Walk in the Cloud: Learning Curves for Point Clouds Shape Analysis

Tiange Xiang
University of Sydney
txia7609@uni.sydney.edu.au

Chaoyi Zhang
University of Sydney
chaoyi.zhang@sydney.edu.au

Yang Song
University of New South Wales
yang.song1@unsw.edu.au

Jianhui Yu
University of Sydney
jianhui.yu@sydney.edu.au

Weidong Cai
University of Sydney
tom.cai@sydney.edu.au

Abstract

Discrete point cloud objects lack sufficient shape descriptors of 3D geometries. In this paper, we present a novel method for aggregating hypothetical curves in point clouds. Sequences of connected points (curves) are initially grouped by taking guided walks in the point clouds, and then subsequently aggregated back to augment their point-wise features. We provide an effective implementation of the proposed aggregation strategy including a novel curve grouping operator followed by a curve aggregation operator. Our method was benchmarked on several point cloud analysis tasks where we achieved the state-of-the-art classification accuracy of 94.2% on the ModelNet40 classification task, instance IoU of 86.8% on the ShapeNetPart segmentation task and cosine error of 0.11 on the ModelNet40 normal estimation task. Our project page with source code is available at: https://curvenet.github.io/.

1. Introduction

The point cloud is a primary data structure in a string of indoor/outdoor computer vision applications. A large variety of 3D sensors (e.g. LiDAR sensors) are now able to capture real-world objects, and their projections to digital forms can be made by sampling discrete points on the surface. To reach a better understanding of the 3D targets, effective point cloud analysis techniques and methods are in great demand. With the thriving of deep learning, the pioneer works [27, 29] and their followers [21, 6, 44, 1, 43, 7, 18, 48] processed point clouds through well-designed neural networks to learn the latent mappings between input point coordinates and the ground truth labels. Differing from conventional 2D vision tasks, the points are usually in irregular and unordered forms, hence, effective design of feature aggregation and message passing schemes among point clouds still remains challenging.

Local feature aggregation is a basic operation that has been widely studied recently. For each key point, its neighborhood point features are first grouped by pre-defined rules (e.g. KNN). The relative position encodings between the query point and neighboring points will be computed subsequently and then passed into various point-based transformation and aggregation modules for local feature extraction. Although the above operations help depict local patterns to some extent, long-range point relations are neglected. While the non-local module [35] provides a solution to aggregate global features, we argue that the global point-to-point mapping might still be insufficient to extract underlying patterns implied by the point cloud shapes.

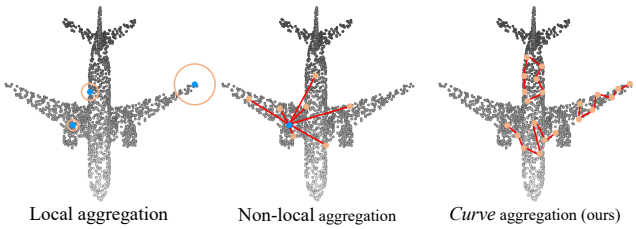

Figure 1. **Common aggregations and the *curve* aggregation**. Blue circles denote key points, orange circles denote query points or query range. In *curve* aggregation, query features will be aggregated into all points.

To this end, we propose to improve the point cloud geometry learning through generating continuous sequences of point segments. We posit that such continuous descriptors are more adequate for depicting the geometry of point cloud objects, compared to popular existing local and non-local operators nowadays. We denote such continuous descriptors as *curves*. By regarding a point cloud as an undirected graph, where the discrete points serve as the graph nodes and the neighbor point connections as the graph edges, a *curve* can therefore be described as a walk in the graph. Figure 1 intuitively compares the local aggregation, non-local aggregation, and our *curve aggregation* operators. In this paper, we first revisit the local feature aggregation in its general form and provide in-depth discussions on why long-range feature aggregation strategies are desired (Sec. 3.1). We then formulate our method formally by defining the curve grouping policy (Sec. 3.2) and the aggregation between curve features and point features (Sec. 3.3). A novel

point cloud processing network, CurveNet, is constructed by integrating the proposed modules along with several basic building blocks into a ResNet [4] style network.

Our main contributions are three-fold: **(1)** We propose a novel feature aggregation paradigm for point cloud shape analysis. Sequences of points (*curves*) are grouped and aggregated for better depiction of point cloud object geometries. A novel curve grouping operator along with a curve aggregation operator are proposed to achieve curve feature propagation. **(2)** We study potential drawbacks of grouping *loops* and provide our solutions to alleviate them. Moreover, a dynamic encoding strategy is proposed so that *curves* can contain richer information while suppressing potential *crossovers*. **(3)** We embed the curve modules into a network named *CurveNet*, which achieves state-of-the-art results on the object classification, normal estimation, and object part segmentation tasks.

2. Related Works

3D point cloud processing. One of the greatest challenges to point cloud analysis is processing unstructured representations. Starting from indirect representation transformation methods [17, 11, 30, 14] that first transform the point cloud into another representation (e.g. octree, kdtree) to ease the analysis difficulty, many recent works [27, 47, 6] extract features from the raw point cloud directly.

As one of the pioneer direct approaches, PointNet/PointNet++ [27, 29] utilize shared MLPs to learn point-wise features. Following them, recent works have extended the point-wise method to various directions, which include designing advanced convolution operations [12, 43, 38, 20], considering a wider neighborhood [15, 50, 36, 22], and the adaptive aggregation of it [6, 44, 49, 45]. The success of the above methods is inseparable from the help of feature aggregation operators, which achieve the direct message passing of discrete points in deep networks.

Current feature aggregation operators can be generally classified into local and non-local feature aggregations. As a representative of local aggregation operator, *EdgeConv* [36] learns semantic displacement between key points and their feature-space neighbors. The thrive of non-local aggregation operator starts from the non-local network [35], with which global features are transformed and aggregated together to learn many-to-one feature mappings. With the recent success of applying Transformer [34] in vision tasks, Guo *et al*. [3] designed a point cloud processing architecture comprised of simple Transformers.

Beyond the local and non-local feature aggregation operators, we suggest that point cloud analysis can be better achieved with special consideration of shape segments, edges, and curves. By aggregating the additional *curve* features, latent feature information can be enriched.

Sampling techniques for 3D point cloud. Sampling technologies aggregate indicative point patterns and are hence essential to all point cloud processing methods. Voxelization-based approaches [28, 37, 39, 24] transform the discrete point space into 3D grid (voxels), where the input point clouds behave as the discrete sampling on a continuous 3D space. However, the quality of such sampling is highly sensitive to the subdivision frequency. Similar to voxelization-based sampling methods, view-based approaches [33, 10, 40] start from capturing 2D snapshots of point clouds from different angles, and make predictions based on the 2D images. The loss of spatial information is inevitable during such image samplings.

Advanced sampling methods from recent literatures overcome the above drawbacks, and obtain promising results on basic point cloud analysis tasks. GS-Net [42] exploits an Eigen-Graph to group points with similar Euclidean distance and geometric information. PointASNL [44] samples both adjacent and global points for a complete description of point cloud objects. Unlike the above methods, PAT [45] models point clouds with the help of a Transformer [34] and learns the point sampling through a Gumbel-Softmax gate. RandLA-Net [6] reviews multiple sampling techniques and adopts the random sampling for efficiently processing large scale point clouds. In a concurrent work, MeshWalker [13] also experiments on taking random walks on mesh surfaces for better mesh analysis. Differing from all existing sampling methods, we take guided walks to group contiguous segments of points as curves, which contain rich information describing object shapes and geometry.

3. Methods

In this section, we present the proposed operators to group and aggregate curves for any point cloud $\mathbf{P} = \{\mathbf{p}\}$ and its point-wise features $\mathbf{F} = \{\mathbf{f}\}$. As aforementioned, a curve represents a connected sequence of points in the point cloud, and can be formally defined as:

Definition 1. ***Curves in point cloud.*** Given $\mathbf{P}$, $\mathbf{F}$ and an isomorphic graph $\mathbf{G} = (\mathbf{F}, \mathbf{E})$ with the connectivity $\mathbf{E}$ computed by the KNN algorithm on $\mathbf{P}$. A curve $\mathbf{c}$ with length l in feature space, is generated as a sequence of point features in $\mathbf{F}$, such that $\mathbf{c} = \{\mathbf{s}_1, \cdots, \mathbf{s}_l | \mathbf{s} \in \mathbf{F}\}$. To group curves, we consider a walk policy π defined on the isomorphic graph $\mathbf{G}$ that starts a walk (curve) from a starting point $\mathbf{s}_1$ and transits for l steps.

3.1. Rethinking Local Feature Aggregation

The general purpose of local feature aggregation is to learn the underlying patterns within a local space of k elements. For each point $\mathbf{p}$, the neighborhood $N = \{\mathbf{p}^1, \cdots, \mathbf{p}^k\}$ is first grouped by a deterministic rule, and KNN is the most frequently used grouping algorithm [6, 44,

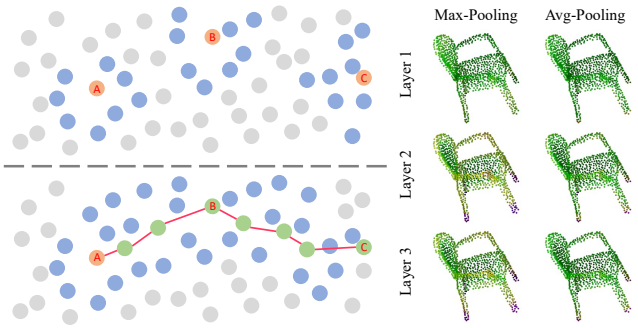

Figure 2. **Left:** A point cloud projected on the 2D plane. **Left top:** Three key points and their k=9 query neighbors. **Left bottom:** A possible *curve* through the three key points connected by red lines. *Curve aggregation* fuses the features along the *curve*. **Right:** Visualizations of the averaged channel $\mathbf{g}$ of a ModelNet40 chair object under different layers of local feature aggregation and different pooling strategies.

21, 47] due to its computational efficiency. Then, pair-wise differences Φ between every two elements in N are computed and stacked together. Finally, shared MLPs are employed to further aggregate the computed encodings leading to the locally aggregated features $\mathbf{g}$. Formally, the above local feature aggregation process can be formulated as:

$$\mathbf{g} = \texttt{pooling}(\{\texttt{MLP}(\Phi(\mathbf{f}, \mathbf{f}^j) | \mathbf{f}^j \in N)\}). \quad (1)$$

Using Manhattan distance $\Phi(\mathbf{f}, \mathbf{f}^j) = \mathbf{f} - \mathbf{f}^j$ as the relative encoding is the most natural practice and has been widely adopted. However, we argue that such encoding method does not provide abundant relative signals since most $\mathbf{g}$ in a point cloud contains nearly identical information in the same feature channel (regardless of pooling strategy), especially in shallow layers, as shown in Figure 2. As the raw point cloud sampled from an explicit representation $\mathcal{R}$ of 3D objects is unordered, the point cloud $\mathbf{P}$ can be regarded as a set of random variables sampled in a particular probability density function (PDF) $\mathbf{U}$ modeled on $\mathcal{R}$, such that $\mathbf{P} \sim \mathbf{U}(\mathcal{R})$. After propagating through a certain number of network layers, $\mathbf{F}$ becomes the transformation over the random variable set $\mathbf{P}$.

We first consider an extreme case where $\mathbf{F}$ represents the initial point features, such that $\mathbf{F} = \mathbf{P}$. Using a simple 2D plane (Figure 2 left) as an exemplary $\mathcal{R}$, the sparse points are scattered on a compact subspace of $\mathbb{R}^2$. In Figure 2 left top, three key points are highlighted with their KNN computed neighbors. It can be practically observed that, after sampling the points, $\{\mathbf{p}_\text{A} - \mathbf{p}_\text{A}^j\}$ and $\{\mathbf{p}_\text{B} - \mathbf{p}_\text{B}^j\}$ are most likely to have similar values, as the key points are surrounded by their query neighbors in a similar pattern. However, point C at the boundary (an edge or an irregular segment of surface in 3D space) stands as an exception. Restricted by the geometry of $\mathcal{R}$, the distribution of point C query neighbors is considerably different than A and B, which leads to varying $\{\mathbf{p}_\text{C} - \mathbf{p}_\text{C}^j\}$.

Based on the above observation, we claim that in any structured PDF that ensures the same sampling behaviour on similar geometries, $\mathbf{g}$ in Eq. 1 is dependent on the distribution of $\mathbf{F}$ and $\mathbf{P}$. Point cloud objects, which have similar geometry information at most parts, turn out encoding similar and indistinguishable information in $\mathbf{g}$. As shown in Figure 2 right, the chair's back and seat have close features, especially in shallow layers. One possible strategy to enrich $\mathbf{g}$ is using more relative encoding rules rather than element-wise difference solely [6, 1]. In this paper, we enrich the local features $\mathbf{g}$ by combining the features aggregated from curves, as illustrated in Figure 2 left bottom. Each curve covers a long path in the point cloud encoding unique geometric information, which could be used to further increase point feature diversity.

3.2. Curve Grouping

In this subsection, we present the details on grouping curves in the feature space of point clouds. The starting point of a curve is essential to the overall grouping quality. To group n curves simultaneously, the starting point set in $\mathbb{R}^{n \times ||\mathbf{f}||}$ needs to be determined beforehand. Borrowing the top-k selection method from [2], we employ a sigmoid gated MLP to learn the selection score for each of the point features in $\mathbf{F}$. The starting points are the points with top n scores. To enable gradient flow, we multiply the scores back to $\mathbf{F}$ via a self-attention manner.

After the construction of the starting point set, a walk $\mathcal{W}$ then starts from one of the starting points $\mathbf{s}_1$ and transits for exactly l steps. The points traveled by $\mathcal{W}$ are grouped to form a curve $\mathbf{c}$. Given an intermediate state of curve $\mathbf{s}_i$ ($\mathbf{s}_i$ numerically equals $\mathbf{f}_i$ when grouping curves in feature space) arrived after walking for i steps, we are interested in finding a walk policy $\pi(\mathbf{s}_i)$ that determines the next state of curve at the $i+1$ step. With a predefined $\pi(\cdot)$, a curve $\mathbf{c} = \{\mathbf{s}_1, \cdots, \mathbf{s}_l\}$ can be finally grouped by executing the following equation iteratively for l times:

$$\mathbf{s}_{i+1} = \pi(\mathbf{s}_i), 1 <= i \in \mathbb{Z}^+ <= l. \quad (2)$$

A good $\pi(\cdot)$ is essential to guarantee an effective curve grouping. Instead of a deterministic policy, we present a learnable $\pi(\cdot)$ that can be optimized along with the backbone network. In more detail, for a state $\mathbf{s}$, we apply MLPs on a *state descriptor* $\mathbf{h}_\mathbf{s} \in \mathbb{R}^{2||\mathbf{s}||}$ to decide the next step. A state descriptor is constructed as the concatenation of point feature $\mathbf{s}_i$ and a *curve descriptor* $\mathbf{r}_i$, which will be introduced later in this subsection. Selection logits α on all neighboring points in $N_\mathbf{s}$ can therefore be learned via the MLPs. We then feed α to a scoring function (e.g. softmax) to distribute each of the neighbors a score-based multiplier

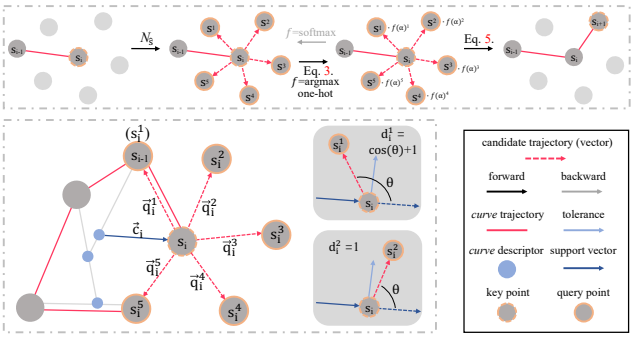

Figure 3. **Top:** An overview of our curve grouping process. **Bottom:** Visualization of the proposed dynamic momentum and the crossover suppression strategies.

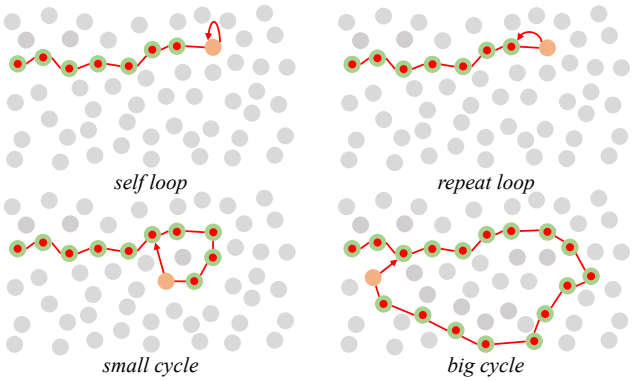

Figure 4. **Four possible *loops* in a curve**. The orange circle denotes current curve head, and the red arrow denotes current curve traveling direction.

within $[0, 1]$. The point with the greatest score is then the output of our $\pi(\cdot)$. Formally, we formulate $\pi(\mathbf{s})$ as:

$$\alpha = \{\texttt{MLP}(\mathbf{h}_{\mathbf{s}^j} | \mathbf{s}^j \in N_{\mathbf{s}})\}, \tag{3}$$

$$\pi(\mathbf{s}) = \mathbf{F}[\arg\max(\texttt{softmax}(\alpha))], \tag{4}$$

where $\mathbf{h}_{\mathbf{s}^j}$ is the state descriptor of a KNN neighbor $\mathbf{s}^j$. In forward propagation, Eq. 4 determines the next state with the computed α. However, during backward propagation, $\arg\max$ obscures gradients, and hence the $\texttt{MLP}$ in Eq. 3 is not able to be updated as expected.

Given the computed α, we present an alternative equation to Eq. 4 that discards the $\arg\max$ gate and enables gradient flow. First, we generate a hard one-hot style score vector for α instead of soft scores obtained from softmax function. By using gumbel-softmax [9, 23, 45] [1] as the scoring function, logits can be converted into an one-hot vector based on the $\arg\max$ index. Gradients through gumbel-softmax are computed identically to the ones through softmax. Then, we broadcastly multiply the query neighbors by the one-hot score vector and sum the multiplications together. The final result of the above operations is numerically identical to the ones computed by Eq. 4. Consequently, our learnable policy is defined as follows:

$$\pi(\mathbf{s}) = \sum_{1}^{k}(\texttt{gumbel-softmax}(\alpha) \cdot N_{\mathbf{s}})), \tag{5}$$

where $\cdot$ denotes broadcast multiplication along the feature dimension. An overview of the above pipeline is shown in Figure 3 bottom.

By extending the curve with the highest score point, $\pi(\cdot)$ essentially determines the traveling direction of the curve based on the state descriptors in the neighborhood. We firstly follow a simple approach that constructs the state descriptor $\mathbf{h}_{\mathbf{s}^j}$ as the direct concatenation of $\mathbf{s}$ and the neighbor $\mathbf{s}^j$. However, such naive approach can easily lead to *loops*, as Eq. 3 will always have the same output for the same input at each point. A *loop* is a $\mathbf{c} = \{\mathbf{s}_1, \cdots, \mathbf{s}_l\}$ with repeated $\mathbf{s}$, which carries redundant and limited information, and hence should be avoided. Figure 4 shows four kinds of possible loops, among which *self loop* can be easily avoided by excluding the key point itself during KNN computation. To avoid the other loops, the simple state descriptor formation will not suffice.

Dynamic momentum. The key to avoid loops lies in a dynamic encoding of state descriptor with consideration of current curve progress. We maintain a *curve descriptor* $\mathbf{r}_i \in \mathbb{R}^{||\mathbf{s}||}$ that encodes the prefix of the curve at step i. The state descriptor $\mathbf{h}_{\mathbf{s}^j}$ for each neighbor of the key point $\mathbf{s}_i$ now becomes the concatenation of $\mathbf{s}_i^j$ and $\mathbf{r}_i$.

Inspired by [8], we update $\mathbf{r}_i$ following the momentum paradigm. However, we find that setting a fixed momentum coefficient β is limited in terms of the final result. We propose that the prefix $\mathbf{r}$ of a curve can be better encoded by a dynamic momentum variant, such that:

$$\begin{aligned} \beta &= \texttt{softmax}(\texttt{MLP}([\mathbf{r}_{i-1}, \mathbf{s}_i])), \\ \mathbf{r}_i &= \beta \mathbf{r}_{i-1} + (1-\beta)\mathbf{s}_i, \end{aligned} \tag{6}$$

where $[\cdot]$ represents concatenation. Figure 3 bottom illustrates the dynamic momentum paradigm.

Crossover suppression. Although the dynamic momentum strategy avoids loops, curves may still encounter *crossovers*. Unlike loops, small number of crossovers may imply useful patterns and should not be avoided completely. However, when a large number of crossovers occurs, same node will be included repeatedly, hurting the curve representation. Therefore, we propose to suppress crossovers by investigating the curve's traveling direction.

We first construct a *support vector* $\overrightarrow{\mathbf{c}}_i = \mathbf{s}_i - \mathbf{r}_{i-1}$ representing the general direction of the current curve at step i. Subsequently, for each query neighbor of the curve head $\mathbf{s}_i$, we compute the *candidate vector* as $\overrightarrow{\mathbf{q}}_i^j = \mathbf{s}_i^j - \mathbf{s}_i$. The

[1] Gumbel samplings are disabled for avoiding any randomness.

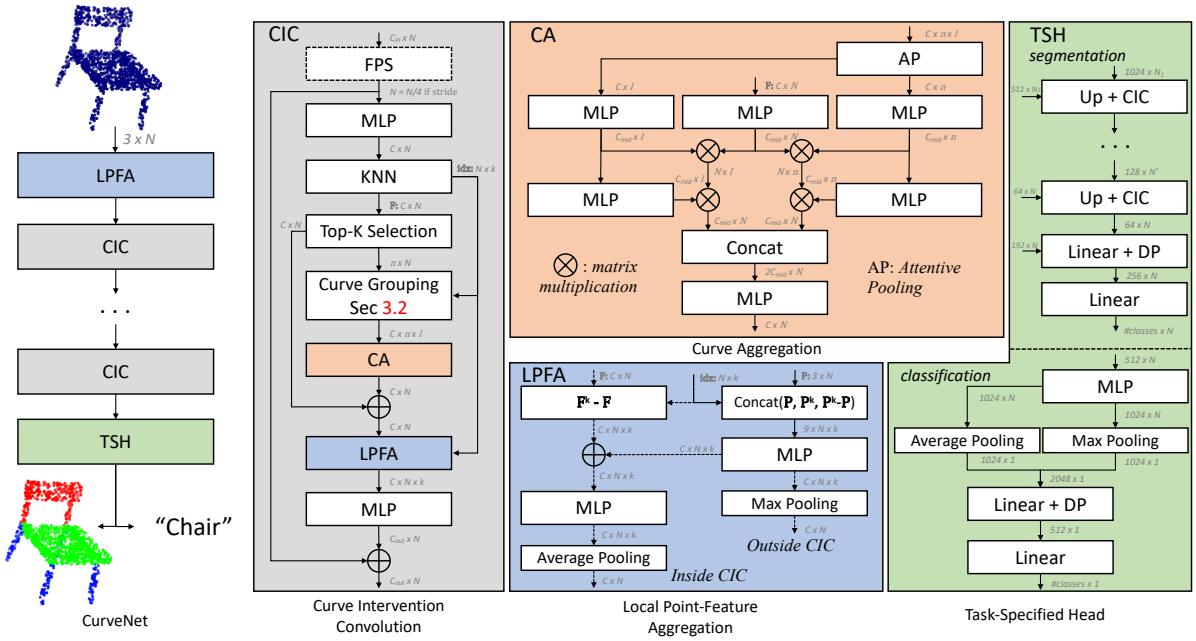

Figure 5. **CurveNet overview**. The network is comprised of a stack of building blocks. FPS denotes the farthest point sampling method [29]. Dotted blocks and lines are optional regarding different blocks. Building blocks are matched in abbreviation and color.

included angle θ between $\overrightarrow{\mathbf{c}}$ and $\overrightarrow{\mathbf{q}}^j$ indicates whether the curve is about to go straight or make a turn. We suppress crossovers by scaling down α^j that has large θ (i.e. likely to turn around and cause potential crossovers).

Specifically, we determine the angle distance between $\overrightarrow{\mathbf{c}}$ and $\overrightarrow{\mathbf{q}}^j$ through cosine similarity and the measurement for each vector pair strictly falls into the range $[-1, 1]$. A value close to -1 represents the two vectors are in the opposite direction (should be suppressed) and 1 represents they are in the same direction (should not be suppressed). Considering there exist boundaries in the latent space, a curve cannot go straight forever without making a turn. We therefore set a tolerance threshold angle $\bar{\theta}$ [2], so that only the candidate vector with included angle greater than $\bar{\theta}$ needs to be suppressed. Following the above intuition, we then construct the *crossover suppression multiplier* $\mathbf{d}^j$ by shifting and clipping the cosine similarity score into [0, 1]. Potential crossovers are suppressed by scaling the candidate logit α^j with $\mathbf{d}^j$. An illustration of the crossover suppression strategy is outlined in Figure 3 bottom.

3.3. Curve Aggregation and CurveNet

As discussed in Sec 3.1, the purpose of curve aggregation is to enrich the intra-channel feature variety of the relative encodings ϕ and eventually provides a better description to $\mathbf{g}$. For notation simplicity, we define the number of feature channels as C, the number of points as P, and a basic Attentive-Pooling operator [6] as AP. In AP, the input features $\in \mathbb{R}^{C\times *}$ are scaled in the self-attention style and summed up along the $*$ dimension, resulting in $\mathbb{R}^{C\times 1}$.

Given the grouped curves $\mathbf{C} = \{\mathbf{c}_1, \cdots, \mathbf{c}_n\} \in \mathbb{R}^{C\times n\times l}$, to aggregate the features in curves, we consider both inter-relations among the curves and intra-relations within each curve. We first learn an inter-curve feature vector $f_{inter} \in \mathbb{R}^{C\times l}$ and an intra-curve feature vector $f_{intra} \in \mathbb{R}^{C\times n}$ by applying AP on $\mathbf{C}$ along different axis. Point features $\mathbf{F}$ together with f_{intra} and f_{inter} are then fed to three individual bottleneck MLPs to reduce feature dimensions. We apply matrix multiplication on the reduced $\mathbf{F}$ with reduced f_{intra} and f_{inter} separately to learn the respective curve-point mappings. Softmax functions are employed to convert the mappings into scores. In another branch, the reduced f_{intra} and f_{inter} are further transformed with two extra MLPs, which are then fused with the computed mapping scores by matrix multiplication separately. The above process ends up with two fine-grained feature vectors f'_{intra} and f'_{inter} in the same shape of $\mathbb{R}^{C\times P}$. We concatenate f'_{intra} and f'_{inter} along the feature axis and feed into a final MLP. The output of our curve aggregation is the residual addition to the origin input.

We embed our Curve Grouping (CG) block and Curve Aggregation (CA) block into a Curve Intervention Convolution (CIC) block. In each CIC block, curves are first grouped (Sec 3.2) and then aggregated to all point features (Sec 3.3). We stack 8 CIC blocks together to construct a ResNet [4] style network, referred to as CurveNet. Our Cur-

[2] Learning such value yields poorer results in our experiments.

veNet initially learns a relative local encoding of the input point coordinates through the Local Point-Feature Aggregation (LPFA) block, which projects the relative point difference into a higher dimension. CurveNet eventually makes predictions through the Task-Specified Head (TSH) regarding of different point cloud processing tasks. For classification task, the extracted point features are first pooled and then passed into two fully-connected layers. For segmentation task, we use an attention U-Net [25] style decoder that concatenates attentive shortcut connections from the encoder. Figure 5 gives an overview of our CurveNet. Network structure and building block details are presented in the supplemental materials.

4. Experiments

We present experimental results for our point cloud object analysis methods on object classification, shape part segmentation, and normal estimation tasks.

4.1. Implementation Details

In all experiments, Dropout layers [32] were adopted in final linear layers with probability 0.5 [36]. We used LeakyReLU as the activation function in the backbone subnetwork and ReLU in the task-specific heads. $\bar{\theta}$ was set to 90°. To eliminate the influence of randomness, random seed was fixed in all experiments, which were implemented in the PyTorch framework [26].

For classification tasks, we used SGD with momentum 0.9 as the optimizer and set the number of neighbors in KNN to 20. For segmentation tasks, the number of KNN neighbors was set dynamically according to different radius with no more than 32. The point features were interpolated similar to [27] during upsampling. The distances between predictions and ground truth labels were minimized through cross entropy loss.

4.2. Benchmarks

Object classification. ModelNet 10/40 datasets [39] are the most commonly used datasets for object shape classification benchmarks, which collect meshed CAD models across a variety of objects. ModelNet10 dataset is comprised of 4899 individual models that are distributed into 10 different categories. We split the training and testing samples following the same schema as in [19]. In a larger homogeneous dataset, ModelNet40 consists of 12311 models that are classified into 40 categories. In both datasets, we only used the coordinates of 1024 uniformly sampled points as network inputs. The points were normalized into unit spheres before feeding into networks. A random scaling multiplier within the range [0.66, 1.5] was first multiplied on the sampled points. Then, each point was translated along the three directions by random displacements within [-0.2, 0.2]. The scaling and translation settings were consistent to the ones used in [11, 20]. We trained the models for 200 epochs, starting with a learning rate of 0.1 and cosineannealling scheduled to 0.001 in 200 epochs. Batch size was set to 32 for training, and 16 for validation.

Table 1. ModelNet10 (M10) and ModelNet40 (M40) classification accuracy (%). 'nr' denotes using normal vectors as extra inputs. '*' denotes methods evaluated with voting strategy [20].

Methods	Input	#point	M10↑	M40↑
PointNet [27]	xyz	1024	-	89.2
PointNet++ [29]	xyz	1024	-	90.7
DGCNN [36]	xyz	1024	-	92.9
PointASNL [44]	xyz	1024	95.7	92.9
RS-CNN [20]	xyz	1024	-	92.9
RS-CNN [20] *	xyz	1024	-	93.6
Grid-CNN [43]	xyz	1024	**97.5**	93.1
PCT [3]	xyz	1024	-	93.2
PAConv [41]	xyz	1024	-	93.6
PAConv [41] *	xyz	1024	-	93.9
A-CNN [12]	xyz, nr	1024	95.5	92.6
PosPool [21] *	xyz	2048	-	93.2
SO-Net [15]	xyz, nr	5000	95.7	93.4
CurveNet (Ours)	xyz	1024	96.1	**93.8**
CurveNet (Ours) *	xyz	1024	96.3	**94.2**

Table 2. ShapeNet part results in mean intersection of union (%).

Methods	Input	#point	mIoU↑
PointNet [27]	xyz	2048	83.7
DGCNN [36]	xyz	2048	85.1
PointCNN [16] *	xyz	2048	86.1
PointASNL [44]	xyz	2048	86.1
RS-CNN [20] *	xyz	2048	86.2
PAConv [41]	xyz	2048	86.0
PAConv [41] *	xyz	2048	86.1
PCT [3] *	xyz	2048	86.4
PointNet++ [29]	xyz, nr	2048	85.1
SO-Net [15]	xyz, nr	1024	84.6
CurveNet w/o curves (Ours)	xyz	2048	85.9
CurveNet (Ours)	xyz	2048	**86.6**
CurveNet (Ours) *	xyz	2048	**86.8**

Table 3. Normal estimation results in avg cosine-distance error.

Methods	Input	#point	Error↓
PointNet [27]	xyz	1024	0.47
DGCNN [36]	xyz	1024	0.29
RS-CNN [20]	xyz	1024	0.15
PCT [3]	xyz	1024	0.13
CurveNet w/o curves (Ours)	xyz	1024	0.16
CurveNet (Ours)	xyz	1024	**0.11**

Table 1 reports the comparison results between our CurveNet and the most recent methods. With only 1024 uni-

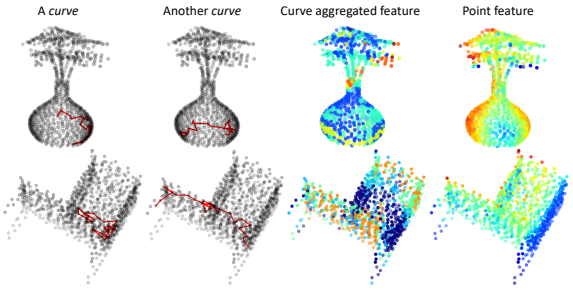

Figure 6. **Visualizations of curves and curve features. Left:** Loops are avoided and crossovers are suppressed in the grouped curves. **Right:** Point features can be enriched by combining the long-range curve features.

formly sampled points, our method achieved the state-of-the-art result on the large scale ModelNet40 dataset at 93.8% without voting [20] and reached 94.2% when averages 10 prediction votes. We also achieved 96.3% on the ModelNet10 subset, which is the second best result among all methods with the same training data. Moreover, CurveNet is a highly memory-efficient architecture that requires only 4.1 G GPU memory for training while DGCNN [36] requires 5.9 G. We visualize the grouped curves, the local aggregated features, and the curve aggregated features on two randomly selected channels in Figure 6. The curves are able to cover long-range semantics and hence bring channel diversity to a great extent.

Object part segmentation. We validated our method on the ShapeNetPart dataset [46] for the 3D shape part segmentation task. The dataset collects 16881 shape models across 16 categories. Most objects in the dataset have been labeled with less than 6 parts, results in a total number of 50 different parts. Our training and testing split scheme follows [27, 29], such that 12137 models were used as training samples while the rest were used as validation. 2048 points were uniformly sampled from each model to be the input to our networks. We trained the models for 200 epochs with batch size 32, starting with a learning rate of 0.05 which decayed by 0.1 at the 140th and 180th epoch. The momentum and weight decay are set to 0.9 and 0.0001, respectively. We inserted a simple SE [5] module before the last linear layer of our CurveNet. Same to [20, 1], the one-hot class label vector and global feature vectors are also adopted.

Mean intersection of union (mIoU) results across instances are reported in Table 2, and the category-wise mIoU scores are presented in the supplemental materials. Our method achieves state-of-the-art overall mIoU of 86.6%, surpasses all existing methods. Without grouping any curves, our base architecture reaches 85.9%, proving the effectiveness of involving curves in point cloud shape segmentation task. Moreover, we visualized four cases qualitatively along with the learned curves in Figure 7. The

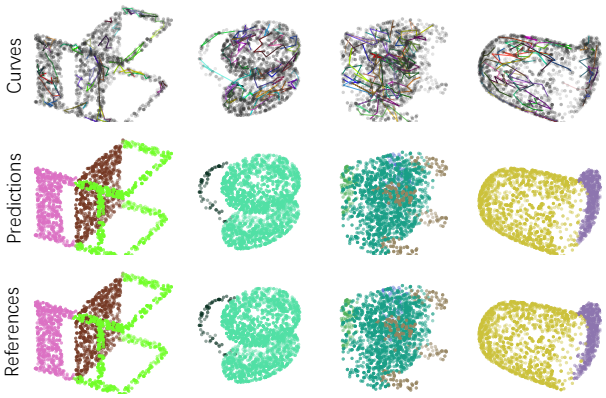

Figure 7. **Visualizations of curves and segmentation results.** Random selected curves are plotted with random colors.

grouped curves were able to explore both short and long range shape relations. Model complexity is reported and analyzed in the supplementary materials.

Object normal estimation. Object surface normal is essential to 3D modeling and rendering. Differing from understanding objects part by part, estimating normal requires a comprehensive understanding of the entire object shape and geometry. We validate our CurveNet on estimating normal by using ModelNet40 dataset, where each point in the point cloud has been labeled with its three-directional normal. CurveNet architecture is constructed similarly to the one used in segmentation task, excluding the one-hot class label vector and global feature vectors. Models are trained with an initial learning rate of 0.05 and cosineanealing scheduled to 0.0005 in 200 epochs.

Table 3 shows the average cosine-distance error comparisons of CurveNet and state-of-the-art methods. Without any curves, our base CurveNet architecture achieves an average error of 0.16, which is closed to [20, 3]. When curves are involved, our full CurveNet demonstrates a superior performance with 0.11 average error, setting a new benchmark to the normal estimation task.

4.3. Ablation Studies

We conducted extensive experiments on ModelNet40 dataset to study our proposed method comprehensively. Unless explicitly specified, implementation details remained the same as the ones described in the benchmark section. All the ablative studies were examined without voting.

Component studies. The impact of individual component of CurveNet was examined by simply removing or replacing them from the full CurveNet architecture. We conducted experiments on replacing LPFA with the common local feature aggregation as in Eq. 1, disabling the dynamic momentum and the crossover suppression strategies, and replacing the proposed CA operator to the vanilla non-local

Table 4. Component study results. Top-1 classification accuracy and per point cloud (per batch) inference time are reported. LPFA: the Local Point-Feature Aggregation, CG: the Curve Grouping operator, DM: the Dynamic Momentum strategy, CS: the Crossover Suppression strategy, and CA: the Curve Aggregation operator.

Model	LPFA	CG			Acc (%)	latency (ms)
		DM	CS	CA		
A					93.1	37.6(140)
B	✓				93.3	37.5(143)
C	✓	✓		✓	93.4	44.3(145)
D	✓		✓	✓	93.3	44.1(144)
E	✓	✓	✓		93.1	45.0(146)
F		✓	✓	✓	93.4	44.8(143)
G	✓	✓	✓	✓	93.8	45.2(146)

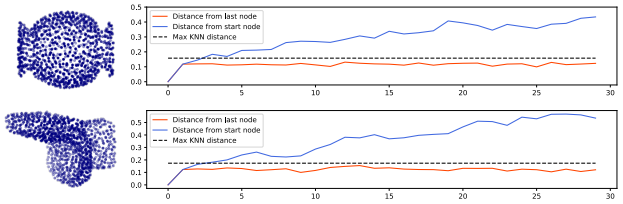

Figure 8. **How far do the curves go?** Left are two objects with distinct shapes. Right are the average euclidean distances to the start node and the last node of all curves with 30 steps in group 1.

module (i.e. intra-relations and inter-relations are not separated). The results are reported in Table 4.

We observed that, although using LPFA solely cannot bring significant performance improvement (models A and B), with the intervention of curves, the presence of LPFA is able to make a huge difference in terms of the classification result (models F and G). As shown in models C and D, both the proposed dynamic momentum and crossover suppression strategy are empirically effective as expected. Moreover, from model E, we find that the proposed curve aggregation operator plays the most significant role in the CurveNet. When aggregating grouped curve features through the vanilla non-local module, the accuracy is dropped by 0.7%, with no benefit on the inference latency.

Shallow layer vs deep layer. In Sec. 3.1, we claimed that shallow features after local aggregation lack single channel diversity, and curve features could be more desired at shallow layers than at deep layers of a network. We conducted experiments on aggregating curves with various quantities and lengths on different groups of our CurveNet, the results are shown in the supplemental materials (Figure 1). Aggregating curves at shallow layers (group 1/2) yield better results than at deep layers (group 3/4), which proved our claim empirically.

Curve quantity vs curve length. Curve quantity n and length l are the two hyper parameters determining the network performance directly. Short curves cannot capture long-range patterns, while long curves require better guidance and may contain redundant information. To study the relations between curve quantity and curve length, we conducted experiments on fixed total point number in curves. Figure 1 bottom right in the supplemental materials shows that although a long curve (length 50) is able to achieve the best result, the network performance degrades as the curve extends further. Aggregating longer curves is also computational inefficient, as the transition of nodes cannot be computed parallelly. To verify whether the curves are trapped in a local region, we present the average Euclidean distances between each node of the curves to the starting/last point in Figure 8. The curves are capable of jumping out of the maximum local KNN range to explore longer range relations.

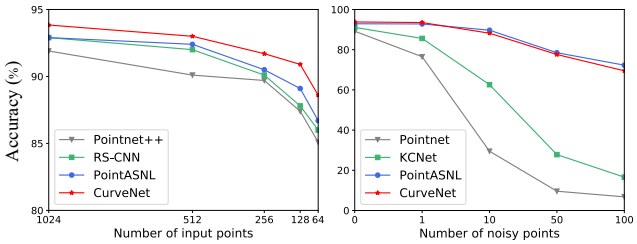

Figure 9. **Left:** Comparison on sparser training and testing inputs. **Right:** Comparison on noisy inputs, with voting enabled.

Sparser input points and noisy testing points. Curve grouping could be sensitive to point cloud sparsity and noise. We conduct extensive experiments on **(1)** training and testing with sparser input points and **(2)** training on 1024-point raw coordinates and testing with noisy points [44]. As shown in Figure 9 left, our CurveNet achieves the best results on all experiments regarding of different number of input points. For noise tests, we add an extra max pooling layer following the first LPFA block. Our CurveNet outperforms [31, 27] in all experiments and achieves on par results to [44], demonstrating the robustness to noise.

5. Conclusion

In this paper, we proposed a long-range feature aggregation method, namely *curve aggregation*, for point clouds shape analysis. We first discussed the potential drawbacks of the existing local feature aggregation paradigm, and claimed the need for the aggregation of point cloud geometry. We then presented our method in two sequential steps: the rules for grouping curves in a point cloud and the integration of the grouped curve features with the extracted point features. During the process, potential problems were defined and resolved. Our method achieved state-of-the-art results on multiple point clouds object analysis tasks.

References

[1] Qendrim Bytyqi, Nicola Wolpert, and Elmar Schömer. Local-area-learning network: Meaningful local areas for efficient point cloud analysis. *arXiv preprint arXiv:2006.07226*, 2020.

[2] Hongyang Gao and Shuiwang Ji. Graph U-Nets. *arXiv preprint arXiv:1905.05178*, 2019.

[3] Meng-Hao Guo, Jun-Xiong Cai, Zheng-Ning Liu, Tai-Jiang Mu, Ralph R Martin, and Shi-Min Hu. Pct: Point cloud transformer. *arXiv preprint arXiv:2012.09688*, 2020.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[5] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

[6] Qingyong Hu, Bo Yang, Linhai Xie, Stefano Rosa, Yulan Guo, Zhihua Wang, Niki Trigoni, and Andrew Markham. Randla-net: Efficient semantic segmentation of large-scale point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11108–11117, 2020.

[7] Zeyu Hu, Mingmin Zhen, Xuyang Bai, Hongbo Fu, and Chiew-lan Tai. JSENet: Joint semantic segmentation and edge detection network for 3D point clouds. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 222–239, 2020.

[8] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[9] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[10] Asako Kanezaki, Y Matsushita, and Y Nishida. RotationNet: Learning object classification using unsupervised viewpoint estimation. *CoRR abs/1603.06208*, 3, 2016.

[11] Roman Klokov and Victor Lempitsky. Escape from cells: Deep kd-networks for the recognition of 3D point cloud models. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 863–872, 2017.

[12] Artem Komarichev, Zichun Zhong, and Jing Hua. A-CNN: Annularly convolutional neural networks on point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7421–7430, 2019.

[13] Alon Lahav and Ayellet Tal. Meshwalker: Deep mesh understanding by random walks. *ACM Transactions on Graphics (TOG)*, 39(6):1–13, 2020.

[14] Huan Lei, Naveed Akhtar, and Ajmal Mian. Octree guided cnn with spherical kernels for 3D point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9631–9640, 2019.

[15] Jiaxin Li, Ben M Chen, and Gim Hee Lee. So-Net: Self-organizing network for point cloud analysis. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9397–9406, 2018.

[16] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. PointCNN: Convolution on X-transformed points. In *Advances in neural information processing systems*, pages 820–830, 2018.

[17] Yangyan Li, Soeren Pirk, Hao Su, Charles R Qi, and Leonidas J Guibas. FPNN: Field probing neural networks for 3D data. In *Advances in Neural Information Processing Systems*, pages 307–315, 2016.

[18] Cheng Lin, Changjian Li, Yuan Liu, Nenglun Chen, Yi-King Choi, and Wenping Wang. Point2skeleton: Learning skeletal representations from point clouds. *arXiv preprint arXiv:2012.00230*, 2020.

[19] Xinhai Liu, Zhizhong Han, Yu-Shen Liu, and Matthias Zwicker. Point2Sequence: Learning the shape representation of 3D point clouds with an attention-based sequence to sequence network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 8778–8785, 2019.

[20] Yongcheng Liu, Bin Fan, Shiming Xiang, and Chunhong Pan. Relation-shape convolutional neural network for point cloud analysis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8895–8904, 2019.

[21] Ze Liu, Han Hu, Yue Cao, Zheng Zhang, and Xin Tong. A closer look at local aggregation operators in point cloud analysis. *arXiv preprint arXiv:2007.01294*, 2020.

[22] Zhe Liu, Shunbo Zhou, Chuanzhe Suo, Peng Yin, Wen Chen, Hesheng Wang, Haoang Li, and Yun-Hui Liu. LPD-Net: 3D point cloud learning for large-scale place recognition and environment analysis. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2831–2840, 2019.

[23] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

[24] Daniel Maturana and Sebastian Scherer. VoxNet: A 3D convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 922–928. IEEE, 2015.

[25] Ozan Oktay, Jo Schlemper, Loic Le Folgoc, Matthew Lee, Mattias Heinrich, Kazunari Misawa, Kensaku Mori, Steven McDonagh, Nils Y Hammerla, Bernhard Kainz, et al. Attention U-Net: Learning where to look for the pancreas. *arXiv preprint arXiv:1804.03999*, 2018.

[26] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 8024–8035, 2019.

[27] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. PointNet: Deep learning on point sets for 3D classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[28] Charles R Qi, Hao Su, Matthias Nießner, Angela Dai, Mengyuan Yan, and Leonidas J Guibas. Volumetric and multi-view CNNs for object classification on 3D data. In

Proceedings of the IEEE conference on computer vision and pattern recognition, pages 5648–5656, 2016.

[29] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. PointNet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in neural information processing systems*, pages 5099–5108, 2017.

[30] Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. OctNet: Learning deep 3D representations at high resolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3577–3586, 2017.

[31] Yiru Shen, Chen Feng, Yaoqing Yang, and Dong Tian. Mining point cloud local structures by kernel correlation and graph pooling. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4548–4557, 2018.

[32] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[33] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. Multi-view convolutional neural networks for 3D shape recognition. In *Proceedings of the IEEE international conference on computer vision*, pages 945–953, 2015.

[34] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[35] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7794–7803, 2018.

[36] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph CNN for learning on point clouds. *ACM Transactions On Graphics (TOG)*, 38(5):1–12, 2019.

[37] Jiajun Wu, Chengkai Zhang, Tianfan Xue, Bill Freeman, and Josh Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Advances in neural information processing systems*, pages 82–90, 2016.

[38] Wenxuan Wu, Zhongang Qi, and Li Fuxin. Pointconv: Deep convolutional networks on 3D point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9621–9630, 2019.

[39] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015.

[40] Jin Xie, Guoxian Dai, Fan Zhu, Edward K Wong, and Yi Fang. DeepShape: Deep-learned shape descriptor for 3d shape retrieval. *IEEE transactions on pattern analysis and machine intelligence*, 39(7):1335–1345, 2016.

[41] Mutian Xu, Runyu Ding, Hengshuang Zhao, and Xiaojuan Qi. Paconv: Position adaptive convolution with dynamic kernel assembling on point clouds. *arXiv preprint arXiv:2103.14635*, 2021.

[42] Mingye Xu, Zhipeng Zhou, and Yu Qiao. Geometry sharing network for 3d point cloud classification and segmentation. In *AAAI*, pages 12500–12507, 2020.

[43] Qiangeng Xu, Xudong Sun, Cho-Ying Wu, Panqu Wang, and Ulrich Neumann. Grid-GCN for fast and scalable point cloud learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5661–5670, 2020.

[44] Xu Yan, Chaoda Zheng, Zhen Li, Sheng Wang, and Shuguang Cui. PointASNL: Robust point clouds processing using nonlocal neural networks with adaptive sampling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5589–5598, 2020.

[45] Jiancheng Yang, Qiang Zhang, Bingbing Ni, Linguo Li, Jinxian Liu, Mengdie Zhou, and Qi Tian. Modeling point clouds with self-attention and gumbel subset sampling. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3323–3332, 2019.

[46] 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 (ToG)*, 35(6):1–12, 2016.

[47] Chaoyi Zhang, Yang Song, Lina Yao, and Weidong Cai. Shape-oriented convolution neural network for point cloud analysis. In *AAAI*, pages 12773–12780, 2020.

[48] Chaoyi Zhang, Jianhui Yu, Yang Song, and Weidong Cai. Exploiting edge-oriented reasoning for 3d point-based scene graph analysis. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[49] Wenxiao Zhang and Chunxia Xiao. PCAN: 3D attention map learning using contextual information for point cloud based retrieval. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 12436–12445, 2019.

[50] Hengshuang Zhao, Li Jiang, Chi-Wing Fu, and Jiaya Jia. Pointweb: Enhancing local neighborhood features for point cloud processing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5565–5573, 2019.