PointVector: A Vector Representation In Point Cloud Analysis

Xin Deng* WenYu Zhang* Qing Ding† XinMing Zhang†
University of Science and Technology of China
{xin_deng, wenyuz}@mail.ustc.edu.cn, {dingqing, xinming}@ustc.edu.cn

Abstract

In point cloud analysis, point-based methods have rapidly developed in recent years. These methods have recently focused on concise MLP structures, such as PointNeXt, which have demonstrated competitiveness with Convolutional and Transformer structures. However, standard MLPs are limited in their ability to extract local features effectively. To address this limitation, we propose a Vector-oriented Point Set Abstraction that can aggregate neighboring features through higher-dimensional vectors. To facilitate network optimization, we construct a transformation from scalar to vector using independent angles based on 3D vector rotations. Finally, we develop a PointVector model that follows the structure of PointNeXt. Our experimental results demonstrate that PointVector achieves state-of-the-art performance 72.3% mIOU on the S3DIS Area 5 and 78.4% mIOU on the S3DIS (6-fold cross-validation) with only 58% model parameters of PointNeXt. We hope our work will help the exploration of concise and effective feature representations. The code will be released soon.

1. Introduction

Point cloud analysis is a cornerstone of various downstream tasks. With the introduction of PointNet [25] and PointNet++ [26], the direct processing of unstructured point clouds has become a hot topic. Many point-based networks introduced novel and sophisticated modules to extract local features, e.g., attention-based methods [52] explore attention mechanisms as Fig.1a with lower consumption, convolution-based methods [36] explore the dynamic convolution kernel as Fig.1c, and graph-based methods [39] [53] use graph to model relationships of points. The application of these methods to the feature extraction module of PointNet++ brings an improvement in feature quality. However, they are somewhat complicated to design in terms of network structure. PointNeXt [28] adopts the SetAbstrac-

*Co-first authors with equal contribution to refining the theory and experimental design
†Corresponding authors

Figure 1. Illustrations of the core operations of the different methods. (a) The features of each point are calculated separately by applying a fixed/isotropic kernel (black arrow) like Linear layer. Then, it imparts anisotropy by weights generated from inputs. (b) The displacement vector is used to filter points that approximate the kernel pattern for features aggregation. (c) It applies unique dynamic kernels with anisotropy for each point feature. (d) Differently, we generate vector representations based on features, and the aggregation methods for vectors are anisotropic due to the direction of the vectors.

PointNeXt uses all standard MLPs, which has insufficient feature extraction capability. In addition to attention and dynamic convolution mechanisms, template-based methods as Fig.1b such as 3D-GCN [19] employ relative displacement vectors to modulate the association between input points and the convolutional kernel. We introduce a vector representation of features to extend the range of feature variation with the intention of more effectively regulating the connections between local features. Our approach as Fig.1d differs from template-based methods. Instead of using displacement vectors as a property of the kernel, we
generate a vector representation for each neighboring point and aggregate them. Our method introduces less inductive bias, resulting in improved generalization capabilities. Furthermore, we enhance the generation of 3D vector representations by utilizing a vector rotation matrix with two independent angles in 3D space. This method facilitates the network to find the better solution.

Influenced by PointNeXt [28] and PointNet++ [26], we present the VP SA module. This module adheres to the structure of Point Set Abstraction (SA) module of the PointNet series. Vector representations are obtained from input features and aggregated using a reduction function. The vector of each channel is then projected into a scalar to derive local features. By combining VP SA and SA modules, we construct a PointVector model with an architecture akin to that of PointNeXt.

Our model undergoes comprehensive validation on public benchmark datasets. It achieves state-of-the-art performance on the S3DIS [1] semantic segmentation benchmark and competitive results on the ScanObjectNN [47] and ShapeNetPart [48] datasets. By incorporating a priori knowledge of vectors, our model attains superior results with fewer parameters on S3DIS. Detailed ablation experiments further demonstrate the efficacy of our methodology. The contributions are summarized below:

- We propose a novel immediate vector representation with relative features and positions to better guide local feature aggregation.
- We explore the method of obtaining vector representation and propose the generation method of 3D vector by utilizing the vector rotation matrix in 3D space.
- Our proposed PointVector model achieves 72.3% mean Intersection over Union (mIOU) on S3DIS area5 and 78.4% mIOU on S3DIS (6-fold cross-validation) with only 58% model parameters of PointNeXt.

2. Related work

Point-based network. In contrast to the voxelization [54] [15] [31] and multiview [32] [10] [41] methods, point-based methods deal directly with point clouds. PointNet first proposes using MLP to process point clouds directly. PointNet++ subsequently introduces a hierarchical structure to improve the feature extraction. Subsequent works focused on the design of fine-grained local feature extractors. Graph-based methods [39] [38] rely on a graph neural network and introduce point features and edge features to model local relationships. Conv-based methods [36] [45] [42] [2] [17] propose several dynamic convolution kernels to adaptively aggregate neighborhood features. Many transformer-like networks [11] [50] [49] [9] [14] extract local features with self-attention. Recently, MLP-like networks are able to obtain good results with simple networks by enhancing the features. PointMLP [24] proposes a geometric affine module to normalize the feature. RepSurf [30] fits the surface information through the triangular plane, models umbrella surfaces to provide geometric information. PointNeXt [28] integrates training strategies and model scaling.

MLP-like Architecture. The MLP-like structure has recently shown the ability to rival the Transformer with simple architecture. In the image field, MLP-Mixer [37] first use the combination of Spatial MLP and Channel MLP. The subsequent works [3] [18] reduce computational complexity by selecting objects for the spatial MLP while maintaining a large perceptual field to preserve accuracy. Since the point cloud is too large, the MLP-like network determines the perceptual field generally using K-Nearest neighbor sampling or ball sampling methods. The MLP structure in point cloud analysis starts with PointNet [25] and PointNet++ [26], using MLPs to extract features and aggregating them by symmetric functions. Point-Mixer [6] proposes three point-set operators, PointMLP [24] to modify the distribution of features by geometric affine module, and PointNeXt [28] to scale up the PointNet++ model and improve the performance using by training strategies and model scaling.

Feature Aggregation. PosPool [21] improves the reduction function defined in PointNet++ by providing a parameter-free position-adaptive pooling operation. ASANet [27] introduces a new anisotropic reduction function. Also, the introduction of the attention mechanism [46] provides new dynamic weights for the reduction function. Vectors have direction, and this property is naturally satisfied for anisotropic aggregation functions. GeoCNN [4] projects features based on vectors and angles of neighbor points and centroids in six directions and sums them. WaveMLP [35] represents image patches as waves and describes feature aggregation using wave phase and amplitude. The Vector Neuron [7] constructs a triad of neurons to reconstruct standard neural networks and represent features through vector transformations. The template-based methods represented by 3DGCN [19] uses the cosine value of the relative displacement vectors to filter for aggregation features from neighbors that more conform to the pattern of the kernel. Local displacements [40] use local displacement vectors to update features by combining the weights of fixed kernels. In our method, an intermediate vector representation is generated by modifying the point feature extraction function. The vector direction is determined based on both features and position to fulfill the anisotropic aggregation function.
3. Method

We propose an intermediate vector representation to enhance local feature aggregation in point cloud analysis. This section includes a review of the Point Set Abstraction(SA) operator of the PointNet family in Section 3.1, the presentation of our Vector-oriented Point Set Abstraction module in Section 3.2, a description of our method of extending vectors from scalars in Section 3.3, and the network structure of PointVector in Section 3.4.

3.1. Preliminary

The SA module include a grouping layer (K-NN or Ball-Query) to query each point’s neighbors, shared MLPs, and a reduction layer to aggregate neighbor features. The SA module has an subsample layer to downsample the point cloud in the first layer. We denote $f_i^{l+1}$ as the extracted feature of point $i$ after stage $l+1$, $N_i$ as the neighbors of point $i$ and $n$ is the number of incoming points. The content of the SA module can be formulated as follows:

$$f_i^{l+1} = R\{H\{[f_j^l, p_j - p_i]|j \in N_i\}\}, \quad (1)$$

where $R$ is the reduction function that aggregates features for point $i$ from its neighbors $N_i$ and $H$ means the shared MLPs. $f_j^l, p_j, p_i$ denote the input features of point $j$, the position of point $j$ and the position of point $i$, respectively.

In the local aggregation operation, the classical method assigns weights to components of c-dimensional features as shown in Eq.2 and sums the neighboring features in spatial dimensions. We consider the component $f_i$ of the c-dimensional feature $f$ as a base vector with only one non-zero value, and define the vector transformation as follows:

$$f_i \ast w = w f_i, i = 0 \cdots c, \quad (2)$$

$$\begin{bmatrix} w & 0 & \cdots & 0 \\ f_i & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} = \begin{bmatrix} w f_i & 0 & \cdots & 0 \end{bmatrix}, \quad (3)$$

where $w$ is the scalar weight. In Eq.3, the transformation changes one value of the vector. The two equations above are equivalent. Unchanged zeros in the equation do not contribute to subsequent operations and can be disregarded. In Physics, the degree of freedom of a motion is equal to the number of state quantities that the motion causes the system to change. A greater number of degrees of freedom in a physical system indicates a larger range of independent variation in the parameters that define its state. Similarly, the degrees of freedom of a vector transformation refer to the number of values in the vector that can change independently. So, the 3D vector we mentioned means the degrees of freedom of the vector transformation is 3.

3.2. Vector-oriented Point Set Abstraction

As discussed in Section 3.1, feature components can be represented as vectors. A higher degree of freedom in vector transformations allows for increased variation and improved representation of connections between neighboring elements. Vectors, with their size and direction properties, are more expressive than scalars for representing features. When aggregated, they exhibit anisotropy due to their directional nature. So, we introduce an intermediate vector representation as Fig.2.

It should be noted that in our assumptions, the component of a c-dimensional feature represents the projection of the feature vector along the c coordinate axes. After aggregating the vectors to obtain the $c \times 3$ centroid feature, where the number of changing values in the component vectors is 3. To merge them into a c-dimensional feature vector requires aligning the c components and then summing them. Due to the difficulty in implementing component alignment with this method, we directly project the c components into scalars and combine them into centroid features. Similar to the intermediate features in a convolutional network, the values on each channel’s feature map represent the response strength to a specific feature at that location.

The input features in our method are transformed into a series of vectors and then aggregated by the reduction function. Note that the element in each channel of the vector representation is vector. We obtain a vector representation that is channel independent. We denote $f p_j$ as a mixed feature of relative features $f_j - f_i$ and relative positions $p_j - p_i$. The content of the vector-guided aggregation module can be formulated as:

$$f_i^{l+1} = \eta(f_i^l) + H_v\{H_p\{R\{H_c(f p_j)|j \in N_i\}\}\}, \quad (4)$$

where $H_v$ is the function that generates the vector representation, $H_p$ denotes the projection Linear transform vector to a scalar, and $H_c$ is the channel mixing Linear that.
interacts with the information of each channel while transforming dimensions to fit the network. However, the feature representation we introduce is actually represented using a triplet form. We denote \( m \) as the dimension of the vector, and \( c \) is the channel of the feature. In fact, the set of \( c \) \( m \)-dimensional vectors is represented in the same form as the \((m \times c)\)-dimensional feature vectors. The reduction function is followed by a grouped convolution [13] that transforms the vectors to scalars for each channel, which distinguishes the intermediate vector representation from the general feature vector.

When the reduction function \( R \) selects sum, the \( R \) and \( H_p \) functions together constitute a special case of GroupConv [13]. Let \( k \) denote the number of neighbor features. For one group, the convolution kernel of GroupConv is a \( k \times m \times k \) parameter matrix, while our method can be viewed as \( k \) identical \( m \times 1 \) parameter matrices. This is because we treat vectors as wholes and assign equal weight to each element. We will explain in the supplementary material why the original groupconv operation is not suitable for our vector-guided feature aggregation.

### 3.3. Extended Vector From Scalar

The simplest idea for the \( H_v \) function defined in Eq.4 is to obtain \( c \) \( m \)-dimensional vectors of point \( j \) directly with MLPs. However, while single-layer MLPs may have limited expressive capability, multi-layer MLPs can be resource-intensive. As discussed in Section 3.1, input features are considered as vectors and we aim to design a transformation with high degrees of freedom. This transformation combines rotation and scaling, represented by a rotation matrix and a learnable parameter respectively. This method achieves better results with lower resource consumption.

As shown in Fig.3, a scalar can be directly converted into an \( m \)-dimensional vector by adding \( m-1 \) zero-value components. Each channel of the extended vector representation can then be considered as an \( m \)-dimensional vector along a specific coordinate axis direction. Therefore, we can obtain the proper vector direction by additionally training a rotation matrix. Directly predicting the rotation matrix can cause difficulties for nonlinear optimization because the matrix elements are interdependent. Instead, we first predict the rotation angle and then derive the rotation matrix based on this angle. The rotation of a 3D vector can be decomposed into rotations around three axes. However, we have not yet determined how to represent the rotation of a 4D vector around a plane. As shown in Fig.4, since the extended 3D vector is on the coordinate axis, one rotation around that axis can be omitted. We keep the default rotation direction as counterclockwise. The vector \( \vec{r} \) is first rotated around the \( x \)-axis by an angle \( \pi/2 - \beta \) and then rotated around the \( z \)-axis by an angle \( \alpha \) to finally obtain the vector \( \vec{r}_0 \). The rotation can be formulated as follows:

\[
\vec{r}_0 = \text{Rot}_z \text{Rot}_x \vec{r} = \begin{bmatrix} \cos(\alpha) - \sin(\alpha) & 0 & 0 \\ \sin(\alpha) \cos(\alpha) & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \sin(\beta) & -\cos(\beta) \\ 0 & \cos(\beta) & \sin(\beta) \end{bmatrix} \begin{bmatrix} 0 \\ xz \cos(\alpha) \sin(\beta) + zx \sin(\alpha) \sin(\beta) \\ xz \cos(\alpha) \cos(\beta) + zx \cos(\alpha) \cos(\beta) - \sin(\alpha) \sin(\beta) \end{bmatrix}
\]

where \( \text{Rot}_x, \text{Rot}_z \) denote the rotation matrix rotated around the \( x \)-axis and the rotation matrix rotated around the \( z \)-axis, respectively, and \( zx \) is generated by \text{Linear}. The independence of \( \alpha \) and \( \beta \) facilitates network optimization. Therefore, we can expand each scalar value of the features into a 3D vector according to Fig.4 and Eq.5. The feature aggregation in a local area is influenced by the relationship between neighboring points and centroids. Methods such as PointTransformer [52], PACConv [44], and AdaptConv [53] model this relationship using relative position and features. Our approach also extracts rotation angles using MLP on relative positions and features. The acquisition of the vector can be formulated as follows:

\[
zx_j = \text{Linear}(fp_j)
\]

\[
[\alpha_j, \beta_j] = \text{Relu}(BN(\text{Linear}([fp_j])))
\]

where \( fp_j \) denotes a mixed feature of relative features \( f_j - f_i \) and relative positions \( p_j - p_i \), and \( f_j \) means the fea-
Figure 5. Overall Architecture. We reuse the SA module and Feature Propagation module of PointNet++ and propose the VPSA module to improve the feature extraction of sampled point clouds.

3.4. Architecture

In summary, we propose PointVector, modified from PointNeXt [28] by replacing its InvResMLP module with our proposed VPSA module, we define its vector dimension \( m = 3 \). The architecture is illustrated in Fig.5. Referring to the classical PointNet++, we use a hierarchical structure containing an encoder and a decoder. For the segmentation task, we use an encoder and a decoder. For the classification task, we only use an encoder. For a fair comparison with PointNeXt, we set up three sizes of models with reference to the parameter settings of PointNeXt. We denote \( C \) as the channel of embedding MLP in the beginning, \( S \) as the numbers of the SA module, \( V \) as the numbers of the VPSA module. The three sizes of models are shown as follows:

- PointVector-S: \( C=32, S=0, V=[1,1,1,1] \)
- PointVector-L: \( C=32, S=[1,1,1,1], V=[2,4,2,2] \)
- PointVector-XL: \( C=64, S=[1,1,1,1], V=[3,6,3,3] \)

Since PointNeXt uses only the PointNeXt-S model for classification, we use our VPSA module instead of the SA module in PointVector-S for a fair comparison. The detailed structure of the classification tasks will appear in the supplementary material. There is a skip connection path in the VPSA module in Fig.5, which is added to the main path and then through a ReLU layer. The reason for using this summation method is that RepSurf [30] indicates how two features with different distributions should be combined. For the segmentation task, finer local information is needed, and we set reduction function as sum. For the classification task, which favors aggregating global information, we choose the original reduction function such as max.

4. Experiments

We evaluate our model on three standard benchmarks: S3DIS [1] for semantic segmentation and ScanObjectNN [47] for real-world object classification and ShapeNetPart [48] for part segmentation. Note that our model is implemented on the basis of PointNeXt. Since we use the training strategy provided by PointNeXt, we refer to the metrics reported by PointNeXt for a fair comparison.

Experimental setups. We train PointVector using CrossEntropy loss with label smoothing [33], AdamW optimizer [23], and initial learning rate \( lr=0.002, \) weight\_decay \( 10^{-4} \), with Cosine Decay, and a batch\_size of 32. The above are the base settings for all tasks, and specific parameters will be changed for specific tasks. We follow the train, valid, and test divisions for the dataset. The best model on the validation set will be evaluated on the test set. For S3DIS segmentation task, point clouds are downsampled with a voxel size of 0.4 m following previous methods [36] [27] [52]. The initial learning rate on this task is set to 0.01. For 100 epochs, we use a fixed 24000 points as a batch and set batch\_size to 8. During training, the input points are selected from the nearest neighbors of the random points. Similar to Point Transformer [52], we evaluate our model using the entire scene as input. For ScanObjectNN [47] classification task, we set the weight\_decay to 0.05 for 250 epochs. Following Point-BERT [50], the number of input points is 1024. The training points are randomly sampled from the point cloud, and the testing points are uniformly sampled.
during evaluation. The details of data augmentation are the same as those in PointNeXt. For ShapeNetPart part segmentation, we train PointVector-S with a batch size of 32 for 300 epochs. Following PointNet++ [26], 2,048 randomly sampled points with normals are used as input for training and testing.

For voting strategy [20], we keep it the same as PointNeXt and use it only on part segmentation task. To ensure a fair comparison with standard methods, we do not use any ensemble methods, such as SimpleView [8]. We also provide the model parameters (Params) and GFLOPs. We additionally, similarly to PointNeXt, provide throughput (instance per second) as an indicator of inference speed. The input data for the throughput calculation are kept consistent with PointNeXt for fair comparison. The throughput of all methods is measured using 128 × 1024 (batch size 128, number of points 1024) as input on ScanObjectNN and 64 × 2048 on ShapeNetPart. On S3DIS, 16 × 15,000 points are used to measure the throughput following [28] [27]. We evaluate our model using an NVIDIA Tesla V100 32 GB GPU and a 48 core Intel Xeon @ 2.10 Hz CPU.

4.1. 3D Semantic segmentation on S3DIS

S3DIS [1] (Stanford Large-Scale 3D Indoor Spaces) is a challenging benchmark composed of 6 large-scale indoor areas, 271 rooms, and 13 semantic categories in total. For our models in S3DIS, the number of neighbors in SetAbstraction is 32, and the number of neighbors in the Local Vector module is 8. PointTransformer [52] also employs most of the training strategies and data enhancements used by PointNeXt, so it is fair for us to compare with it. For a comprehensive comparison, we report the experimental results of PointVector-L and PointVector-XL on S3DIS with 6-fold cross-validation in Table 1 and S3DIS Area 5 in Table 2, respectively. As shown in table 1&2, we achieve state-of-the-art performance on both validation options. Table 1 shows that our largest mode PointVector-XL outperforms PointNeXt-XL by 1.6%, 3.1% and 3.5% in terms of overall accuracy (OA), mean accuracy (mAcc) and mIOU, respectively, while has only 58% Params. At the same time, the computational consumption of ours is only 69% of PointNeXt-XL in terms of GFLOPs. The reduction in computational consumption because the number of neighbors is reduced to 8. The limitation is that we make heavy use of GroupConv (groups=channel), which is not well optimized in PyTorch and is slower than standard convolution. Therefore, our inference speed is 6 instances/second lower than PointNeXt-XL. Our model shows better results at all sizes.

On S3DIS Area 5, we selected the best results reported by PointNeXt for comparison and did not repeat the experiment. Our PointVector-XL model outperforms StratifiedFormer [14] and PointNeXt-XL by 0.3% and 1.8% in mIOU, respectively. StratifiedFormer expands the scope of the query by combining high-resolution and low-resolution keys while efficiently extracting contextual information. Even though its receptive field is much wilder than our model, we still show a competitive performance. Additionally, there are some differences in the experimental setup between our model and it, in which it has 80k points of input, much larger than our 24k points of input. In addition it uses KPConv [36] instead of Linear in the first layer. It seems that these measures have significant effects. However, the comparison is not fair enough for us due to the difference of the experimental configurations. We will synchronize its experimental configuration later. Additionally, our models of the same size on Area 5 show better results than PointNeXt. PointVector-L and PointVector-XL perform better than PointNeXt-L and PointNeXt-XL by 1.7% and 1.5% in mIOU, respectively, and we performs better on most of categories.

4.2. 3D Object Classification on ScanObjectNN

ScanObjectNN [47] contains approximately 15,000 real scanned objects that are categorized into 15 classes with 2,902 unique object instances. The dataset has significant challenges due to occlusion and noise. As with PointNeXt, we chose the hardest variant PB,T50_RS of ScanObjectNN and report the mean±std Overall Accuracy and Mean Accuracy score. For our model in ScanObjectNN, the number of neighbors in SetAbstraction is 32. As shown in table 3, our PointVector-S model achieves a comparable performance on ScanObjectNN in OA, while outperforms PointNeXt-S by 0.4% in mAcc. This illustrates that our approach is not more biased toward certain categories and is relatively robust. Our approach is at a disadvantage in terms of speed and scale compared to the SA module. Since we introduce high-dimensional vectors, we generate more computations before the reduction compared to the standard SA module. Due to group convolution operations and trigonometric functions, there is a speed bottleneck. Although the inference speed is slower than PointNeXt, we are still faster

<table>
<thead>
<tr>
<th>Method</th>
<th>OA %</th>
<th>mAcc %</th>
<th>mIOU %</th>
<th>Params</th>
<th>GFLOPs</th>
<th>Throughput (ins/sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PointNet [25]</td>
<td>78.5</td>
<td>66.2</td>
<td>47.6</td>
<td>3.6</td>
<td>35.5</td>
<td>162</td>
</tr>
<tr>
<td>PointCNN [17]</td>
<td>88.1</td>
<td>75.6</td>
<td>65.4</td>
<td>0.6</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>DGCNN [19]</td>
<td>84.1</td>
<td>-</td>
<td>56.1</td>
<td>1.3</td>
<td>-</td>
<td>8</td>
</tr>
<tr>
<td>DeepGCN [14]</td>
<td>85.9</td>
<td>-</td>
<td>60.0</td>
<td>3.6</td>
<td>-</td>
<td>3</td>
</tr>
<tr>
<td>KPConv [36]</td>
<td>-</td>
<td>79.1</td>
<td>70.6</td>
<td>15.0</td>
<td>-</td>
<td>30</td>
</tr>
<tr>
<td>RandLA-Net [12]</td>
<td>88.0</td>
<td>82.0</td>
<td>70.0</td>
<td>1.3</td>
<td>5.8</td>
<td>159</td>
</tr>
<tr>
<td>Point Transformer [52]</td>
<td>7.8</td>
<td>81.9</td>
<td>73.5</td>
<td>7.8</td>
<td>5.6</td>
<td>34</td>
</tr>
<tr>
<td>CBL [53]</td>
<td>89.6</td>
<td>79.4</td>
<td>73.1</td>
<td>18.6</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>RepSurf [30]</td>
<td>90.9</td>
<td>82.6</td>
<td>74.3</td>
<td>0.976</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>PointNet++ [26]</td>
<td>81.0</td>
<td>67.1</td>
<td>54.5</td>
<td>1.0</td>
<td>7.2</td>
<td>186</td>
</tr>
<tr>
<td>PointNeXt-L [28]</td>
<td>89.8</td>
<td>82.2</td>
<td>73.9</td>
<td>7.1</td>
<td>15.2</td>
<td>115</td>
</tr>
<tr>
<td>PointNeXt-XL [28]</td>
<td>90.3</td>
<td>83.0</td>
<td>74.9</td>
<td>41.6</td>
<td>84.8</td>
<td>46</td>
</tr>
<tr>
<td>PointVector-L</td>
<td>91.4</td>
<td>85.5</td>
<td>77.4</td>
<td>4.2</td>
<td>10.7</td>
<td>98</td>
</tr>
<tr>
<td>PointVector-XL(Ours)</td>
<td>91.9</td>
<td>86.1</td>
<td>78.4</td>
<td>24.1</td>
<td>58.5</td>
<td>40</td>
</tr>
</tbody>
</table>

Table 1. Semantic segmentation on S3DIS with 6-fold cross-validation. Methods are in chronological order. The highest and second scores are marked in bold.
We perform ablation experiments at S3DIS to verify the effectiveness of the module, and because PointVector-XL is too large, we make changes to PointVector-L. To make the comparison fair, we did not change the training parameters. 

Table 3. Object classification on ScanObjectNN. * denotes StratifiedFormer use 80k points as input points. The highest and second scores are marked in bold.

<table>
<thead>
<tr>
<th>Method</th>
<th>OA</th>
<th>mAcc</th>
<th>mIoU</th>
<th>ceiling</th>
<th>floor</th>
<th>wall</th>
<th>beam</th>
<th>column</th>
<th>window</th>
<th>door</th>
<th>table</th>
<th>chair</th>
<th>sofa</th>
<th>bookcase</th>
<th>board</th>
<th>clutter</th>
</tr>
</thead>
<tbody>
<tr>
<td>PointNet [25]</td>
<td>68.2</td>
<td>63.4</td>
<td>3.5</td>
<td>4112</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PointCNN [17]</td>
<td>78.5</td>
<td>75.1</td>
<td>0.6</td>
<td>44</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DGCNN [39]</td>
<td>78.1</td>
<td>73.6</td>
<td>1.8</td>
<td>402</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GBNNet [29]</td>
<td>80.5</td>
<td>77.8</td>
<td>8.8</td>
<td>194</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRANet [5]</td>
<td>82.1</td>
<td>79.1</td>
<td>2.3</td>
<td>493</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PointMLP [24]</td>
<td>85.4±1.3</td>
<td>83.9±1.5</td>
<td>13.2</td>
<td>191</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RepSurf-U [30]</td>
<td>86.0</td>
<td>83.1</td>
<td>6.8</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PointNet++ [26]</td>
<td>77.9</td>
<td>75.4</td>
<td>1.5</td>
<td>1872</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PointNeXt-S [28]</td>
<td>87.7±0.4</td>
<td>85.8±0.6</td>
<td>1.4</td>
<td>2040</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PointVector-S(Ours)</td>
<td>87.8±0.4</td>
<td>86.2±0.5</td>
<td>1.55</td>
<td>901</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3. Object classification on ScanObjectNN. * denotes StratifiedFormer use 80k points as input points. The highest and second scores are marked in bold.

4.3. 3D Object Part Segmentation on ShapeNetPart

ShapeNetPart [48] is an object-level dataset for part segmentation. It consists of 16,880 models from 16 different shape categories, 2-6 parts for each category, and 50 part labels in total. As shown in Tab.4, our PointVector-S and PointVector-S-C64 models both achieve results that are comparable to PointNeXt. For the PointNeXt-S model with C=160, the number of parameters is large, and we do not give a corresponding version of the model.

4.4. Ablation Study

We perform ablation experiments at S3DIS to verify the effectiveness of the module, and because PointVector-XL is too large, we make changes to PointVector-L. To make the comparison fair, we did not change the training parameters.

Table 4. Object Part Segmentation on ShapeNetPart. *Our evaluation results on this task alone are not consistent with the throughput results derived from that paper. Other works we did not test one by one.

<table>
<thead>
<tr>
<th>Method</th>
<th>Ins.mIoU</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>PointNet [25]</td>
<td>83.7</td>
<td>1184</td>
</tr>
<tr>
<td>DGCNN [39]</td>
<td>85.2</td>
<td>147</td>
</tr>
<tr>
<td>KPConv [36]</td>
<td>86.2</td>
<td>44</td>
</tr>
<tr>
<td>PRANet [5]</td>
<td>85.1</td>
<td>-</td>
</tr>
<tr>
<td>CurveNet [43]</td>
<td>86.8</td>
<td>97</td>
</tr>
<tr>
<td>ASSANet-L [27]</td>
<td>86.1</td>
<td>640</td>
</tr>
<tr>
<td>Point Transformer [52]</td>
<td>86.6</td>
<td>297</td>
</tr>
<tr>
<td>PointMLP [24]</td>
<td>86.1</td>
<td>270</td>
</tr>
<tr>
<td>Stratifiedformer [14]</td>
<td>86.6</td>
<td>398</td>
</tr>
<tr>
<td>PointNeXt++ [26]</td>
<td>85.1</td>
<td>560</td>
</tr>
<tr>
<td>PointNeXt-S* (C=64)</td>
<td>86.5</td>
<td>446</td>
</tr>
<tr>
<td>PointNeXt-S* (C=160)</td>
<td>87.2</td>
<td>75</td>
</tr>
<tr>
<td>PointVector-S(Ours)</td>
<td>86.5</td>
<td>446</td>
</tr>
<tr>
<td>PointVector-S-C64(Ours)</td>
<td>86.9</td>
<td>211</td>
</tr>
</tbody>
</table>
Table 5. Core operation of VPSA. We abstract the module into sum and GroupConv operations, and replace this part. FC means Channel-FC as Linear. * means it acts as a baseline.

<table>
<thead>
<tr>
<th>Method</th>
<th>OA</th>
<th>mAcc</th>
<th>mIOU</th>
<th>Params</th>
</tr>
</thead>
<tbody>
<tr>
<td>max+FC*</td>
<td>90.6</td>
<td>76.4</td>
<td>70.6</td>
<td>6.35</td>
</tr>
<tr>
<td>Conv</td>
<td>90.4</td>
<td>75.7</td>
<td>69.4</td>
<td>24.56</td>
</tr>
<tr>
<td>GroupConv</td>
<td>90.6</td>
<td>76.5</td>
<td>70.8</td>
<td>4.76</td>
</tr>
<tr>
<td>sum+FC</td>
<td>90.7</td>
<td>76.6</td>
<td>71.0</td>
<td>6.35</td>
</tr>
<tr>
<td>max+GroupConv</td>
<td>90.6</td>
<td>76.2</td>
<td>70.6</td>
<td>4.71</td>
</tr>
<tr>
<td>sum+GroupConv</td>
<td>90.8</td>
<td>77.3</td>
<td>71.2</td>
<td>4.71</td>
</tr>
</tbody>
</table>

Table 6. Methods for obtaining vector representations.

<table>
<thead>
<tr>
<th>Method</th>
<th>OA</th>
<th>mAcc</th>
<th>mIOU</th>
<th>Params</th>
</tr>
</thead>
<tbody>
<tr>
<td>MLP</td>
<td>91.0</td>
<td>76.5</td>
<td>70.8</td>
<td>5.55</td>
</tr>
<tr>
<td>Linear+direction</td>
<td>90.8</td>
<td>76.6</td>
<td>70.8</td>
<td>5.55</td>
</tr>
<tr>
<td>Linear+rotation</td>
<td>90.8</td>
<td>77.3</td>
<td>71.2</td>
<td>4.71</td>
</tr>
</tbody>
</table>

Table 7. Different dimensional vector.

<table>
<thead>
<tr>
<th>Method</th>
<th>OA</th>
<th>mAcc</th>
<th>mIOU</th>
<th>Params</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalar</td>
<td>90.4</td>
<td>76.1</td>
<td>69.8</td>
<td>3.87</td>
</tr>
<tr>
<td>2D vector</td>
<td>90.4</td>
<td>77.2</td>
<td>70.9</td>
<td>3.9</td>
</tr>
<tr>
<td>3D vector</td>
<td>90.8</td>
<td>77.3</td>
<td>71.2</td>
<td>4.7</td>
</tr>
</tbody>
</table>

Vector dimension. We need to explore the connection between the effect of vector representation and dimension. Intuitively, higher dimensional vectors will be more expressive of features than lower dimensional vectors. Tab.7 shows that the 3D vector has a better ability to express the features and that the increase in the number of parameters is not very large. The mIOU without our vector representation is still higher than the results of PointNeXt. We will discuss the validity of the other parts of our network in the supplementary material.

Robustness. Table 8 shows that our method is extremely robust to various perturbations as Stratified Transformer. The ball query we use cannot get the same neighbors in the scaled point cloud. If the query radius is scaled together, then mIOU is invariant. It indicates that our method also has scale invariance.

Table 8. Robustness study on S3DIS (mIOU %). We apply z-axis rotation ($\frac{\pi}{2}$, $\pi$, $\frac{3\pi}{2}$), shifting ($\pm0.2$), scaling ($\times0.8$, $\times1.2$) and jitter in testing. PointTr: Point Transformer. Stratified: Stratified Transformer.

Extended Vector From Scalar. To verify the effectiveness of our vector rotation-based method, we compare it with two other methods. As shown in Tab.6, MLP is represented by two Linear layers and a ReLU activation and BatchNorm layers. Linear+direction means that Linear predicts the vector modulus length, then uses MLP to obtain the unit vector as direction, and the final modulus length is multiplied by the unit vector. The rotation-based vector expansion method proposed in Section 3.3 is ahead of other methods and has fewer parameters. This shows that the rotation-based approach can use fewer parameters to obtain a vector representation more suitable for neighbor features.

5. Conclusion and Limitation.

We introduce PointVector, which achieves state-of-the-art results on the S3DIS semantic segmentation task. Our vector-oriented point set abstraction improves local feature aggregation with fewer parameters. The rotation-based vector expansion method bridges the gap between vector representation and standard feature forms. By optimizing two independent perspectives, it achieves better results. Additionally, our method exhibits robustness to various perturbations. It is noteworthy that further exploration of vector representation’s meaning may reveal additional applications, i.e. dominant neighbor selection.

The speed of our approach is constrained by the grouped convolution implementation. An interesting avenue for future work includes exploring rotations above three dimensions and decomposing four-dimensional rotations into combinations of plane rotations. Additionally, summing after component alignment aligns with our assumptions better than scalar projection.

Acknowledgement

This work was supported in part by the National Key Research and Development Program of China under Grant 2020YFB2103803.
References


[46] Jiancheng Yang, Qiang Zhang, Bingbing Ni, Linguo Li, Jinxian Liu, Mengdie Zhou, and Qi Tian. Modeling point

[47] Li Yi, Vladimir G. Kim, Duygu Ceylan, I-Chao Shen, Mengyan Yan, Hao Su, Cewu Lu, Qixing Huang, Alla Sheffer, and Leonidas Guibas. A scalable active framework for region annotation in 3d shape collections. ACM Transactions on Graphics, 35(6), 2016. 2, 5, 6


