVConv and RepPVCNN

For RepPVConv, we adopt three branches with $t = 1, 3, 5$ in the reparameterizable multi-branch module and a dense kernel with $u = 2$ in the reparameterizable dense-kernel module, and then summarize their outputs. Since these two modules can also be considered as two branches, we can further reparameterize them together. For RepPVCNN, we replace all the PVConv blocks in PVCNN with RepPVConv and use the same decoding layer. We implement RepPVCNN and reproduce all the evaluated networks with PyTorch [24] and measure the latency and memory consumption at test time on a workstation with an Intel Xeon E5-2678 CPU@2.5Hz and 64 GB of memory using a single RTX 2080Ti GPU. Since the process of k nearest-neighbor (KNN) search is recognized to be time-consuming, all the KNN operations in the compared networks are implemented using CUDA if not specially mentioned.

Table 3 Indoor scene segmentation performance on S3DIS when adding different configurations of the reparameterizable dense-kernel modules to the single-branch baseline with a kernel size of $3 \times 3 \times 3$ ($t = 3$)

Channel	Metric	$u = 1$	$u = 1, 2$	$u = 1, 3$
$0.25 \times C$	MIoU	52.68	**54.09**	52.93
	MAcc	85.49	**85.61**	85.36
$0.5 \times C$	MIoU	54.25	**55.28**	54.30
	MAcc	85.62	**85.87**	85.66

Significant bold values are the best results for that sub-regions divided by lines

4.2 Indoor scene segmentation

Setup We conduct indoor scene segmentation experiments on the large-scale Stanford 3D semantic parsing dataset (S3DIS) [1,2]. S3DIS contains 3D scans from Matterport scanners in 6 areas including 271 rooms, where each point in the scan is annotated with one of the semantic labels from 13 categories (chair, table, floor, wall, etc., plus clutter). We train the models on regions 1, 2, 3, 4, and 6 and then test them on region 5 to evaluate the mean intersection-over-union (mIoU) and the mean class-wise accuracy (mAcc) following [19,21]. Both the mIoU and mAcc are indicated as percentages (%). To facilitate fair comparisons, the data processing procedure is the same as that in [19,21].

Models We use PointNet [25], PointNet++ [26], PointCNN [19], DGCNN [44], PVCNN [21] and the latest RandLA-Net [15], KPConv [39] as our baselines, and report the reproduced results. For complexity measurement, we report the statistics of models with KNN implemented using both python and CUDA. Following [21], in addition to the original version (denoted as $1 \times C$), we also evaluate narrower versions of PVCNN and RepPVCNN by reducing the number of channels to 12.5% (denoted as $0.125 \times C$), 25% (denoted as $0.25 \times C$), 50% (denoted as $0.5 \times C$) and 75% (denoted as $0.75 \times C$) of the original number.

Ablation studies To validate the importance of the three key components in RepPVConv, i.e., the reparameterizable multi-branch module, reparameterizable dense-kernel module and attentive voxel-point fusion module, we evaluate the performance of RepPVCNN with ablating one component

Table 2 Indoor scene segmentation performance on S3DIS with different configurations of the reparameterizable multi-branch modules with a kernel size of $3 \times 3 \times 3$ ($t = 3$)

Channel	Metric	$t = 3$	$t = 1, 3$	$t = 1, 5$	$t = 3, 5$	$t = 1, 3, 5$
$0.25 \times C$	MIoU	52.68	53.57	54.77	53.42	**55.01**
	MAcc	85.49	85.66	85.97	85.19	**86.06**
$0.5 \times C$	MIoU	54.25	54.47	55.66	56.01	**56.38**
	MAcc	85.62	85.73	**86.35**	86.43	**86.35**

Significant bold values are the best results for that sub-regions divided by lines

Table 4 Indoor scene segmentation performance on S3DIS

	MAcc	MIoU	Latency (ms)	GPU mem. (GB)	#param.
PointNet	82.54	43.70	16.1	1.50	3.53M
PVCNN (0.125 × C)	82.60	46.94	7.6	0.46	43.16K
DGCNN	83.64	47.94	79.0	4.50	987.00K
Ours (0.125 × C)	84.13	**51.37**	7.8	0.51	45.27K
PVCNN (0.25 × C)	84.52	51.96	8.1	0.56	166.21K
PointNet++ (CUDA)	85.01	52.41	71.24	1.89	951.40K
PointNet++ (python)	–	–	590.02	2.63	–
PVCNN (0.5 × C)	85.88	54.73	14.5	0.73	650.35K
PVCNN	86.25	55.54	30.9	1.12	2.57M
Ours (0.25 × C)	86.10	**55.88**	12.3	0.64	171.98K
Ours (0.5 × C)	86.73	56.54	22.1	0.91	671.90K
PointCNN (CUDA)	85.91	57.26	252.3	3.02	1.95M
PointCNN (python)	–	–	459.3	3.62	–
RandLA-Net	85.09	57.30	304.1	2.57	4.76M
Ours (1 × C)	86.79	**57.32**	44.3	1.30	2.65M
KPConv†	–	64.19	143.4–244.2	4.23–9.82	12.06M

The input data of PointCNN include 16 × 2048 points, while the data of the other methods include 8 × 4096 points. In particular, KPConv† uses unspecific number of points with batch size 8 as input, and is evaluated on the unaligned version of S3DIS

Significant bold values are the best results for that sub-regions divided by lines

at a time. The results reported in Table 1 show that all the performance drops when any one component is omitted, validating their usefulness. Particularly, the reparameterizable multi-branch module affects the performance the most, e.g., the performance of Ours (1 × C) drops approximately 2% in terms of mIoU without this module, indicating that ensembling multiple branches is a good direction for improving the performance of 3D CNNs.

Parameter tuning results To choose the settings of the reparameterizable multi-branch module and reparameterizable dense-kernel module, we compare the performance with multiple different configurations. For the reparameterizable multi-branch module, we investigate four configurations: (a) $t = 1, 3$, (b) $t = 1, 5$, (c) $t = 3, 5$ and (d) $t = 1, 3, 5$. The results reported in Table 2 show that the configuration with $t = 1, 3, 5$ performs the best, validating the usefulness of adopting multiple branches. Since a larger t increases the final complexity of the model after reparameterization (see Sect. 3.3.1 again), we thus choose $t = 1, 3, 5$ in our experiments. For the dense-kernel module, we investigate two different configurations: (a) $u = 1, 2$ and (b) $u = 1, 3$ with $t = 3$. The results reported in Table 3 show that the configuration with $u = 1, 2$ improves the performance significantly, e.g., by approximately 1.5% for Ours (0.25 × C). However, kernels that are too dense are not easy to train, and the configuration with $u = 1, 3$ performs only comparably to the baseline. Therefore, we choose the configuration $u = 1, 2$ in our experiments.

Comparison results To demonstrate the superiority of RepPVCNN, we compare it and its narrow versions with the state-of-the-art methods, i.e., PointNet [25], PointNet++ [26], DGCNN [44], PointCNN [19], PVCNN [21] and RandLA-Net [15]. The results in Table 4 show that RepPVCNN improves upon its backbone, i.e., PointNet, by **14.4%** and the state-of-the-art point-based models, i.e., DGCNN, with **1.8**× lower latency and **3.5**× lower GPU memory consumption, and PointCNN, with nearly **6.5**× lower latency and **3.5**× lower GPU memory consumption. Since it is specifically designed for large-scale scenes, RandLA-Net performs much worse than RepPVCNN under the same setting in the room-scale scenes. For KPConv†, although with the best performance by using a flexible number of kernel points, its latency and consumed memory fluctuate all the time, and are much larger than Ours (1 × C) even for the minimal case. In addition, we would like to point out that Ours (0.25 × C) performs better than the full version of PVCNN (see the visual comparisons in Fig. 5), which is the state-of-the-art point-voxel fusion method, but with nearly **3**× lower latency, **2**× lower GPU memory consumption and **15**× fewer parameters, validating the efficiency of RepPVConv. As expected, PointNet++ and PointCNN that implement KNN using CUDA require much less latency than using python, but are still far

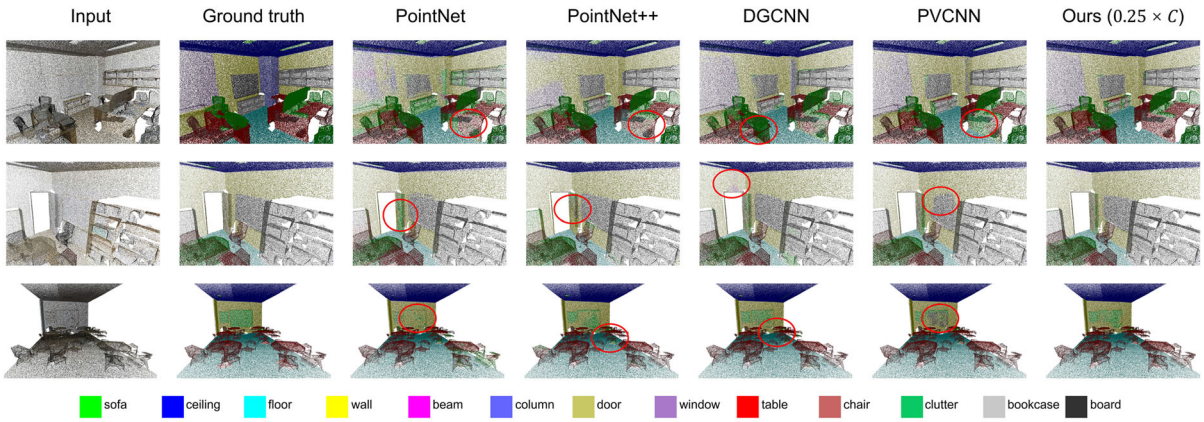

Fig. 5 Visualization of the semantic segmentation results on S3DIS

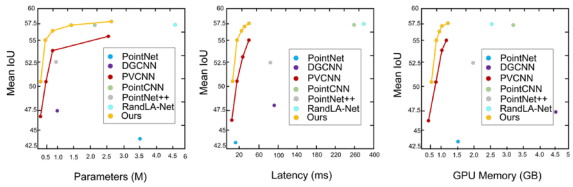

Fig. 6 The trade-off between accuracy and measured latency, GPU memory consumption and the number of parameters for RepPVCNN and the state-of-the-art baselines on S3DIS

Table 5 The overhead of our solution for training indoor scene segmentation on S3DIS

	Latency (ms)	GPU mem. (GB)
Ours (0.125 × C)	64.1	1.99
Ours (0.25 × C)	84.6	2.68
Ours (0.5 × C)	133.2	4.13
Ours (1 × C)	225.1	7.01

behind our method. Please refer to Fig. 6 to see an illustrative demonstration of the trade-off between accuracy (mIoU) and the incurred overhead (latency, GPU memory consumption and the number of parameters).

Overhead in the training phase We also report the overhead of RepPVCNN in the training phase in Table 5. Although the latency and consumed GPU memory in the training phase are much larger than those in the inference phase, the overhead is acceptable since the training can be conducted offline and it is affordable by most mainstream GPU servers.

4.3 Object part segmentation

Setup We conduct object part segmentation experiments that assign part category labels (e.g., chair legs and cup handles) to each point or face on the large-scale ShapeNet Parts dataset [4]. ShapeNet Parts is a collection of 16681 point clouds from 16 categories, each with 2–6 part labels from a total of 50 different parts. We follow the same 14006/2874 training/testing split as that in [19]. We evaluate the models by calculating the part-averaged IoU for each of the 2874 testing point clouds and then averaging the values as the final metrics, i.e., the mIoU (%).

Models We use point-based methods, e.g., PointNet [25], PointNet++ [26] and DGCNN [44], a voxel-based method, i.e., 3D-UNet [8], and PVCNN [21] as our baselines and compare them with RepPVCNN.

Comparison results The reported results in Table 6 show that Ours (0.25 × C) improves upon its backbone PointNet by more than **10%** in terms of the mIoU with nearly half the latency and 80% of the GPU memory consumption and outperforms the voxel-based 3D-UNet with nearly **40**× lower latency and **13**× less GPU memory consumption. In addition, Ours (0.5 × C) outperforms the full version of PVCNN, and Ours (1 × C) outperforms the state-of-the-art PointCNN with lower latency and less GPU memory. We also visualize the comparison results in Fig. 7. It could be seen that, since with fully exploiting voxel features, Ours can better utilize local information to avoid isolated error predictions.

4.4 3D object classification

Setup We conduct 3D object classification experiments on the ModelNet40 dataset [47]. ModelNet40 contains 12311 CAD models from 40 object categories, among which 9843 models are split for training and the remaining 2468 models are utilized for testing.

Models We compare three versions of RepPVCNN, i.e., Ours (0.25 × C), Ours (0.5 × C) and Ours (1 × C), with voxel-based methods, i.e., 3DShapeNets [47] and VoxelNet [53], one view-based method, i.e., MVCNN [35], point-based meth-

Table 6 Object part segmentation performance on ShapeNet Parts

	Input data	Conv	MIoU	Latency	GPU mem
PointNet	Points (8×2048)	–	83.70	16.0 ms	1.5 GB
3D-UNet	Voxels (8×96^3)	Voxel	84.60	480.0 ms	17.6 GB
PointNet++	Points (8×2048)	Point	84.70	55.6 ms	4.0 GB
DGCNN	Points (8×2048)	Point	84.70	61.8 ms	4.8 GB
PVCNN ($0.25 \times C$)	Points (8×2048)	Voxel	84.72	10.1 ms	1.2 GB
Ours ($0.25 \times C$)	Points (8×2048)	Voxel	**84.99**	12.0 ms	1.3 GB
PVCNN ($0.5 \times C$)	Points (8×2048)	Voxel	85.38	17.5 ms	1.8 GB
PVCNN	Points (8×2048)	voxel	85.70	35.2 ms	3.1 GB
Ours ($0.5 \times C$)	Points (8×2048)	Voxel	**85.73**	27.3 ms	2.3 GB
PointCNN	Points (8×2048)	Point	86.10	95.6 ms	5.1 GB
Ours ($1 \times C$)	Points (8×2048)	Voxel	**86.12**	60.1 ms	3.1 GB

Significant bold values are the best results for that sub-regions divided by lines

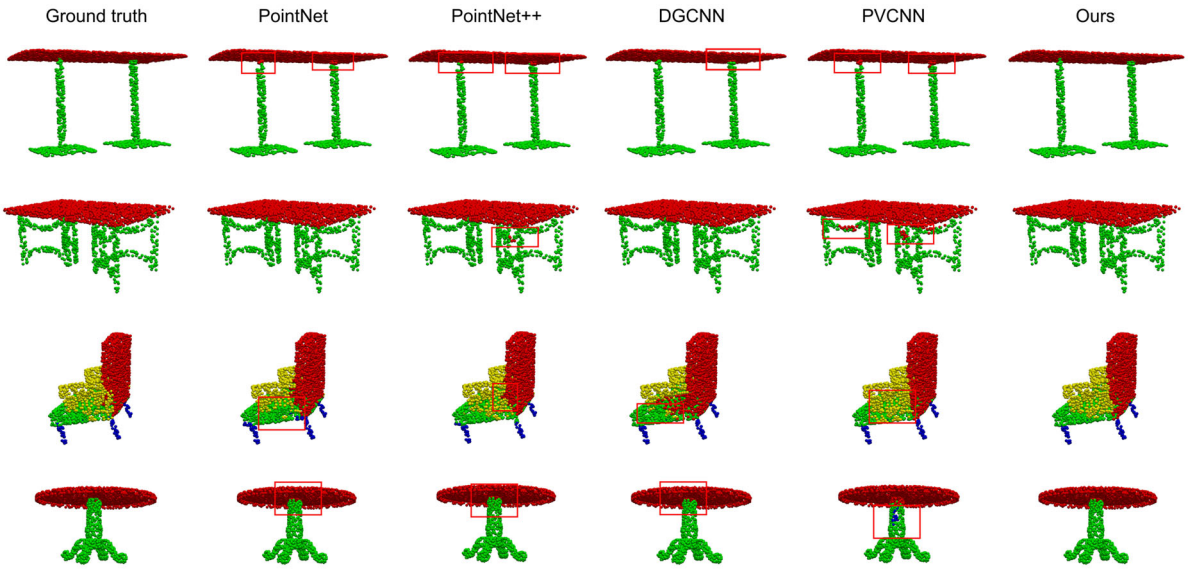

Fig. 7 Visualization of object part segmentation results on ShapeNet Parts

ods, i.e., PointNet [25] and PointCNN [19]and PVCNN [21] with three different configurations of channels.

Results The reported results in Table 7 show that Ours ($0.5 \times C$) is very efficient and performs comparably to the state-of-the-art methods, e.g., PVCNN and PointCNN. Particularly, Ours ($1 \times C$) performs the best with much less latency and GPU memory consumption, thereby validating the efficiency and effectiveness of RepPVConv.

4.5 3D Object detection

Setup We conduct 3D object detection experiments on the popular KITTI outdoor scene dataset [11]. The KITTI dataset provides 7481 training and validation samples for the cars, pedestrians and cyclists object classes, from which 3711 samples are utilized for training, leaving the other 3769 samples for validation (following [27]). We report the mean average precision (mAP) values (indicated as percentages (%)) yielded on the validation set.

Models We build Frustum Networks [27] with various backbone architectures, i.e., PointNet, PointNet++, PVCNN and Ours ($0.25 \times C$), following [21]. Particularly, Ours ($0.25 \times C$) is a narrow version of our proposed RepPVCNN by reducing the number of channels to 25% of the original number.

Results The reported results in Table 8 show that the Frustum Network with Ours ($0.25 \times C$) as the backbone outperforms all the other methods while incurring much less latency and GPU memory consumption, thereby validating the generalization performance of RepPVCNN on 3D object detection tasks.

Table 7 Classification performance on ModelNet40

	Input	Accuracy overall	Accuracy avg. class	Latency	GPU mem
VoxNet	Voxels	85.9	83.0	10.2 ms	1.85 GB
PointNet	Points	89.2	86.2	8.5 ms	1.16 GB
3DShapeNets	Voxels	90.1	–	31.1 ms	7.41 GB
PVCNN (0.25 × C)	Points	90.1	87.3	6.2 ms	0.98 GB
PointNet++	Points	90.7	–	85.1 ms	4.10 GB
Ours (0.25 × C)	Points	91.1	87.8	8.1 ms	1.03 GB
PVCNN (0.5 × C)	Points	91.2	88.3	12.4 ms	1.10 GB
PVCNN (1 × C)	Points	91.4	89.0	16.8 ms	1.72 GB
Ours (0.5 × C)	Points	91.6	89.2	18.9 ms	1.12 GB
PointCNN	Points	**92.2**	88.1	191 ms	2.68 GB
Ours (1 × C)	Points	92.1	**89.6**	21.4 ms	1.80 GB

Significant bold values are the best results for that sub-regions divided by lines

Table 8 3D object detection performance of the Frustum Networks [27] on the validation set of KITTI with different backbones

Backbone	Efficiency		Car			Pedestrian			Cyclist		
	Latency	GPU Mem	Easy	Mod	Hard	Easy	Mod	Hard	Easy	Mod	Hard
PointNet	29.1 ms	1.5 GB	83.26	69.28	62.56	65.08	55.85	49.28	74.54	55.95	52.65
PointNet++	101.2 ms	2.5 GB	83.76	70.92	63.65	70.00	61.32	53.59	77.15	56.49	53.37
PVCNN	51.9 ms	1.9 GB	**84.22**	71.11	63.63	69.16	60.28	52.52	**78.67**	57.79	54.16
Ours (0.25 × C)	**26.8 ms**	**1.1 GB**	84.00	**71.36**	**63.66**	**71.20**	**61.90**	**56.56**	**78.67**	**58.30**	**55.25**

Significant bold values are the best results for that sub-regions divided by lines

5 Conclusion

In this paper, we have presented a novel reparameterizable point-voxel convolution block (RepPVConv) with the aim of efficient 3D point cloud perception. The key of RepPVConv is to utilize reparameterization technology to facilitate training in high-capacity modes and evaluation in low-capacity modes. Extensive experiments validate the superiority of RepPVConv over the state-of-the-art methods. We hope RepPVConv can act as an important component of various network architectures for processing point clouds and inspire more research on efficient 3D deep learning.

Limitations and future work Our proposed reparameterizable modules yield larger improvements for smaller models, e.g., narrower versions with fewer channels. In the future, we plan to investigate more powerful reparameterization modules that are also suitable for large models.

Acknowledgements We thank the reviewers for the valuable comments. This work was supported in part by the National Natural Science Foundation of China (62102105, 62072126), Guangdong Basic and Applied Basic Research Foundation (2020A1515110997, 2022A1515011501, and 2022A1515010138), the Science and Technology Program of Guangzhou (202002030263, 202102010419 and 202201020229), and the Open Project Program of the State Key Lab of CAD and CG (A2218), Zhejiang University.

Declarations

Conflict of interest We declare that we have no competing financial interests or personal relationships that could have appeared to influence our work.

References

1. Armeni, I., Sener, O., Zamir, A.R. et al.: 3d semantic parsing of large-scale indoor spaces. In: CVPR, pp. 1534–1543 (2016)
2. Armeni, I., Sax, S., Zamir, A.R., et al.: Joint 2d-3d-semantic data for indoor scene understanding. arXiv preprint arXiv:1702.01105 (2017)
3. Bronstein, M.M., Bruna, J., LeCun, Y., et al.: Geometric deep learning: going beyond Euclidean data. IEEE Signal Process. Mag. **34**(4), 18–42 (2017)
4. Chang, A.X., Funkhouser, T., Guibas, L., et al.: Shapenet: an information-rich 3d model repository. arXiv preprint arXiv:1512.03012 (2015)
5. Chen, L., Zhang, Q.: Ddgcn: graph convolution network based on direction and distance for point cloud learning. Vis. Comput. 1–11 (2022) https://doi.org/10.1007/s00371-021-02351-8
6. Chen, Y., Peng, W., Tang, K., et al.: Pyrapvconv: efficient 3d point cloud perception with pyramid voxel convolution and sharable attention. Comput. Intell. Neurosci. **2022**:1–9 https://doi.org/10.1155/2022/2286818
7. Choy, C., Gwak, J., Savarese, S.: 4d spatio-temporal convnets: Minkowski convolutional neural networks. In: CVPR, pp. 3075–3084 (2019)

8. Çiçek, Ö., Abdulkadir, A., Lienkamp, S.S., et al.: 3d u-net: learning dense volumetric segmentation from sparse annotation. In: MICCAI, pp. 424–432 (2016)
9. Ding, X., Zhang, X., Ma, N., et al.: Repvgg: making vgg-style convnets great again. In: CVPR (2021)
10. Engelcke, M., Rao, D., Wang, D.Z., et al.: Vote3deep: fast object detection in 3d point clouds using efficient convolutional neural networks. In: ICRA, pp. 1355–1361. IEEE (2017)
11. Geiger, A., Lenz, P., Stiller, C., et al.: Vision meets robotics: the kitti dataset. IJRR **32**(11), 1231–1237 (2013)
12. Graham, B., Engelcke, M., Van Der Maaten, L.: 3d semantic segmentation with submanifold sparse convolutional networks. In: CVPR, pp. 9224–9232 (2018)
13. Guo, Y., Bennamoun, M., Sohel, F., et al.: 3d object recognition in cluttered scenes with local surface features: a survey. IEEE TPAMI **36**(11), 2270–2287 (2014)
14. Guo, Y., Wang, H., Hu, Q., et al.: Deep learning for 3d point clouds: a survey. IEEE Trans. Pattern Anal. Mach. Intell. **43**(12), 4338–4364 (2020)
15. Hu, Q., Yang, B., Xie, L., et al.: Randla-net: efficient semantic segmentation of large-scale point clouds. In: CVPR, pp. 11,108–11,117 (2020)
16. Ioannidou, A., Chatzilari, E., Nikolopoulos, S., et al.: Deep learning advances in computer vision with 3d data: a survey. ACM Comput. Surv. (CSUR) **50**(2), 1–38 (2017)
17. Kingma, D.P., Welling, M., et al.: An introduction to variational autoencoders. Found. Trends® Mach. Learn. **12**(4), 307–392 (2019)
18. Li, B.: 3d fully convolutional network for vehicle detection in point cloud. In: IROS, pp. 1513–1518 (2017)
19. Li, Y., Bu, R., Sun, M., et al.: Pointcnn: convolution on x-transformed points. In: NeurIPS, pp. 820–830 (2018)
20. Lin, N., Li, Y., Tang, K., et al.: Manipulation planning from demonstration via goal-conditioned prior action primitive decomposition and alignment. IEEE Robot. Autom. Lett. **7**(2), 1387–1394 (2022)
21. Liu, Z., Tang, H., Lin, Y., et al.: Point-voxel CNN for efficient 3d deep learning. In: NeurIPS (2019)
22. Maturana, D., Scherer, S.: Voxnet: a 3d convolutional neural network for real-time object recognition. In: IROS, pp. 922–928 (2015)
23. Noh, J., Lee, S., Ham, B.: Hvpr: hybrid voxel-point representation for single-stage 3d object detection. In: CVPR, pp. 14,605–14,614 (2021)
24. Paszke, A., Gross, S., Massa, F., et al.: Pytorch: an imperative style, high-performance deep learning library. In: NeurIPS, pp. 8026–8037 (2019)
25. Qi, C.R., Su, H., Mo, K., et al.: Pointnet: deep learning on point sets for 3d classification and segmentation. In: CVPR, pp. 652–660 (2017a)
26. Qi, C.R., Yi, L., Su, H., et al.: Pointnet++ deep hierarchical feature learning on point sets in a metric space. In: NeurIPS, pp. 5105–5114 (2017b)
27. Qi, C.R., Liu, W., Wu, C., et al. Frustum pointnets for 3d object detection from rgb-d data. In: CVPR, pp. 918–927 (2018)
28. Que, Z., Lu, G., Xu, D.: Voxelcontext-net: an octree based framework for point cloud compression. In: CVPR, pp. 6042–6051 (2021)
29. Riegler, G., Osman Ulusoy, A., Geiger, A.: Octnet: learning deep 3d representations at high resolutions. In: CVPR, pp. 3577–3586 (2017)
30. Shi, S., Guo, C., Jiang, L., et al.: PV-RCNN: point-voxel feature set abstraction for 3d object detection. In: CVPR, pp. 10,529–10,538 (2020)
31. Shi, S., Jiang, L., Deng, J., et al.: PV-RCNN++: point-voxel feature set abstraction with local vector representation for 3d object detection. arXiv preprint arXiv:2102.00463 (2021)
32. Shi, W., Rajkumar, R.: Point-GNN: graph neural network for 3d object detection in a point cloud. In: CVPR, pp. 1711–1719 (2020)
33. Song, S., Xiao, J.: Sliding shapes for 3d object detection in depth images. In: ECCV, pp. 634–651 (2014)
34. Song, S., Xiao, J.: Deep sliding shapes for amodal 3d object detection in rgb-d images. In: CVPR, pp. 808–816 (2016)
35. Su, H., Maji, S., Kalogerakis, E., et al.: Multi-view convolutional neural networks for 3d shape recognition. In: ICCV, pp. 945–953 (2015)
36. Sun, Y., Miao, Y., Chen, J., et al.: Pgcnet: patch graph convolutional network for point cloud segmentation of indoor scenes. Vis. Comput. **36**(10), 2407–2418 (2020)
37. Tang, H., Liu, Z., Zhao, S., et al.: Searching efficient 3d architectures with sparse point-voxel convolution. In: ECCV, pp. 685–702 (2020)
38. Tang, K., Ma, Y., Miao, D., et al.: Decision fusion networks for image classification. IEEE Trans. Neural Netw. Learn. Syst. (2022) https://doi.org/10.1109/TNNLS.2022.3196129
39. Thomas, H., Qi, C.R., Deschaud, J.E., et al.: Kpconv: flexible and deformable convolution for point clouds. In: ICCV, pp. 6411–6420 (2019)
40. Vaswani, A., Shazeer, N., Parmar, N., et al.: Attention is all you need. In: NeurIPS (2017)
41. Veit, A., Wilber, M., Belongie, S.: Residual networks behave like ensembles of relatively shallow networks. In: NeurIPS, pp. 550–558 (2016)
42. Wang, P.S., Liu, Y., Guo, Y.X., et al.: O-cnn: Octree-based convolutional neural networks for 3d shape analysis. ACM TOG **36**(4), 1–11 (2017)
43. Wang, P.S., Sun, C.Y., Liu, Y., et al.: Adaptive o-cnn: a patch-based deep representation of 3d shapes. ACM TOG **37**(6), 1–11 (2018)
44. Wang, Y., Sun, Y., Liu, Z., et al.: Dynamic graph cnn for learning on point clouds. ACM TOG (SIGGRAPH) **38**(5), 1–12 (2019)
45. Wei, Y., Wang, Z., Rao, Y., et al.: Pv-raft: point-voxel correlation fields for scene flow estimation of point clouds. In: CVPR, pp. 6954–6963 (2021)
46. Wu, W., Qi, Z., Fuxin, L.: Pointconv: deep convolutional networks on 3d point clouds. In: CVPR, pp. 9621–9630 (2019)
47. Wu, Z., Song, S., Khosla, A., et al.: 3d shapenets: a deep representation for volumetric shapes. In: CVPR, pp. 1912–1920 (2015)
48. Xu, M., Ding, R., Zhao, H., et al.: Paconv: position adaptive convolution with dynamic kernel assembling on point clouds. In: CVPR (2021)
49. Zagoruyko, S., Komodakis, N.: Diracnets: Training very deep neural networks without skip-connections. arXiv preprint arXiv:1706.00388 (2017)
50. Zhang, F., Fang, J., Wah, B.W., et al.: Deep fusionnet for point cloud semantic segmentation. In: ECCV, pp. 644–663 (2020)
51. Zhao, H., Jiang, L., Fu, C.W., et al.: Pointweb: enhancing local neighborhood features for point cloud processing. In: CVPR, pp. 5565–5573 (2019)
52. Zhao, H., Jiang, L., Jia, J., et al.: Point transformer. In: ICCV, pp. 16,259–16,268 (2021)
53. Zhou, Y., Tuzel, O.: Voxelnet: end-to-end learning for point cloud based 3d object detection. In: CVPR, pp. 4490–4499 (2018)

Keke Tang received the B.Eng. degree from Jilin University, Changchun, China, in 2012 and the Ph.D. degree from the University of Science and Technology of China, Hefei, China, in 2017. He was a Post-Doctoral Fellow with The University of Hong Kong, Hong Kong, China. In 2019, he joined Guangzhou University, Guangzhou, China, where he is currently an Associate Professor. His research interests include the areas of robotics, computer vision, computer graphics, and cyberspace security.

Yuhong Chen is currently pursuing his master degree in School of Computer Science and Cyber Engineering, Guangzhou University. His research interests include computer vision and computer graphics.

Weilong Peng received the Ph.D. degree in computer application technology from Tianjin University, Tianjin, China, in 2017. He is currently a Lecturer with Guangzhou University. His research interests lie in artificial intelligence, computer vision.

Yanling Zhang received her Ph.D. degree from Guangdong University of Technology. She is currently an Associate Professor at Guangzhou University. Her research interests lie in artificial intelligence, computer vision.

Meie Fang is a Professor of School of Computer Science and Cyber Engineering at Guangzhou University. She earned her Ph.D. in Applied Mathematics from Zhejiang University and her BS and MS in Mathematics from Hunan Normal University. Her research interest is in intelligent computer graphics, geometric deep learning, and medical image analysis.

Zheng Wang received the B.Sc. degree (Hons.) in automatic control from Tsinghua University, Beijing, China, in 2004, the M.Sc. degree (Hons.) in control systems from the Imperial College London, London, UK, in 2005, and the Ph.D. degree (Hons.) in electrical engineering from Technische Universitaet Muenchen, Munich, Germany, in 2010. He is currently a Professor of robotics with the Department of Mechanical and Energy Engineering, Southern University of Science and Technology. His research interests include haptics human-robot interaction, endoscopic surgical robot, underwater robots, and soft robotics.

Peng Song received the Ph.D. degree from Nanyang Technological University, Singapore, in 2013. He was a Research Scientist with the Swiss Federal Institute of Technology Lausanne, Lausanne, Switzerland. In 2019, he joined the Singapore University of Technology and Design, Singapore, where he is currently an Assistant Professor of computer science. His research interest includes computer graphics, with an emphasis on geometry modeling, computational designs, and digital fabrication. He received SIGGRAPH Technical Papers Honorable Mention Award in 2022. He has served as a co-organizer of a virtual seminar series on computational fabrication in 2021 and 2022. He was invited to give a keynote talk at the Symposium on Solid and Physical Modeling 2022.