PSNet: Fast Data Structuring for Hierarchical Deep Learning on Point Cloud

Luyang Li, Ligang He, *Member, IEEE,* Jinjin Gao and Xie Han

***Abstract*—In order to retain more feature information of local areas on a point cloud, local grouping and subsampling are the necessary data structuring steps in most hierarchical deep learning models. Due to the disorder nature of the points in a point cloud, the significant time cost may be consumed when grouping and subsampling the points, which consequently results in poor scalability. This paper proposes a fast data structuring method called PSNet (Point Structuring Net). PSNet transforms the spatial features of the points and matches them to the features of local areas in a point cloud. PSNet achieves grouping and sampling at the same time while the existing methods process sampling and grouping in two separate steps (such as using FPS plus kNN). PSNet performs feature transformation pointwise while the existing methods uses the spatial relationship among the points as the reference for grouping. Thanks to these features, PSNet has two important advantages: 1) the grouping and sampling results obtained by PSNet is stable and permutation invariant; and 2) PSNet can be easily parallelized. PSNet can replace the data structuring methods in the mainstream point cloud deep learning models in a plug-and-play manner. We have conducted extensive experiments. The results show that PSNet can improve the training and reasoning speed significantly while maintaining the model accuracy.**

***Index Terms*—Deep learning, point cloud, data structuring, computer vision, grouping, sampling.**

I. Introduction

POINT cloud is a common 3D data format. It has been widely used in areas such as robotics and autonomous driving. Since a point cloud has a non-Euclidean data structure [1], it is still facing significant challenges to apply deep learning methods directly to point clouds and extracting effective information.

Following on the great success of CNN in images [2]–[7], the voxel model [8]–[16] extends the 2D CNN and is applied directly to the regularized point cloud data. However, the voxel model loses the resolution of the point cloud, and causes the significant increase in computation cost.

Manuscript received December 13, 2021; revised March 15, 2022; accepted April 26, 2022. (Corresponding author: Xie Han.)

Luyang Li is with School of Data Science and Technology, North University of China, Taiyuan 030051, China, and also with Shanxi Information Industry Technology Research Institute Co., Ltd., Taiyuan 030012, China (e-mail: lly007@live.cn).

Ligang He is with Department of Computer at the University of Warwick, Coventry, CV4 7AL, United Kingdom (e-mail: ligang.he@warwick.ac.uk).

Jinjin Gao is with experimental center, Shanxi University of Finance and Economics, Taiyuan 030006, China (e-mail: 20141005@sxufe.edu.cn).

Xie Han is with School of Data Science and Technology, North University of China, Taiyuan 030051, China (e-mail: hanxie@nuc.edu.cn).

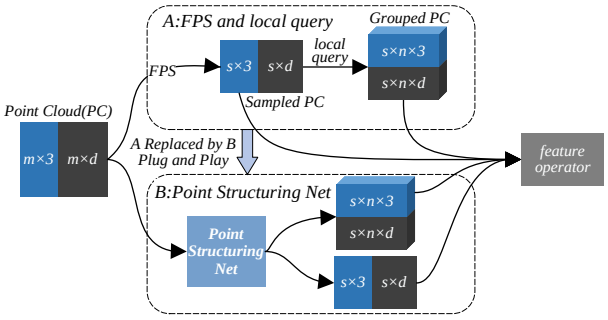


Fig. 1. The role of PSNet: replacing FPS and local points query in the hierarchical feature abstraction of local areas. Blue tensors are the sets of the 3D coordinates, black tensors are the sets of abstract features, m, s, n is the number of points, local areas, and local points respectively. PSNet can replace the traditional data structuring methods easily in the plug-and-play manner.

The current mainstream methods [17]–[34] take the points directly as the input of the network model. Based on the seminal method presented in [18], many research studies have been conducted to improve the abstraction of local context features. These works can be divided into three categories: point-based [17], spatial convolution [21]–[28] and graph convolution [29]–[31], [33]. For local context feature abstraction, these methods mainly adopt the hierarchical architecture for local context aggregation. Due to the disorder feature of the points in a point cloud, the points with similar spatial positions are often stored in non-contiguous locations in memory. However, the hierarchical architecture for local context aggregation needs to organize the data of the point cloud according to their spatial feature. This is a typical data structuring problem and has to be handled in all the above three categories of deep learning methods.

A current mainstream data structuring method includes two steps as shown in Part A of Fig. 1: 1) uniform subsampling such as Farthest Point Sampling (FPS) [35], 2) local grouping such as ball query, kNN or cube query. The subsampling step obtains a subset of points in the point cloud, and then the local grouping step uses each sampling point as the center of a local area, and groups the points around the center point into a local area. The grouping methods have to calculate the Euclidean distances between each of the sampled points and all other points to determine which points should be placed in the local area. However, most of the calculations are actually not necessarily needed, and the same calculations are repeated in both steps. Moreover, FPS searches for the

sampling points in sequence, which makes it difficult to parallelize the calculations. The study in [36] shows that the time spent in data structuring can account for 88% of the total processing time.

On the other hand, FPS is unstable in the sense that the sampling result of FPS is closely related to the starting points of the FPS process (which are typically selected randomly) and the point ordering in a point cloud. The unstable sampling results of FPS make the effectiveness of FPS unstable too. In addition, although FPS can ensure the uniformity of sampling, FPS is essentially based on a single metric *i.e*. the Euclidean distance between the points, and does not actively consider other features of the points, which may limit the effectiveness of the sampling results.

Now let us think how human would perform the data structuring work. When human tries to group the points in a point cloud into local areas, the points that are far apart or show unrelated features would be ignored straightaway. Consequently, the calculation of the distances between these points can be avoided. Human is able to sense which points should be grouped together, which are the points with the similar spatial location feature. This insight inspired the Point Structuring Net (PSNet) method proposed in this paper.

PSNet tackles the data structuring issue in point clouds, which accounts for a substantial proportion of the time in both model training and inference due to the non-Euclidean data. PSNet aims to replace the existing data structuring methods in deep networks on point clouds as shown in Fig. 1. PSNet is novel in the sense that it addresses this issue from a totally different perspective, which significantly reduces the time spent in data structuring. In PSNet, each individual point, represented by its Cartesian coordinates (x, y, z), is transformed to the high-dimensional features by our spatial feature transform function (SFTF), which determines the correlation degree between each point and the abstract features of each local area. Then, the points that are highly correlated with the local abstract features are grouped into a local area. The point with the feature closest to the abstract feature of a local area is regarded as the sampling point of this local area.

Moreover, we found that the symmetry of the Cartesian coordinates may cause incorrect grouping of the points in the symmetrical parts of a point cloud. Inspired by the spherical coordinates, we add two more parameters as the input spatial feature of a point, which reduces the grouping errors effectively. The data structuring of PSNet is based on the spatial features of the points, not only on a single distance metric. PSNet is embedded into a deep network and co-trained with the network (supervised by the loss function of the original deep network). Consequently, PSNet is adaptable to the objectives of the original learning tasks while encoding the features of the points and the local areas. This is the underlying reason why PSNet is more effective than the heuristic methods such as FPS and kNN.

The existing methods perform subsampling and grouping as two separate stages in sequence (first subsampling and then grouping). They make use of the spatial relations among the points to make subsampling and local grouping decisions. Therefore, they have to calculate the Euclidean distances between the points repetitively, which takes long computing time. In PSNet, the features transformed by SFTF can be used for both sampling and grouping simultaneously, which not only avoids the expensive distance computations, but also reduces unnecessarily repeated computations.

The SFTF can be performed on each point independently. The process can therefore be fully parallelized. PSNet can generate the effective data structuring results with much less time during the model training, while the data structuring time spent by PSNet in model inference can be almost neglected. PSNet can easily replace the data structuring methods in point cloud deep learning models in a plug-and-play manner. We have conducted extensive experiments on a variety of mainstream deep learning models on point clouds. The experimental results show that PSNet is effective and significantly reduces the training and inference time of the models. We have implemented PSNet, which has been open-sourced at *https://github.com/lly007/PointStructuringNet*.

The remainder of this paper is organized as follows. The related work is discussed in section II. PSNet is presented in detail in section III. The experimental results are presented in section IV. The paper is concluded in section V.

II. Related Work

In this section, we discuss various types of data structuring methods in deep learning on point clouds. They mainly include basic but widely used methods, the custom-designed methods for specific models, the random point sampling method and learning-based methods.

A. Most Widely Used Methods: FPS, kNN and Ball Query

PointNet++ [17] aggregates the local features hierarchically. Other methods [21]–[28] define unique spatial convolution operators, and perform the convolution on the local area. In addition, there are many graph-based methods [29]–[31], [33], [37], which use the coordinate-based local-area queries to construct the graphs. All these methods use FPS [35] for subsampling and use kNN or ball query for local grouping. However, FPS, kNN and ball query have the issues in both efficiency and effectiveness.

In terms of efficiency, as the number of points in the point cloud increases, the amount of calculations in FPS increases significantly. Moreover, FPS searches for the farthest points iteratively in a point cloud, which is difficult to be parallelized. The mainstream local grouping methods are ball query or kNN. Both methods need to calculate the Euclidean distances between each of the sampled points and all other points. Many distance calculations are repetitively performed in the sampling and the grouping phase. Our studies show that a large portion of these calculations are unnecessary.

As for the effectiveness, since FPS and kNN (or ball query) are essentially the heuristic approaches (based on the distance between the points), the data structuring results obtained by FPS and kNN (or ball query) are the same for different training models and learning tasks, not adapted to the feature abstraction operators, the network architecture and the objectives of the learning tasks. These static data structuring results limit the effectiveness of the training models.

B. Bespoke Methods for Specific Models

Some works custom-designed the data structuring methods to suit their unique network architectures. SO-Net [38] subsamples the point clouds via Self-Organizing Map (SOM) and discovers the neighbourhood of the sampled points by kNN. KD-Net [39] organizes the point cloud structure and transforms the features hierarchically through KD-Tree. SPG [40] generates local grouping of the point clouds according to the basic geometric shapes through the Superpoint Graph. But its computational overhead of shape partitioning and supergraph analysis are expensive. PVCNN [36] and VoxelNet [14] find the local areas by the voxel. They do not sample the points and therefore cannot reduce the number of input features in the deeper layers of the networks. The architectures of these methods are different from the usual PointNet++-based hierarchical deep networks. Rather, they design the special feature abstraction operators in their training networks in order to aggregate the local features. Therefore, it is hard to plug these data structuring methods into the existing, generic training networks.

C. Random Sampling

The Random Point Sampling (RPS) finds the sampling points randomly in the point cloud and is therefore very fast. However, the effectiveness of the sampling results is unstable. Consequently, using RPS in a typical method [17] can significantly reduce the effectiveness of the network.

Some works [41], [42] use RPS as the subsampling method. However they have to design the special network architecture or feature abstraction operators for the training networks in order to offset the instability caused by RPS. For example, Grid-GCN [42] uses the spatial voxel division to constrain RPS and prevent the excessive randomization of sampling. RandLA-Net [41] does not have the constraints in the random sampling phase, but has to be mapped to a dilated residual block to counteract the information loss caused by random sampling. These specially designed methods are not universally applicable and cannot be used in most deep learning models for point clouds.

D. Learning-based Methods

Some works proposed the learning-based sampling methods, which learn the sampling strategies through the lightweight neural networks. SampleNet [43] and S-NET learn a subset of a point cloud by the neural networks. In these methods, other heuristic sampling methods (such as FPS) have to be used to provide the baseline for learning. Then the sampling points generated by the learning-based methods have to be compared with the points in the original point cloud, which introduces extra computations. CAE [44] and PAT [45] use the reparameterization trick to calculate the sampling weight matrix according to the relationship among all points. CP-Net [46] evaluated the contribution to the aggregated features and selected Critical Points (CP) as the sampling points. The learning-based methods are optimized for both the task and the deep network in sampling to improve the effectiveness of the sampling. However, these methods do not perform local grouping. Rather, the traditional local grouping methods such as kNN or ball query have to be used to complete the data structuring.

PSNet proposed in this work is a fast data structuring method based on deep learning, and adopts a totally different approach from all the data structuring methods discussed above. PSNet does not reply on the spatial relations among the points for data structuring, and can be fully parallelized. PSNet is a generic approach, and can be embedded into the existing mainstream deep learning networks for point clouds in a plug-and-play manner. Moreover, in most methods in the literature, sampling and local grouping are two separate tasks. In PSNet, grouping and sampling are performed at the same time. This feature further improves the efficiency of PSNet.

III. PSNET

Querying local areas on a point cloud can be regarded as a multi-clustering problem in the Euclidean distance space (*i.e*. clustering the points into local areas). In a 3D space, the points that are closer to each other should be grouped into a local area. The traditional methods (*e.g*. radius query and kNN) find a class of points close to a certain point from the above perspective and group them together. This type of methods first uses subsampling (*e.g*. FPS) to find a subset of the point cloud as the center points of the neighborhoods. PSNet (Point Structuring Net) proposed in this work does not use the Euclidean distance between the points as the selection criterion for the neighborhood, but exploits a multi-clustering method to divide the points into local areas.

A. Spatial Location Feature Transform Function

Traditional methods explicitly rely on the heuristic geometric meaning of points such as the Euclidean distance between points. More specifically, they must calculate the distance between a certain point and **all** remaining points, and then determine their relationship (*i.e*. whether they belong to the same local area). Since a local area is usually small, most of the calculations, which are performed all remaining points, are unnecessary. It is also worth noting that in the grouping methods, whether a point belongs to a local area is only related to its distance to the center of the local area (*i.e*. a sampled point), irrelevant to the local shape.

The points that are close to each other in the 3D space often have similar features when they are transformed by the same neural network operator. The *Spatial Features Transform Function (SFTF)* proposed in this work does not focus on the heuristic correlation between the points (*e.g*. the Euclidean distance). Rather, *SFTF* abstracts the spatial features of each individual point, and groups the points with similar features in the same local area.

A point cloud is $P = \{\boldsymbol{c_i} | i = 1, 2, \cdots, m\}$, where m is the number of points, $\boldsymbol{c_i} \in \mathbb{R}^d$ is the spatial features of the i-th point p_i, $L = \{l_j | j = 1, 2, \cdots, s\}$ is the set of local areas which P is divided into, l_j is local area j, s is the number of local areas and also the number of sampling points.

In *SFTF*, local grouping for a point cloud P is treated as the problem of determining the correlation between a point p_i and one (or more) of the s local areas (*i.e.* whether p_i is a member of a particular local area). SFTF aims to transform the spatial features of a point into the degree of correlation between a point and local areas.

Define $t(x) : \mathbb{R}^d \rightarrow \mathbb{R}^s$ as the *SFTF* function for point p_i with the spatial features $\boldsymbol{c_i}$ (the feature dimension is d):

$$\boldsymbol{f_i} = \boldsymbol{t}\left(\boldsymbol{c_i}\right) \tag{1}$$

where $\boldsymbol{f_i} \in \mathbb{R}^s$ is the vector of correlation degrees between point p_i and each of s local areas.

With *SFTF*, the problem is converted into an s-class classification problem given a correlation vector $\boldsymbol{f_i}$ (*i.e.* given $\boldsymbol{f_i}$ which of the s classes point p_i is classified into).

Eq. (2) is used to apply the $sigmoid$ function to the correlation vector $\boldsymbol{f_i}$ to obtain the probability vector $\boldsymbol{q_i}$, an element of which holds the probability that point p_i belongs to one of the s local areas $l_j (j = 1, 2, \cdots, s)$, where $\boldsymbol{q_i} \in Z^s,\ 0 < Z < 1$. For a single classification problem, we only need to obtain the index of the element which has the largest value in the vector (denoted by $argmax(\boldsymbol{q_i})$). For a multi-class (assuming n-class) problem, the elements which have top n values (denoted by $argtop_n(\boldsymbol{q_i})$) are the n classes that p_i should be allocated to.

$$\boldsymbol{q_i} = sigmoid(\boldsymbol{f_i}) \tag{2}$$

However, after local grouping of a point cloud, the maximum number of points in a local area should be fixed (*i.e.* each area contains at most k points), which cannot be guaranteed by the above formulation of the classification problem. Therefore, we modify the classification process as follows. We first apply Eq. (2) to each point and then apply Eq. (3), where T is the pointwise broadcast version of t. $T\left(P\right) \in \mathbb{R}^{m \times s}$ takes as input the entire set of points in point cloud P and outputs a two dimensional matrix in which there are m rows (corresponding to m points in P) and s columns (each row is $\boldsymbol{f_i}$ in Eq. (1)); $Q \in \mathbb{R}^{m \times s}$ is the membership probability matrix between each point in P and each local area in L (the set of local areas).

$$Q = sigmoid\left(T\left(P\right)\right) \tag{3}$$

A column in Q is denoted by $\boldsymbol{e_j} \in \mathbb{R}^m$. Each element in $\boldsymbol{e_j}$ is the probability that point p_i belongs to local area l_j. The indices of the points in area l_j can be obtained through these elements.

n is the number of points in l_j. Then finding the indices of the points in l_j can be formulated as:

$$indices_j = argtop_n\left(\boldsymbol{e_j}\right) \tag{4}$$

where $indices_j \in \mathbb{R}^n$, $argtop_n$ is the indices of the top n elements in the vector.

The above process can be understood in the following way. A local area l_j is abstracted to be a type of feature. The points belonging to l_j should be closer to the abstract feature of l_j after performing the feature transformation function T. The process of finding the indices of the top values in e_j is equivalent to finding n points in the point cloud that best match the feature of local area l_j.

Our grouping method reflects an important viewpoint of us in performing local grouping for point cloud models. In the existing grouping methods such as ball query and kNN, the grouping decisions are made essentially based on the Euclidean distance, which is a single heuristic metric. We would like to argue that in point cloud models, which is non-Euclidean data structure, the distance should not be the sole metric to determine the grouping of points. Our PSNet can adjust the grouping of points adaptively based on the extracted local features, rather than only on a single heuristic metric such as the Euclidean distance. Actually, even for the Euclidean data structure such as images, some studies have proposed to use adaptive kernels (instead of a kernel with the fixed size such as a 3x3 matrix) to perform the convolution operation. For example, the work in [47], [48] proposed the "Deformable Convolutions", in which the convolution kernel may be deformed adaptively in the learning process and as the result the local divisions of the image are also adjusted adaptively.

B. Selection of Subsampling Points

Vector $\boldsymbol{e_j}$ introduced in Eq. (4) represents the probabilities that point p_i is a member of each local area $l_j (j = 1, 2, \cdots, s)$. The point with the highest probability is the point whose features best match the features of local area l_j among all points in the local area and therefore should be selected as the subsampling point. Therefore, the subsampling point for local area l_j can be determined by:

$$sub_j = \{p_k | k = argmax\left(\boldsymbol{e_j}\right) = argtop_1\left(\boldsymbol{e_j}\right)\} \tag{5}$$

It is easy to understand that the output of $argsort_1$ in Eq. (5) exists in the output of $argsort_n$ in Eq. (4). Namely, after local grouping is completed, the set of subsampled points can also be determined. $sub \in \mathbb{R}^{s \times 3}$ denotes the set of subsampled point cloud.

It is worth noting that in the PointNet paper [17], when the authors discuss the reason for the effectiveness of their proposed method, they stated that the points corresponding to the maximum values in the feature channels "summarize the skeleton of the shape". In the experiment section, we will visualize our sampling results (Fig. 5), which show that the points sampled by our method form the skeleton of an object.

C. Micro-geometric Meaning

The micro-geometric meaning of PSNet is that after being transformed by the trained *SFTF*, the points with similar locations are also similar in each channel (a channel represents an abstract feature of the local area) of the s-dimensional feature space. Since *SFTF* transforms each point independently and $argtop_n$ is a symmetric function, PSNet is permutation invariant. This means that the data structuring results obtained by PSNet are stable. Namely, the results of PSNet are not affected by the point ordering in a point cloud, and PSNet will obtain similar data structuring results for the 3D objects with similar

spatial features. In contrast, the results obtained traditional sampling and grouping methods such as FPS and kNN heavily rely on the starting points of the sampling/grouping method. However, the starting points are often randomly selected, which makes the effectiveness of their results unstable. Also due to this problem, the traditional methods may generate different sampling and grouping results for the objects with similar features.

D. Correct Symmetry Errors

Although the Cartesian coordinate can fully describe the position of a point in a point cloud, our studies show that if the shape of a point cloud is symmetric, *SFTF* may transform the symmetric points in a point cloud to the similar high dimensional features. This is due to the symmetrical parts of an object often have similar spatial features. For example, when the two symmetric points $[1, 2, -3]$ and $[1, 2, 3]$ will be transformed to the same value after right multiplying by $[1, 1, 0]^T$. In order to address this issue, we introduce two more features, polar angle (θ) and azimuthal angle (φ) (inspired by the spherical coordinate) as the input spatial feature of a point (in addition to its Cartesian coordinate). The values of θ and φ are supposed to be in the ranges of (0, π] and (0, 2π]. Therefore, the problem of transforming the symmetric points can be almost eradicated, which is supported by our experimental results. Typically, the data format of a point cloud only provides the Cartesian coordinates of the points. θ and φ required by *SFTF* can be calculated through the Cartesian coordinates, as in Eq. (6), where x,y,z are the Cartesian coordinates of a point, $r = \sqrt{x^2 + y^2 + z^2}$ is the radius of the spherical coordinate.

$$\begin{cases} \theta = \arccos\left(z/r\right) \\ \varphi = \arctan\left(y/x\right) \end{cases} \tag{6}$$

However, when $\arctan\left(y/x\right)$ in Eq. (6) is used to calculate φ, it has to determine the quadrant of the xy plane. In the implementation, we avoided this computation overhead by invoking the $atan2(y, x)$ function instead (which does not have to determine the quadrant). $atan2(y, x)$ is provided by most mathematical libraries [49]–[51]. Moreover, since the value range of $atan2$ is $(-\pi, \pi]$, we define $\varphi = \ atan2(y, x) + \pi$, so that the value range of φ can become $(0, 2\pi]$. In summary, the final input spatial features of a point are:

$$\begin{aligned} c_i &= [x, y, z, \theta, \varphi] \\ &= [x, y, z, \arccos\left(z/r\right), atan2\left(y, x\right) + \pi] \end{aligned} \tag{7}$$

E. Differentiable Indexing

The $argmax$ function in Eq. (5) is non-differentiable, because the indices are a series of discrete integer values. So backpropagation cannot propagate through $argmax$ in training to calculate the gradients of the parameters in the feature transform function. $argmax$ needs to be replaced by a differentiable function [52].

We apply the $Gumbel\ Softmax$ distribution function to the probability distribution $\boldsymbol{e_j} \in \mathbb{R}^m$ of the sampling points:

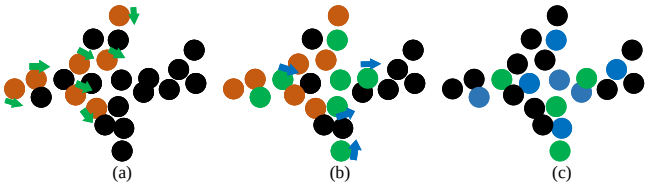


Fig. 2. Changes of sampling results during training. Brown, green and blue points respectively represent the sampling results of different training stages. The arrows indicate the deviation between the current sampling points and the target sampling points. Deviations provide directions from current sampling points to target. Green arrows are the deviation from brown points to green points, blue arrows are the deviation from green points to blue points. Blue points are the final sampling result.

$$\widetilde{\boldsymbol{e_j}} = softmax\left(\left(\ln\left(\boldsymbol{e_j}\right) + noise_j\right)/temperature\right) \tag{8}$$

where $temperature$ is annealing temperature. $noise_j$ can be expressed by Eq. (9), where U is a uniform distribution.

$$noise_j = -\ln\left(-\left(U_i\right)\right), U_i \sim U\left(0, 1\right) \tag{9}$$

where U is uniform distribution.

When the annealing temperature value approaches 0, the $Gumbel\ Softmax$ function tends to convert $\boldsymbol{e_j}$ to a one hot vector. The index of the element with the value of 1 in $\widetilde{\boldsymbol{e_j}}$ is $argmax\left(\boldsymbol{e_j}\right)$. We apply the $Gumbel\ Softmax$ function to all $\widetilde{\boldsymbol{e_j}}$ in matrix Q:

$$\widetilde{Q} = GumbelSoftmax_m(Q) \tag{10}$$

where $GumbelSoftmax_m$ is Gumbel Softmax calculated by the column. $\widetilde{Q} \in \mathbb{R}^{m \times s}$ is the sparse matrix of sampling indices. The sampled points can be obtained by left-multiplying the point cloud $P \in \mathbb{R}^{m \times 3}$ by $\widetilde{Q} \in \mathbb{R}^{m \times s}$, *i.e.* Eq. (11), where $sub \in \mathbb{R}^{s \times 3}$.

$$sub = \widetilde{Q}^T \times P \tag{11}$$

Differentiable indexing is an important reason for the effectiveness of PSNet. Traditional heuristic data structuring methods rely on uniform sampling, FPS guarantees the uniform distribution of local areas. This idea is very intuitive and vanilla, which has certain universal applicability. But we would like to argue that it is not the best way for data structuring. In fact, recent studies have shown that there are differences in the degree of attention of different regions of the sample data (*e.g.*, attention mechanism [53]), and a heuristic method will restrict this difference. In this sense, the adaptive point cloud data structuring method is more effective. We chose *Gumbel* as the index strategy instead of *Reinforce*, because *Gumbel* can provide the guidance for adaptive point cloud sampling to converge in the best way. Fig. 2 shows how *Gumbel* affects the sampling results during the training process. Due to the random initialization of the network, the positions of sampling points in PSNet may be uneven at first, as shown by brown points in the Fig. 2(a), which is obviously not the best sampling result. However, *Gumbel* will affect the training of the parameters of PSNet, causing a slight deviation of the

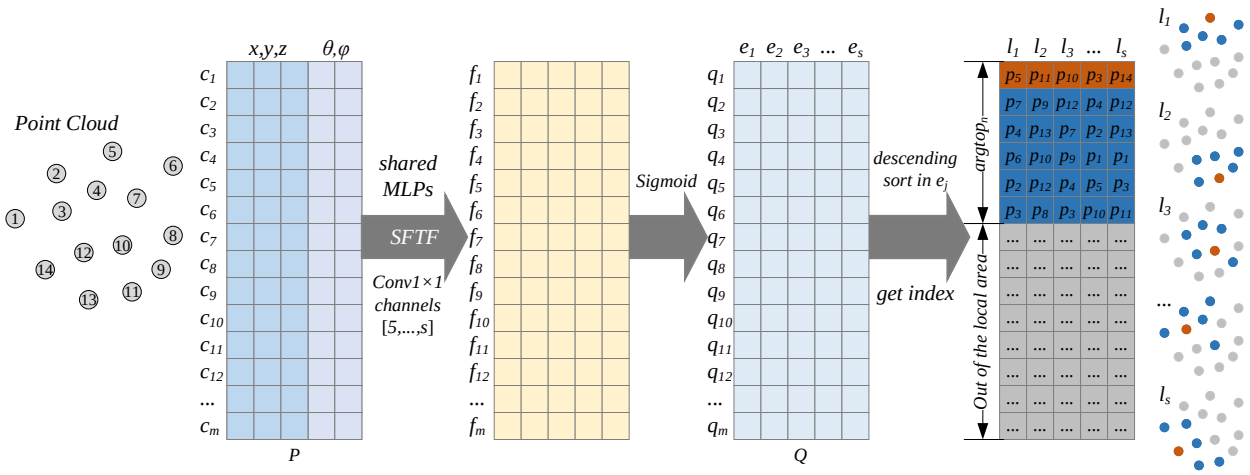


Fig. 3. The architecture of PSNet. The spatial location feature transform function is implemented by 1×1 convolution, then applies sigmoid to the matrix, sort each column in descending order. In the output matrix of PSNet, the orange points are the subsampling points, and the blue points are the points in the local areas of orange points.

target sampling point (shown by the arrows in the figure). This deviation will cause a slight change in the sampling target, which is more conducive to task optimization results. The deviation degree depends on the temperature setting. After a period of training, the sampling points will reach the positions of the green points in (Fig. 2(b)). As the training continues further, the sampling points will reach the positions of the blue points (Fig. 2(c)). PSNet will be trained at the same time as the task, which means that in addition to optimizing the parameters of the original operator to improve the effectiveness of feature abstraction, the training **also adjusts the sampling method** to improve the effectiveness of the model from another perspective.

F. The Architecture of PSNet

PSNet is a sub-network that can be plugged in a deep learning network for point clouds. Since PSNet does not use the position relationship among the points, T can be applied to all points in parallel. This is a main reason why PSNet can speed up the subsampling process significantly over FPS.

The architecture of the network is shown in the Fig. 3. The MLP (Multilayer Perception) network is shared among different points to implement the function T (the first step in Fig. 3). In order to take full advantage of the parallel computing capability of modern GPUs, we implement the MLP by multi-layer 1×1 convolution. The number of channels the first layer in the convolution is 5 as shown in Eq. (7), while the number of channels of the last layer is the number of sampling points s (which is also the number of local areas). There can be one or more hidden layers in MLP (the impact of the number of layers will be evaluated in Table XI). MLP outputs an $m \times s$ matrix. As described in Eq. (2) and Eq. (4), $sigmoid$ is applied to the matrix (Step 2: $sigmoid$ in Fig. 3) and the elements in each column are sorted in the descending order of their values (Step 3 in Fig. 3). By taking the index of the first element and the indices of the top n elements in each column, we can obtain the sampling point and the group of points in the local area.

Different indexing strategies are adopted in the training and the inference stage to improve the inference speed. In order to ensure the differentiability of the MLP parameters, the $Gumbel\ Softmax$ method in Eq. (11) is used to find the sampling points in the training stage. In the inference stage, there is no need to consider whether the backpropagation is truncated. So the $argmax$ and $argsort$ functions can be directly used to find the sampling points and the points in local areas. Note that only the indices of sampling points are trained, because among all points in a local area, the features of a sampling point are the closest to the features of the local area. This can greatly reduce the storage and computation cost caused by differentiable indexing.

PSNet will update the abstract feature of the local areas to reflect the points that carry more meaningful information when the deep learning network performs the feature abstraction in this local area. On the contrary, the points with the weaker impact may be excluded from the local area.

G. Supervision Information

PSNet is embedded into a training network and co-trained with the training network, through which PSNet can be adapted to provide the data-structuring results more effective for the feature abstraction operators of the training network and the objectives of the learning task. PSNet does not have its own loss function. Instead, the training of PSNet is supervised by the loss function of the learning task running on the training network that PSNet is plugged into.

For example, assuming that the loss function is cross-entropy, the loss function for a single sample (*i.e.* a shape in a classification task or a point in a segmentation task) can be formulated as:

$$L_{task} = -\sum_{c=1}^{M} y_c log(p_c) \tag{12}$$

where M is the number of categories, y_c is the label of the sample, p_c is the prediction probability of the sample.

H. Handling Large Scale Point Cloud Scenes

Current mainstream deep learning models (*e.g*. PointNet++, PointConv, RS-CNN, GAC, *et al*.) can only handle point clouds of limited scale (usually within 10K points). When being faced with larger-scale point cloud data (such as 1 million points), the input point clouds of the deep learning networks are typically pre-processed to reduce to a smaller scale. The following are the pre-processing methods used in the above mainstream deep learning models for processing shape classification tasks and the scene segmentation task.

Shape classification for a point cloud focuses more on the overall features of the object, the points are randomly sampled in the preprocessing phase to reduce the point cloud to an acceptable scale. The sampled points are then input into the deep learning networks. The time complexity of the random sampling pre-processing method is $O(1)$.

As for the scene segmentation task, the unconstrained random sampling method may cause the loss of fine-grained features of a point cloud. Therefore, a large scale point cloud scene is first partitioned into regular voxels or grids. Then the random sampling is performed within each voxel or grid. This way, the fine-grained features of the point cloud can be preserved.

I. Time Complexity Analysis of PSNet

The time complexities of FPS and kNN neighborhood query is $O(m^2)$ [43] and $O(ms + mlog_2n)$ (distance calculation and heap sort), respectively. The total time complexity of FPS+kNN is $O(m^2 + ms + mlog_2n)$. The time complexity of a single-layer PSNet is $O(ms + mlog_2n)$ (*SFTF* and heap sort).

Considering that s and n are much smaller than m in general, PSNet is expected to be much faster than FPS+kNN. Moreover, since *SFTF* processes each point independently, it can be embarrassingly parallelized in modern deep learning frameworks on GPUs, which manifests the feature of weak scaling. Namely, the processing time for the point cloud remains constant as the number of points increases, and the speedup increases linearly as the number of processing elements increases.

On the contrary, FPS cannot be parallelized because FPS has to query the points one by one in sequence. Therefore, at least s iterations are required for subsampling. kNN calculates the Euclidean distance between each of the sampling points and all other points, which can be parallelized in theory. However, the unit operation in kNN is to calculate the Eulidean distance between two points while the unit operation in *SFTF* is a multiplication, *i.e*. a feature value times a model weight. The cost of unit operation in kNN is much higher than that of multiplication. Thus, even if kNN is fully parallelized, it is still much slower than our PSNet. We have conducted the experiments to verify this (Table III).

Furthermore, it is easy to implement PSNet. It does not require the developers to have advanced CUDA programming skills. It can be easily implemented on GPU using high-level programming paradigm provided in the deep learning frameworks such as Tensorflow and PyTorch [49], [51].

Methods	Classification		Part Segmentation			
	Orig.	PSNet	C-mIoU		I-mIoU	
			Orig.	PSNet	Orig.	PSNet
PN++(MSG)	91.9	**92.3**	81.9	**82.1**	85.1	85.0
PN++(SSG)	91.7	**92.2**	81.6	**82.1**	84.8	84.8
PointCNN	92.2	**92.3**	86.1	**86.2**	84.6	**84.7**
PointConv	92.5	92.4	82.8	**82.9**	85.7	85.7
RS-CNN	92.6	92.6	84.1	**84.2**	85.8	85.8
DensePoint	93.2	**93.3**	84.2	84.2	86.4	86.4

TABLE I
EFFECTIVENESS OF DEEP LEARNING MODELS INTEGRATED WITH PSNET; THE CLASSIFICATION TASK IS RUN ON MODELNET40 AND THE SHAPE PART SEGMENTATION TASK RUN ON SHAPENET, ORIG. STANDS FOR ORIGINAL MODEL, C-MIOU FOR CLASS MIOU, I-MIOU FOR INSTANCE MIOU, PN++ FOR POINTNET++, SSG FOR SINGLE SCALE GROUPING, MSG FOR MULTI-SCALE GROUPING.

IV. EXPERIMENTS

We have conducted extensive experiments to evaluate PSNet. Our experiments consist of five parts: effectiveness, efficiency, robustness, ablation studies and visualization. In the experiments about effectiveness, we process the tasks in three mainstream application scenes of point clouds. We modified the deep learning models in the literature by replacing sub-sampling and grouping in the original models with PSNet, and compared the performance between the original and the modified models. In the efficiency experiments, we recorded the inference time and the training time of the deep learning models with PSNet, and compared the time with the original models. In addition, we provided the visualization of the data structuring results.

Our comparison experiments strictly follow the experimental and hyperparameter settings of the original model. The basic model transformation such as translation, scaling, rotation and jitter are also randomly performed to enhance the robustness of model training.

A. Effectiveness

1) Classification and Part Segmentation: To evaluate the performance of PSNet in the application of shape classification and part segmentation, we modified the point-based method PointNet++ [17], convolution-based methods PointCNN [22], PointConv [26], RS-CNN [24] and DensePoint [23] by replacing their data structuring methods with PSNet. The benchmark dataset of classification and segmentation is ModelNet40 [54] and ShapeNet [55] respectively. The results are shown in Table I.

In PointConv and RS-CNN, the performance of the network with PSNet is almost the same as that of the original model. There is the slight performance improvement compared to the original model in PointCNN and DensePoint with PSNet. Note that no matter it is single scale or multi-scale grouping, the accuracy of PointNet++ with PSNet is even better than that of the original models. This may be because PSNet is able to provide more appropriate local grouping than the

Methods	mIoU		OA	
	original	PSNet	original	PSNet
PointNet++	53.1 (MSG)	**53.2 (SSG)**	84.0 (MSG)	**84.4 (SSG)**
PointCNN	57.3	**57.4**	85.9	85.9
GACNet	62.9	62.9	87.9	87.8

TABLE II
SCENE SEGMENTATION COMPARISON ON S3DIS AREA5.

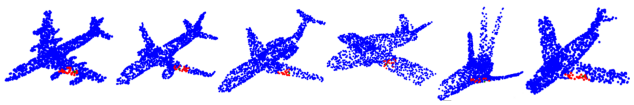


Fig. 4. Visualization of stability of the data structuring results by PSNet in processing objects with similar features. In each shape, the red points represent a local area.

m→s	Inference Time(ms)			Memory	
	FPS	kNN	PSNet	F+k	PSNet
1024→128	28.3	1.4	**0.64**	<1M	<1M
1024→512	109.1	2.1	**0.78**	<1M	<1M
2048→128	28.7	4.0	**0.68**	1.5M	1.5M
2048→512	110.4	4.1	**0.80**	8M	**6M**
30000→512	128.0	30.4	**0.81**	85M	**75M**
30000→1024	235.4	30.7	**0.80**	160M	**140M**
30000→4096	970.3	31.5	**0.81**	633M	**550M**
80000→512	215.6	75.5	**0.81**	270M	**235M**
80000→1024	420.1	77.1	**0.81**	490M	**440M**
80000→4096	1577.3	80.4	**0.90**	1960M	**1735M**
80000→16376	6701.1	97.3	**0.97**	8340M	**7144M**

TABLE III
INFERENCE TIME AND MEMORY IN DATA STRUCTURING, m AND s ARE THE NUMBER OF INPUT POINTS AND SAMPLED POINTS RESPECTIVELY. BATCH SIZE IS 8. F+K MEANS FPS+KNN.

Task	Methods	Original	S&G(%)	PSNet
Shape Classification (32/batch)	PointNet++	262.5	62.9	**97.6**
	PointCNN	277.4	60.4	**110.4**
	PointConv	335.4	61.2	**131.2**
	RS-CNN	340.9	57.5	**145.5**
	DensePoint	320.1	58.1	**135.4**
Part Segmentation (32/batch)	PointNet++	197.3	75.9	**47.6**
	PointCNN	200.5	71.1	**58.1**
	PointConv	210.9	72.6	**58.1**
	RS-CNN	220.6	70.4	**65.5**
	DensePoint	212.7	70.7	**63.1**
Scene Segmentation (16/batch)	PointNet++	468.7	46.1	**256.7**
	PointCNN	494.5	44.7	**275.8**
	GACNet	510.1	44.1	**279.4**

TABLE IV
INFERENCE TIME (MS) SPENT IN NETWORK FORWARD PROPAGATION. S&G(%) IS THE PERCENTAGE OF TIME SPENT IN SAMPLING AND GROUPING IN THE ORIGINAL MODELS.

original models. These results show that PSNet improves the effectiveness of the deep learning models.

2) Scene Segmentation: The scene segmentation task is more complex than shape classification and part segmentation. It contains a large amount of noise data. The number of input points is very large, which can verify the effectiveness of PSNet in processing large-scale point clouds. We use the Stanford 3D Large-scale Indoor Spaces (S3DIS) [56] dataset in the experiments. The common scene segmentation benchmark is trained on areas 1-4 and tested on a completely independent area 5. We modified PointNet++ [17], PointCNN [22] and the graph-based method GAC [37], and conducted the comparison experiments. The experimental results are shown in the Table II.

Compared with the original model, PSNet achieves better or the same effectiveness. The performance of PointNet++ with PSNet is even better than the original PointNet++ multi-scale grouping model. This result shows that PSNet is effective in building local graphs for the graph neural networks. GACNet strengthen the robustness of local changes by local relationships representation. This makes their effectiveness more difficult to benefit from stable data structure.

3) Stability: Stable data structuring methods help improve the effectiveness of the models. Compared with the widely used FPS+kNN method, PSNet is stable. We demonstrate this by randomly selecting some airplane shapes and processing them with PSNet. The results are visualized in Fig. 4.

A channel in PSNet represents an abstract spatial feature (the number of channels is set to be same as the number of local areas). The points with the highest scores in a channel are grouped as a local area (the point with the highest score in a channel is the sampled point), meaning that these points share the similar abstract spatial feature embedded in the channel. We randomly selected a local area no. (No. 16 in Fig. 4). It can be seen from Fig. 4 that when processing different airplane models, local area No. 16 is always located at a similar part of the airplane (the part of airplane wing that is close to the airplane body). These results show that the grouping results produced by PSNet represent some particular geometric meaning (embedded in the corresponding channel). On the contrary, the local areas produced by FPS+kNN are random. If we run FPS multiple times even for the same point cloud model, the sampling results of different runs are different. If there are different point orderings for the same point cloud, the sampling results by FPS will be different too. The randomness of FPS and consequently the grouping results by kNN affect the consistency and effectiveness of the data structuring results.

B. Efficiency

We conducted a series of experiments to verify the efficiency of PSNet. In this section, the MLP channels of the *SFTF* T in PSNet were set to 5, 32, 128 and s. All experiments were conducted on TITAN RTX. For fair comparison, all methods are implemented in PyTorch [51].

1) Inference Time: We evaluated the efficiency of the forward propagation of PSNet, including the running time of PSNet and the overall running time of the entire training network which PSNet is plugged in. We compare the efficiency of PSNet with that of FPS and the networks using FPS. In the experiments, the number of points is 1024 or 2048 for the ordinary point clouds, and 30,000 or 80,000 for large scale point clouds.

We recorded the time spent in sampling and grouping separately. But since in PSNet sampling and grouping are performed at the same time, we only recorded the total time spent by PSNet. We also recorded the memory consumption of the methods. The results are shown in the Table III.

Task	Methods	Original	PSNet
Shape Classification (32/batch)	PointNet++	104	**68**
	PointCNN	109	**70**
	PointConv	117	**71**
	RS-CNN	106	**69**
	DensePoint	101	**66**
Part Segmentation (32/batch)	PointNet++	188	**118**
	PointCNN	193	**125**
	PointConv	201	**133**
	RS-CNN	194	**122**
	DensePoint	184	**114**
Scene Segmentation (16/batch)	PointNet++	1410	**810**
	PointCNN	1433	**811**
	GACNet	1445	**815**

TABLE V
COMPARISON IN NETWORK TRAINING TIME (MS).

It can be observed from Table III that with PSNet the inference time is reduced dramatically (becomes almost negletable), while either the less or same amount of memory are consumed. FPS is sensitive to the number of sampling points. It is caused by the iterative implementation of FPS. The time of the kNN grouping increases only slightly as the number of points in the point clouds increases.

Moreover, the time spent by PSNet remains almost the same as the scale of the point clouds or the number of sampling points increases. This is because the calculations in PSNet can be embarrassingly parallelized and therefore can take full advantage of parallel capacity in GPU. Even in the smallest scales of the point cloud and sampling points, PSNet spends only 2% of the time by FPS+kNN.

In theory, the parallelized kNN should have the same execution time as the parallelized PSNet. However, it can be seen from Table III that the time of PSNet is much less than that of kNN. Also, the time of PSNet remains almost constant as the numbers of input points and sampling points increase, while kNN does not. The reason for these may be because kNN mainly calculates the distances between points while PSNet processes the 1×1 convolution. The default parallelization schemes provided by the deep learning frameworks on GPU may be different for these two types of processing.

We also evaluated the inference time spent in network forward propagation. The results are shown in Table IV. The time spent in forward propagation of the network with PSNet is reduced significantly. For example, the time spent by the PointNet++ network with PSNet is only 24.1% of the original time in the part segmentation task. Further, we notice that the reduction percentage is related to the complexity of the feature abstraction method in the network. If the method itself is more complex, FPS and local grouping will take a smaller proportion of time in forward propagation. This observation indicates that PSNet does not interfere with the feature abstraction process of the model.

2) Training Time: As PSNet incorporates new training parameters in the model, it is a natural concern that it may lead to the increase in training time. We evaluate the training time, the results are shown in the Table V.

Although PSNet introduces new parameters, the training time of the network with PSNet is less than that of the original network. The improvement in forward propagation is far greater than the extra time taken in deriving and updating the new parameters, which results in a substantial reduction in the overall training time.

Methods	Sampling	Grouping	Total	Cls. Acc.	Seg. mIoU
SampleNet	1.04	4.10	5.14	91.8	81.6
CP-Net	0.82	4.10	4.92	91.7	81.9
PSNet	**0.80**		**0.80**	**92.2**	**82.1**

TABLE VI
COMPARISON WITH OTHER LEARNING-BASED METHODS; "SAMPLING" AND "GROUPING" ARE THE INFERENCE TIME (MS) AT THE SAMPLING AND THE GROUPING STAGE OF DATA STRUCTURING RESPECTIVELY, "CLS. ACC." AND "SEG. MIOU" ARE THE ACCURACY OF THE SHAPE CLASSIFICATION TASK AND CLASS-MIOU OF THE PART SEGMENTATION TASK RESPECTIVELY; m=2048, s=512, BATCH SIZE=8.

C. Comparison with Learning-based Sampling Methods

In addition to comparing with the original models of the respective methods, we also compared PSNet with other learning-based sampling methods in literature: SampleNet [43] and CP-Net [46]. The data structuring part of PointNet++ is replaced by a learning-based method. However, SampleNet and CP-Net can only perform sampling. They have to be paired with other grouping methods such as kNN to complete data structuring. To the best of our knowledge, there is yet not a method which can perform sampling and grouping at the same time like PSNet.

We used three metrics to evaluate the comparison: inference time for data structuring, accuracy for the classification task on ModelNet40, and class mIoU for the part segmentation task on ShapeNet. It can be seen from the Table VI that SampleNet and CP-Net take slightly longer time than PSNet even if they only perform sampling. When they are paired with the local gropuing method kNN, the total time spent by them in data structuring is 5.9 times that spent by PSNet. Although PSNet consumes less time, the performance in terms of both classification accuracy and segmentation mIoU is better than that of SampleNet and CP-Net. This is because PSNet can achieve adaptive sampling and grouping results through spatial features, which makes its data structuring results more suitable for both the feature abstraction and the specific objectives of the learning tasks. Although the sampling of SampleNet and CP-Net is adaptive to the tasks or the features to some extent, the paired heuristic grouping method (kNN) cannot improve the effectiveness of grouping.

D. Visualization

1) Visualization of subsampling: In this subsection, we visualize the subsampling results during the PSNet process. The point whose features best match those of a local area is selected as a subsampling point (*i.e.* the representative point of the local area). Fig. 5 shows the subsampling results on a few randomly selected point cloud shapes from ShapeNet when they are processed by PointNet++ plugged with our PSNet, where the point cloud in the first column is subsampled to 25% and 12.5% of the points in the second column and the third column, respectively. As can be seen from the figure, the points

are uniformly sampled from the point cloud. Although the number of the points is greatly reduced after subsampling, the basic structures of the objects are still retained. These results demonstrate the excellent subsampling ability of PSNet.

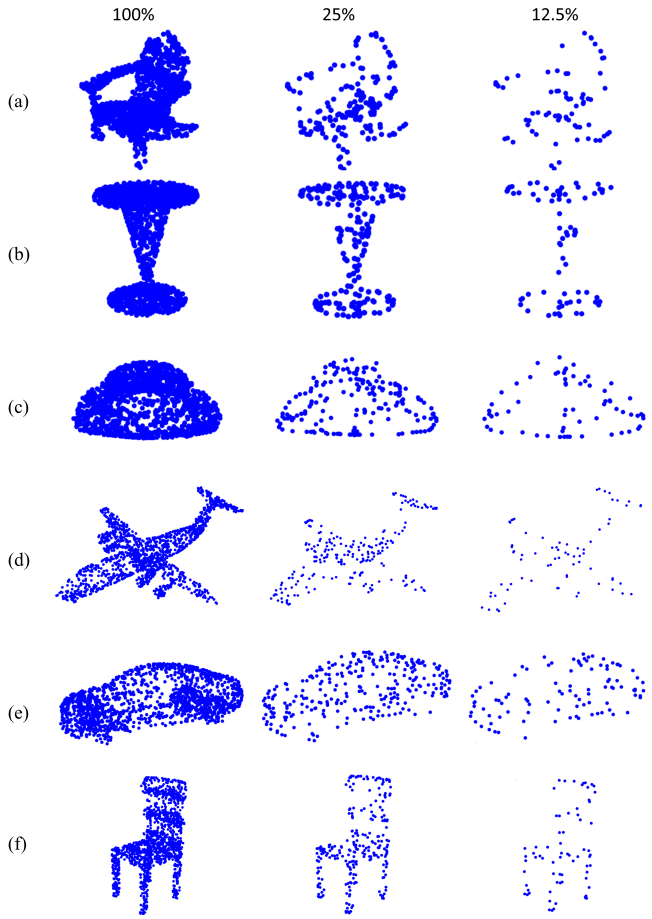


Fig. 5. Visualization of subsampling results; the subsampling rates are 25% in the second column and 12.5% in the third column.

2) Visualization of local grouping on point cloud shapes: In this section, we randomly select and visualize the local areas generated on the point cloud shapes from ShapeNet during the PSNet process. Their visualization results are shown in Fig. 6. The red points are a local area grouped by PSNet. The green point in a local area is the point selected as the subsampling point. Note that the points are depicted according to their actual positions in the 3-D space and a green point may be partially covered by some red points. As can be seen from Fig. 6, PSNet is able to effectively group the points with similar coordinates as a local area by learning from the input spatial features of individual points.

3) Visualization of local grouping on the point clouds generated from real scene: The point cloud data generated by scanning the real scenes may contain a large amount of noise. Therefore it is more difficult to handle these realistic data in training. In this section, we randomly select and visualize the generated local areas of the scenes from the S3DIS dataset during the PSNet process. Their visualization results are shown in Fig. 7. The coloring scheme in this figure is the same as

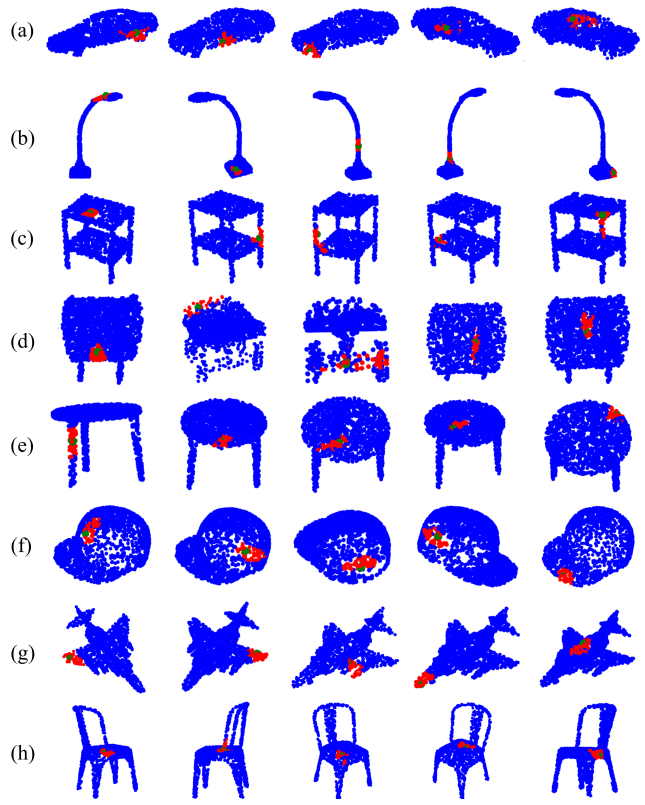


Fig. 6. Visualization of local grouping on point cloud shapes; a red area is a randomly selected local area after grouping.

that in Fig.6. As can be seen from Fig. 7, PSNet also works effectively for the realistic data.

4) Visualization comparison with other methods: In order to further understand the effectiveness of PSNet, in this subsection, we visualized the data structuring results obtained by PSNet and other sampling and grouping methods.

In particular, we visualized the sampling results obtained by the learning-based methods (*i.e.* SampleNet and CP-Net) and the widely used uniform sampling method FPS. The experimental results are shown in Fig. 8.

It can be seen that the points sampled by FPS are uniformly distributed, which causes FPS to equalize the points of different importance. For example, FPS sampled a large number of points for the bed base in Fig. 8(a), the piano cover in (b), the chair back in (c) and the desktop in (d). However, these areas are simple and easy to describe. On the contrary, the contour of the bed headboard in (a), the legs of the piano in (b), the contour of the armrests and the legs of the chair in (c) and the legs of the table in (d) have fewer points.

The sampling result of SampleNet is closer to that of FPS, because its training process is based on the shape uniformly sampled by FPS. However, some features are missed in SampleNet, such as the edge of the bed headboard in (a), the left side of the keyboard in (b), the lower part of the chair legs in (c) *et al.*

The sampling strategies of CP-Net and PSNet are similar since they both use the spatial features for sampling. Compared

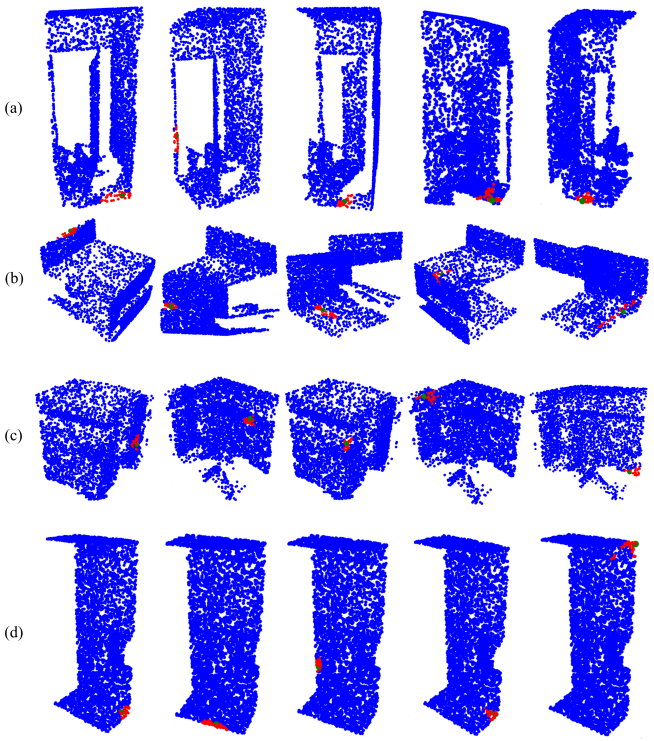


Fig. 7. Visualization of local grouping results for realistic data; a red area is a randomly selected local area after grouping.

with FPS and SampleNet, CP-Net and PSNet reduce the number of the sampled points in the simple planes and retain the sampling points for the skeleton and the contour of the shapes. Compared with PSNet, CP-Net loses some shape details, such as the left edge of the chair back in (c), the middle of the table leg in (d) and so on. In constrast, PSNet retains the points on the edge contour of the shapes and also the points that represent the skeleton and the detailed features of the shapes. This helps improve the effectiveness of the training model that PSNet is plugged into.

In Fig. 9, we visualized some grouping results achieved by PSNet and kNN. The grouping of kNN is based on the nearest points. The spatial feature correlation between the points in a group may be weak. Therefore, it is difficult for the grouped points to form a meaningful local shape. On the other hand, PSNet tends to allocate the points with similar spatial features into the same local area. For example, in Fig. 9(a), the local grouping of PSNet is a local part of the chair cushion; in (b), the local grouping of PSNet is a right-angle shape of the seat cushion. However, the points in the seat cushion and the points in part of the armrest are allocated by kNN to the same local area; in (c) and (d), the local grouping of PSNet retains the linear contour feature of the edge of the seat cushion, while kNN mixes the points in the armrest, the seat cushion and the side panel.

E. Robustness

We evaluated the robustness of PSNet to point permutation and rigid transformation. We performed random permutation,

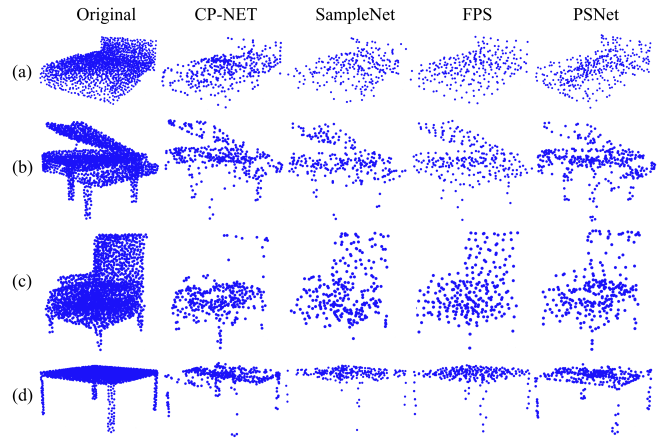


Fig. 8. The visualization comparison of the sampling results. The sampling rate is 25%.

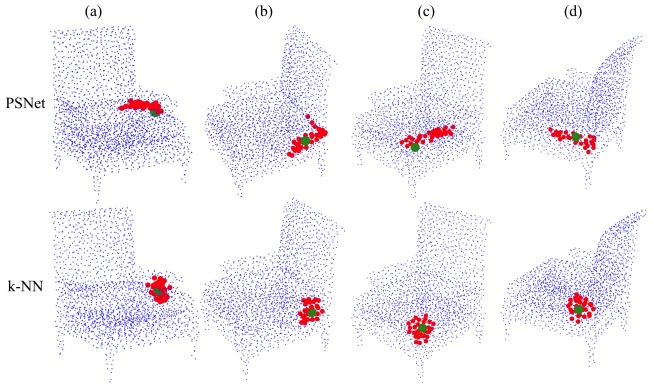


Fig. 9. The visualization comparison of the grouping results between PSNet and kNN. The green points are the sampling points of the local areas. The red points are the points in the local group.

translation and rotation on the test dataset and compared it with the original model. Table VII shows the results of running the shape classification task on RS-CNN with PSNet. The results show that the accuracy is not affected by the transformations.

	Original	Permutation	Translation	Rotation
RS-CNN with PSNet	92.6	92.6	92.6	92.6

TABLE VII
ROBUSTNESS EXPERIMENT OF PSNET.

F. Ablation Experiments

1) The Effect of θ and φ: When only using the Cartesian coordinates, our PSNet may group some symmetric points, which are distant from each other, in a point cloud into the same local area. Fig. 10 visualizes this phenomenon.

We evaluate the effectiveness of introducing θ and φ in Eq. (6) as the input spatial features of the points. Table VIII shows the performance (in terms of class mean IoU and instance mean IoU) of PointNet++ with PSNet in the part segmentation task. The performance with other tasks and networks show the similar trend.

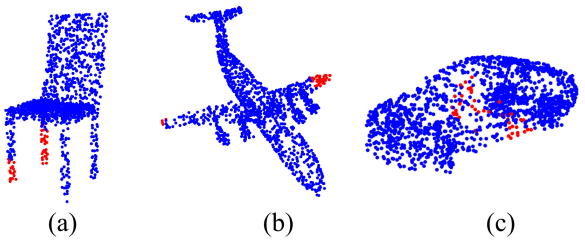


Fig. 10. Grouping error caused by the absence of θ and φ.

The results in Table VIII show that even when only the Cartesian coordinates are used, PSNet is still very effective. But adding θ and φ can further improve the effectiveness of PSNet. We examined the experimental records in detail. We found that when only using the Cartesian coordinates, PSNet may group some symmetric points with long distances into the same local area. We randomly checked 500 local areas of 20 symmetric shapes, and recorded the number of local areas which contain the points from distant symmetric parts (called *error areas*) and the number of points which are incorrectly grouped (called *error points*), and then calculated the grouping error rate. The values of the above records are listed in Table IX. As can be seen from the table, when only the Cartesian coordinates are used as the input spatial features (*i.e.* the "x, y, z" column in Table IX), there are 174 error areas (accounting for 34.8% of all 500 local areas), containing 1,431 error points. There are 64 points in each local area and 500*64=32000 points in all 500 areas. So the grouping error rate is 1431/32000=4.47%. After introducing θ and φ as two extra input spatial features, the errors are almost eradicated as shown in the table. In the experiments with θ and φ as additional input spatial features, however, the proportion is less than 1%. We also conducted the experiments with only spherical coordinates as the input spatial features. The results show that the effectiveness is significantly reduced (last row of Table VIII). The reason may be because it is easier to learn the features from the Cartesian coordinates. In addition, the x, y, z, Q column in Table VIII presents the error rate data

Spatial feature	Class mIoU	Instance mIoU
$[x, y, z]$	81.8	84.4
$[x, y, z, \theta, \varphi]$	**82.1**	**84.8**
$[r, \theta, \varphi]$	77.4	80.3

TABLE VIII
THE EFFECTIVENESS OF θ AND φ IN PSNET. THE PART SEGMENTATION TASK WAS RUN ON POINTNET++ WITH PSNET.

	x, y, z	x, y, z, θ, φ	x, y, z, Q
error areas	174	4	5
error points	1,431	5	7
grouping error rate	4.47%	0.02%	0.02%

TABLE IX
THE GROUPING ERROR RATES WITH AND WITHOUT INTRODUCING θ AND φ IN PSNET. 500 LOCAL AREAS ARE RANDOMLY SELECTED FROM 20 SYMMETRIC SHAPES WHEN RUNNING THE PART SEGMENTATION TASK ON POINTNET++ WITH PSNET. Q IS QUATERNIONS.

Data Structuring	Classification Accuracy(%)	Part Segmentation Class mIoU	Part Segmentation Instance mIoU
FPS + ball query	91.7	81.6	84.8
PSNet(Sampling) +ball query	91.9	81.8	84.8
PSNet	**92.2**	**82.1**	84.8

TABLE X
THE EFFECTIVENESS OF GROUPING IN PSNET; THE PSNET IS EMBEDDED INTO POINTNET++; THE DATA SET FOR THE SHAPE CLASSIFICATION TASK IS MODELNET40, AND THE DATA SET FOR PART SEGMENTATION IS SHAPENET.

for Quaternions [1] as an additional feature. The experimental results show that the effectiveness of Quaternions is similar to spherical coordinate. We used the spherical coordinate in the end because we think it is more intuitive.

2) Effectiveness of Sampling and Grouping: In this section, we conducted the experiments to evaluate the effectiveness of sampling and grouping in PSNet. We integrated PSNet into PointNet++ by replacing the original FPS sampling and ball query grouping in PointNet++ with our PSNet. We used PSNet-integrated PointNet++ to process shape classification and part segmentation tasks. PSNet performs sampling and local grouping at the same time. In order to evaluate the effectiveness of PSNet in sampling and local grouping separately, we also conducted the experiments in which PointNet++ only used PSNet for sampling, but still used its original ball query for local grouping. The experimental results are shown in Table X.

It can be seen from the first two rows of Table X that compared with the traditional sampling and grouping method (*i.e.* FPS+ball query), the combination of PSNet sampling and ball query achieves better performance. This indicates that the adaptive sampling of PSNet is more effective than FPS. By comparing the second and the third row of Table X, we can see that PSNet achieves better performance than the combination of PSNet sampling and ball query, which suggests that PSNet grouping is more effective than ball query. This result supports our argument that using the Euclidean distance (such as in ball query) as the only metric for grouping decisions may not be the best solution. Our PSNet can adjust the division of local areas adaptively based on local features, rather than on a single heuristic metric such as the distance, which we believe is the reason why PSNet achieved better performance.

3) The Channels of MLP: We compare the impact of the number of channels and the number of layers in MLP on the effectiveness of PSNet. Table XI shows the performance of running the part segmentation task with PointNet++ with PSNet. The performance with other tasks and networks show the similar trend. The results show that PSNet implemented with only one layer MLP (*i.e.* $[5, 32, s]$) is already effective. Increasing to two layers ($[5, 32, 128, s]$) can further improve the effectiveness slightly. But increasing the number of channels or layers further does not bring further benefit.

[1] https://en.wikipedia.org/wiki/Quaternion

Channels of MLPs	Class mIoU	Instance mIoU
$[5, 32, s]$	82.0	84.6
$[5, 32, 128, s]$	**82.1**	**84.8**
$[5, 64, 256, s]$	**82.1**	**84.8**
$[5, 32, 128, 256, s]$	**82.1**	**84.8**

TABLE XI
THE IMPACT OF THE NUMBER OF CHANNELS AND THE NUMBER OF LAYERS IN MLP; THE PART SEGMENTATION TASK IS RUN ON POINTNET++ WITH PSNET.

V. CONCLUSION

In this paper, we propose a novel and fast data structuring method called PSNet for deep learning of point clouds. The way in which PSNet structures the point cloud data can significantly improve the training and inference speed without affecting the accuracy of the original model. The sampling and grouping in PSNet can be embarrassingly parallelized. Moreover, the sampling and grouping are performed at the same time in PSNet, while the current mainstream methods perform sampling and grouping in sequence as two separate processes. Because of these features, PSNet can work for the deep-learning of larger scale point clouds than those in literature. Moreover, PSNet can work with the deep learning networks in a plug and play manner. We believe PSNet can also be applied to various non-Euclidean data structures other than point clouds, which we plan to investigate in future.

APPENDIX A
USING PSNET IN A PLUG AND PLAY FASHION

In this section, we present how we can plug and play PSNet in an existing point cloud deep learning model, PointNet++. The reason why we use the modification of PointNet++ as the example is because most hierarchical point cloud deep learning models are based on the PointNet++ architecture.

Fig. 11 is the screen capture of the code snippet that shows how to replace the original data structuring method in the PyTorch implementation of PointNet++ with our PSNet. Only a line of code (the highlighted line) has to be modified to plug PSNet.

In the code snippet in Fig. 11, Class $PointNetSetAbstraction$ is the main part of the feature abstraction component of the model. Each feature abstraction layer will construct an instance of this class. $PointNetSetAbstraction$ is mainly composed of three parts: data structuring, feature transformation and feature aggregation. In the original model, the data structuring method (*i.e.* $sample\_and\_group$ in the highlighted line) is FPS (for subsampling) and ball query (for local grouping). The module lists, mlp_convs and mlp_bns, are the operator sets for feature transformation. The *max* function is used for feature aggregation. When plugging PSNet, the data structuring method in our PSNet module (*i.e.* $psn.PSNet()$) is assigned to the $sample\_and\_group$ method. The input parameters in $psn.PSNet()$ are used to initialize the PSNet network. The parameter $npoint$ specifies the number of sampling points, which is also the number of local areas (*i.e.* s in the main paper), $nsample$ specifies the number of points in a local area (*i.e.* n in the main paper), the parameter $[32, 128]$ means that there are two hidden layers in the PSNet network, and the numbers of channels in these two layers are 32 and 128, respectively. Note that the numbers of channels in the input and output layers of the PSNet network are 5 (5 input spatial features) and s, respectively.

```
class PointNetSetAbstraction(nn.Module):
    def __init__(self, npoint, radius, nsample, in_channel, mlp, group_all):
        super(PointNetSetAbstraction, self).__init__()
        self.npoint = npoint
        self.radius = radius
        self.nsample = nsample
        self.mlp_convs = nn.ModuleList()
        self.mlp_bns = nn.ModuleList()
        last_channel = in_channel
        for out_channel in mlp:
            self.mlp_convs.append(nn.Conv2d(last_channel, out_channel, 1))
            self.mlp_bns.append(nn.BatchNorm2d(out_channel))
            last_channel = out_channel
        self.group_all = group_all
        self.sample_and_group = FPS_and_ballquery

    def forward(self, xyz, points):
        xyz = xyz.permute(0, 2, 1)
        if points is not None:
            points = points.permute(0, 2, 1)

        if self.group_all:
            new_xyz, new_points = sample_and_group_all(xyz, points)
        else:
            new_xyz, new_points = self.sample_and_group(
                self.npoint, self.radius, self.nsample, xyz, points)
        new_points = new_points.permute(0, 3, 2, 1)
        for i, conv in enumerate(self.mlp_convs):
            bn = self.mlp_bns[i]
            new_points = F.relu(bn(conv(new_points)))

        new_points = torch.max(new_points, 2)[0]
        new_xyz = new_xyz.permute(0, 2, 1)
        return new_xyz, new_points
```

```
class PointNetSetAbstraction(nn.Module):
    def __init__(self, npoint, radius, nsample, in_channel, mlp, group_all):
        super(PointNetSetAbstraction, self).__init__()
        self.npoint = npoint
        self.radius = radius
        self.nsample = nsample
        self.mlp_convs = nn.ModuleList()
        self.mlp_bns = nn.ModuleList()
        last_channel = in_channel
        for out_channel in mlp:
            self.mlp_convs.append(nn.Conv2d(last_channel, out_channel, 1))
            self.mlp_bns.append(nn.BatchNorm2d(out_channel))
            last_channel = out_channel
        self.group_all = group_all
        self.sample_and_group = psn.PSNet(npoint, nsample, [32, 128])

    def forward(self, xyz, points):
        xyz = xyz.permute(0, 2, 1)
        if points is not None:
            points = points.permute(0, 2, 1)

        if self.group_all:
            new_xyz, new_points = sample_and_group_all(xyz, points)
        else:
            new_xyz, new_points = self.sample_and_group(
                self.npoint, self.radius, self.nsample, xyz, points)
        new_points = new_points.permute(0, 3, 2, 1)
        for i, conv in enumerate(self.mlp_convs):
            bn = self.mlp_bns[i]
            new_points = F.relu(bn(conv(new_points)))

        new_points = torch.max(new_points, 2)[0]
        new_xyz = new_xyz.permute(0, 2, 1)
        return new_xyz, new_points
```

Fig. 11. An example of using the plug and play PSNet in PointNet++.

Note that when PSNet is assigned to $sample\_and\_group$, the interface of invoking $sample\_and\_group$ remains the same although the input parameter $radius$ (*i.e.* the radius of the ball query) in $sample\_and\_group$ is actually not needed (which is the parameter for the ball query). This is on purpose so that the programmers only need to change one line of code in the original training model for using PSNet.

ACKNOWLEDGMENT

This work was support in part by the National Key R&D Program of China under Grant 2018YFB2101504, in part by the National Natural Science Foundation of China under Grant 61672473, in part by the Key Research and Development Program of Shanxi Province of China under Grant 201803D121081 and Grant 201903D121147, and in part by the Natural Science Foundation of Shanxi Province of China under Grant 201901D111150.

REFERENCES

[1] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: Going beyond euclidean data," *IEEE Signal Process. Mag.*, vol. 34, no. 4, pp. 18–42, Jul 2017.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun 2016, pp. 770–778.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Comm. ACM*, vol. 60, no. 6, pp. 84–90, May 2017.

[4] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Represent.*, 2015.

[5] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun 2015, pp. 1–9.

[6] Y. Pei, Y. Huang, and X. Zhang, "Consistency guided network for degraded image classification," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 31, no. 6, pp. 2231–2246, Jun 2021.

[7] J. Ji, R. Shi, S. Li, P. Chen, and Q. Miao, "Encoder-decoder with cascaded CRFs for semantic segmentation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 31, no. 5, pp. 1926–1938, May 2021.

[8] Y. Ben-Shabat, M. Lindenbaum, and A. Fischer, "3dmfv: Three-dimensional point cloud classification in real-time using convolutional neural networks," *IEEE Robot. Autom. Lett.*, vol. 3, no. 4, pp. 3145–3152, Oct 2018.

[9] A. Brock, T. Lim, J. M. Ritchie, and N. J. Weston, "Generative and discriminative voxel modeling with convolutional neural networks," in *Neural Inf. Process. Conf.: 3D Deep Learning*, 2016.

[10] D. Maturana and S. Scherer, "VoxNet: A 3d convolutional neural network for real-time object recognition," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Sep 2015, pp. 922–928.

[11] G. Riegler, A. O. Ulusoy, and A. Geiger, "OctNet: Learning deep 3d representations at high resolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul 2017, pp. 6620–6629.

[12] D. Z. Wang and I. Posner, "Voting for voting in online point cloud object detection," in *Robotics: Science and Systems XI*, vol. 11, Jul 2015.

[13] P.-S. Wang, Y. Liu, Y.-X. Guo, C.-Y. Sun, and X. Tong, "O-CNN: Octree-based convolutional neural networks for 3d shape analysis," *ACM Trans. Graph.*, vol. 36, no. 4, pp. 1–11, Jul 2017.

[14] Y. Zhou and O. Tuzel, "VoxelNet: End-to-end learning for point cloud based 3d object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun 2018, pp. 4490–4499.

[15] Özgün Çiçek, A. Abdulkadir, S. S. Lienkamp, T. Brox, and O. Ronneberger, "3d u-net: Learning dense volumetric segmentation from sparse annotation," in *Proc. Int. Conf. Med. Image Comput. Comput.-Assist. Intervent. (MICCAI)*, 2016, pp. 424–432.

[16] J. Deng, W. Zhou, Y. Zhang, and H. Li, "From multi-view to hollow-3d: Hallucinated hollow-3d r-CNN for 3d object detection," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 31, no. 12, pp. 4722–4734, Dec 2021.

[17] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017.

[18] R. Q. Charles, H. Su, M. Kaichun, and L. J. Guibas, "PointNet: Deep learning on point sets for 3d classification and segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul 2017, pp. 77–85.

[19] L. Zhao, J. Guo, D. Xu, and L. Sheng, "Transformer3d-det: Improving 3d object detection by vote refinement," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 31, no. 12, pp. 4735–4746, Dec 2021.

[20] T. Sun, G. Liu, R. Li, S. Liu, S. Zhu, and B. Zeng, "Quadratic terms based point-to-surface 3d representation for deep learning of point cloud," *IEEE Trans. Circuits Syst. Video Technol.*, pp. 1–1, 2021.

[21] A. Komarichev, Z. Zhong, and J. Hua, "A-CNN: Annularly convolutional neural networks on point clouds," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun 2019, pp. 7421–7430.

[22] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen, "Pointcnn: Convolution on x-transformed points," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, 2018, pp. 828–838.

[23] Y. Liu, B. Fan, G. Meng, J. Lu, S. Xiang, and C. Pan, "DensePoint: Learning densely contextual representation for efficient point cloud processing," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, oct 2019, pp. 5239–5248.

[24] Y. Liu, B. Fan, S. Xiang, and C. Pan, "Relation-shape convolutional neural network for point cloud analysis," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun 2019, pp. 8895–8904.

[25] J. Mao, X. Wang, and H. Li, "Interpolated convolutional networks for 3d point cloud understanding," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct 2019, pp. 1578–1587.

[26] W. Wu, Z. Qi, and L. Fuxin, "PointConv: Deep convolutional networks on 3d point clouds," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun 2019, pp. 9621–9630.

[27] Y. Xu, T. Fan, M. Xu, L. Zeng, and Y. Qiao, "SpiderCNN: Deep learning on point sets with parameterized convolutional filters," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, vol. 11212, 2018, pp. 90–105.

[28] H. Zhao, L. Jiang, C.-W. Fu, and J. Jia, "PointWeb: Enhancing local neighborhood features for point cloud processing," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun 2019, pp. 5565–5573.

[29] C. Chen, G. Li, R. Xu, T. Chen, M. Wang, and L. Lin, "ClusterNet: Deep hierarchical cluster network with rigorously rotation-invariant representation for point cloud analysis," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun 2019, pp. 4994–5002.

[30] Y. Shen, C. Feng, Y. Yang, and D. Tian, "Mining point cloud local structures by kernel correlation and graph pooling," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun 2018, pp. 4548–4557.

[31] M. Simonovsky and N. Komodakis, "Dynamic edge-conditioned filters in convolutional neural networks on graphs," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul 2017, pp. 29–38.

[32] Z. Song, L. Zhao, and J. Zhou, "Learning hybrid semantic affinity for point cloud segmentation," *IEEE Trans. Circuits Syst. Video Technol.*, pp. 1–1, 2021.

[33] Y. Zhang and M. Rabbat, "A graph-CNN for 3d point cloud classification," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)*, Apr 2018, pp. 6279–6283.

[34] D. Li, G. Shi, Y. Wu, Y. Yang, and M. Zhao, "Multi-scale neighborhood feature extraction and aggregation for point cloud segmentation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 31, no. 6, pp. 2175–2191, Jun 2021.

[35] Y. Eldar, M. Lindenbaum, M. Porat, and Y. Zeevi, "The farthest point strategy for progressive image sampling," *IEEE Trans. Image Process.*, vol. 6, no. 9, pp. 1305–1315, Sep 1997.

[36] Z. Liu, H. Tang, Y. Lin, and S. Han, "Point-voxel cnn for efficient 3d deep learning," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019.

[37] L. Wang, Y. Huang, Y. Hou, S. Zhang, and J. Shan, "Graph attention convolution for point cloud semantic segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun 2019, pp. 10 296–10 305.

[38] J. Li, B. M. Chen, and G. H. Lee, "SO-net: Self-organizing network for point cloud analysis," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun 2018, pp. 9397–9406.

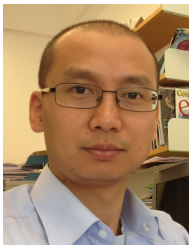
[39] R. Klokov and V. Lempitsky, "Escape from cells: Deep kd-networks for the recognition of 3d point cloud models," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct 2017, pp. 863–872.

[40] L. Landrieu and M. Simonovsky, "Large-scale point cloud semantic segmentation with superpoint graphs," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun 2018, pp. 4558–4567.

[41] Q. Hu, B. Yang, L. Xie, S. Rosa, Y. Guo, Z. Wang, N. Trigoni, and A. Markham, "RandLA-net: Efficient semantic segmentation of large-

scale point clouds," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun 2020, pp. 11 108–11 117.
[42] Q. Xu, X. Sun, C.-Y. Wu, P. Wang, and U. Neumann, "Grid-GCN for fast and scalable point cloud learning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun 2020, pp. 5661–5670.
[43] I. Lang, A. Manor, and S. Avidan, "SampleNet: Differentiable point cloud sampling," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun 2020, pp. 7578–7588.
[44] M. F. Balın, A. Abid, and J. Zou, "Concrete autoencoders: Differentiable feature selection and reconstruction," in *Proc. Int'l Conf. Machine Learning*, vol. 97, Jun 2019, pp. 444–453.
[45] J. Yang, Q. Zhang, B. Ni, L. Li, J. Liu, M. Zhou, and Q. Tian, "Modeling point clouds with self-attention and gumbel subset sampling," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun 2019, pp. 3323–3332.
[46] E. Nezhadarya, E. Taghavi, R. Razani, B. Liu, and J. Luo, "Adaptive hierarchical down-sampling for point cloud classification," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun 2020, pp. 12 956–12 964.
[47] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei, "Deformable convolutional networks," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct 2017, pp. 764–773.
[48] X. Chai, F. Shao, Q. Jiang, X. Meng, and Y.-S. Ho, "Monocular and binocular interactions oriented deformable convolutional networks for blind quality assessment of stereoscopic omnidirectional images," *IEEE Trans. Circuits Syst. Video Technol.*, pp. 1–1, 2021.
[49] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: A system for Large-Scale machine learning," in *Proc. USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, vol. 16, Savannah, GA, Nov 2016, pp. 265–283.
[50] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep 2020.
[51] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019.
[52] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," in *Proc. Int. Conf. Learn. Represent.*, 2017.
[53] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017.
[54] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3d ShapeNets: A deep representation for volumetric shapes," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun 2015, pp. 1912–1920.
[55] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu, "Shapenet: An information-rich 3d model repository," 2015. [Online]. Available: https://arxiv.org/abs/1512.03012
[56] G. Floros and B. Leibe, "Joint 2d-3d temporally consistent semantic segmentation of street scenes," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun 2012, pp. 2823–2830.

Luyang Li was born in China, in 1988. He received the master's degree in computer technology from North University of China (NUC), China, in 2014, where he is currently pursuing the Ph.D. degree in modeling and simulation of complex systems.



Ligang He (Member, IEEE) is a Reader in the Department of Computer at the University of Warwick. He has published more than 130 articles in international conferences and journals, such as the IEEE TC, TPDS, TACO, IPDPS, SC, and VLDB. His research interests focus on parallel and distributed processing, and big data processing.

Jinjin Gao was born in China, in 1988. She received the master's degree in computer technology from North University of China (NUC), China, in 2014, where she is currently a research associate in computer science and technology, Shanxi University of Finance and Economics.

Xie Han (Corresponding author) was born in China, in 1964. She received the master's degree in computer science and technology, North China Institute of Technology, China, and the Ph.D. degree from the Institute of Information Engineering, University of Science and Technology Beijing, China in 2002. She is currently a professor in computer science and technology, North University of China, China. Her current research interests include computer vision, simulation, and visualization.