A Closer Look at Local Aggregation Operators in Point Cloud Analysis

Ze Liu[1,2⋆†], Han Hu[2⋆], Yue Cao[2], Zheng Zhang[2], and Xin Tong[2]

[1] University of Science and Technology of China
liuze@mail.ustc.edu.cn
[2] Microsoft Research Asia
{hanhu,yuecao,zhez,xtong}@microsoft.com

Abstract. Recent advances of network architecture for point cloud processing are mainly driven by new designs of local aggregation operators. However, the impact of these operators to network performance is not carefully investigated due to different overall network architecture and implementation details in each solution. Meanwhile, most of operators are only applied in shallow architectures. In this paper, we revisit the representative local aggregation operators and study their performance using the same deep residual architecture. Our investigation reveals that despite the different designs of these operators, all of these operators make surprisingly similar contributions to the network performance under the same network input and feature numbers and result in the state-of-the-art accuracy on standard benchmarks. This finding stimulate us to rethink the necessity of sophisticated design of local aggregation operator for point cloud processing. To this end, we propose a simple local aggregation operator without learnable weights, named Position Pooling (PosPool), which performs similarly or slightly better than existing sophisticated operators. In particular, a simple deep residual network with PosPool layers achieves outstanding performance on all benchmarks, which outperforms the previous state-of-the methods on the challenging PartNet datasets by a large margin (7.4 mIoU). The code is publicly available at https://github.com/zeliu98/CloserLook3D.

Keywords: 3D point cloud, local aggregation operator, position pooling

1 Introduction

With the rise of 3D scanning devices and technologies, 3D point cloud becomes a popular input for many machine vision tasks, such as autonomous driving, robot navigation, shape matching and recognition, etc. Different from images and videos that are defined on regular grids, the point cloud locates at a set of irregular positions in 3D space, which makes the powerful convolutional neural networks (CNN) and other deep neural networks designed for regular data hard to be applied. Early studies transform the irregular point set into a regular

⋆ Equal contribution. †This work is done when Ze Liu is an intern at MSRA.

grid by either voxelization or multi-view 2D projections such that the regular CNN can be adopted. However, the conversion process always results in extra computational and memory costs and the risk of information loss.

Recent methods in point cloud processing develop networks that can directly model the unordered and non-grid 3D point data. These architectures designed for point cloud are composed by two kinds of layers: the point-wise transformation layers and local aggregation layers. While the point-wise transformation layer is applied on features at each point, the local aggregation layer plays a similar role for points as the convolution layer does for image pixels. Specifically, it takes features and relative positions of neighborhood points to a center point as input, and outputs the transformed feature for the center point. To achieve better performance in different point cloud processing tasks, a key task of point cloud network design is to develop effective local aggregation operators.

Existing local aggregation operators can be roughly categorized into three groups according to the way that they combine the relative positions and point features: point-wise multi-layer perceptions (MLP) based [22, 35, 13, 11], pseudo grid feature based [9, 17, 38, 12, 27, 30] and adaptive weight based [34, 5, 16, 36, 32, 14]. The point-wise MLP based methods treat a point feature and its corresponding relative position equally by concatenation. All the concatenated features at neighborhood are then abstracted by a small PointNet [20] (multiple point-wise transformation layers followed by a MAX pooling layer) to produce the output feature for the center point. The pseudo grid feature based methods first generate pseudo features on pre-defined grid locations, and then learn the parametrized weights on these grid locations like regular convolution layer does. The adaptive weight based methods aggregate neighbor features by weighted average with the weights adaptively determined by relative position of each neighbor.

Despite the large efforts for aggregation layer design and performance improvements of the resulting network in various point cloud processing tasks, the contributions of the aggreation operator to the network performance have never been carefully investigated and fairly compared. This is mainly due to the different network architectures used in each work, such as the network depth, width, basic building blocks, whether to use skip connection, as well as different implementation of each approach, such as point sampling method, neighborhood computation, and so on. Meanwhile, most of existing aggregation layers are applied in shallow networks, it is unclear whether these designs are still effective as the network depth increases.

In this paper, we present common experimental settings for studying these operators, selecting a deep residual architecture as the base networks, as well as same implementation details regarding point sampling, local neighborhood selection and etc. We also adopt three widely used datasets, ModelNet40 [37], S3DIS [1] and PartNet [19] for evaluation, which account for different tasks, scenarios and data scales. Using these common experimental settings, we revisit the performance of each representative operator and make fair comparison between them. We find appropriate settings for some operators under this deep residual architecture are different from that of using shallower and non-residual

networks. We also surprisingly find that different representative methods perform similarly well under the same representation capacity on these datasets, if appropriate settings are adopted for each method, although these methods may be invented by different motivations and formulations, in different years.

These findings also encourage us to rethink the role of local aggregation layers in point cloud modeling: *do we really need sophisticated/heavy local aggregation computation?* We answer this question by proposing an extremely simple local aggregation operator with no learnable weights: combining a neighbor point feature and its 3-d relative coordinates by element-wise multiplication, followed with an AVG pool layer to abstract information from neighborhood. We name this new operator as position pooling (PosPool), which shows no less or even better accuracy than other highly tuned sophisticated operators on all the three datasets. These results indicate that we may not need sophisticated/heavy operators for local aggregation computation. We also harness a strong baseline for point cloud analysis by a simple deep residual architecture and the proposed position pooling layers, which achieves 53.8 part category mIoU accuracy on the challenging PartNet datasets, significantly outperforming the previous best method by 7.4 mIoU.

The contributions of this paper are summarized as

- **A common testbed** to fairly evaluate different local aggregation operators.
- **New findings of aggregation operators.** Specifically, *different operators perform similarly well and all of them can achieve the state-of-the-art accuracy*, if appropriate settings are adopted for each operator. Also, *appropriate settings in deep residual networks are different from those in shallower networks*. We hope these findings could shed new light on network design.
- **A new local aggregation operator (PosPool) with no learnable weights** that performs as effective as existing operators. Combined with a deep residual network, this simple operator achieve state-of-the-art performance on 3 representative benchmarks and outperforms the previous best method by a large margin of 7.4 mIoU on the challenging PartNet datasets.

2 Related Works

Projection based Methods project the irregular point cloud onto a regular sampling grid and then apply 2D or 3D CNN over regularly-sampled data for various vision tasks. View-based methods project a 3D point cloud to a set of 2D views from various angles. Then these view images could be processed by 2D CNNs [3, 6, 21, 25]. Voxel-based methods project the 3D points to regular 3D grid, and then standard 3D CNN could be applied [4, 18, 37]. Recently, adaptive voxel-based representations such as K-d trees [10] or octrees [23, 26, 33] have been proposed for reducing the memory and computational cost of 3D CNN. The view-based and voxel-based representations are also combined [21] for point cloud analysis. All these methods require preprocessing to convert the input point cloud and may lose the geometry information.

Global Aggregation Methods process the 3D point cloud via point-wise 1×1 transformation (fully connected) layers followed by a global pooling layer to aggregate information globally from all points [20]. These methods are the first to directly process the irregular point data. They have no restriction on point number, order and regularity of neighborhoods, and obtain fairly well accuracy on several point cloud analysis tasks. However, the lack of local relationship modeling components hinders the better performance on these tasks.

Local Aggregation Methods Recent point cloud architectures are usually composed by 1×1 point-wise transformation layers and local aggregation operators. Different methods are mainly differentiated by their local aggregation layers, which usually adopt the neighboring point features and their relative coordinates as input, and output a transformed center point feature. According to the way they combine point features and relative coordinates, these methods can be roughly categorized into three groups: point-wise MLP based [22, 13, 11], pseudo grid feature based [9, 17, 38, 12, 27, 30], and adaptive weight based [34, 5, 16, 36, 14], as will be detailed in Section 3. There are also some works use additional edge features (relative relationship between point features) as input [13, 32, 35], also commonly referred to as graph based methods.

While we have witnessed significant accuracy improvements on benchmarks by new local aggregation operators year-by-year, the actual progress is a bit vague to the community as the comparisons are made on different grounds that the other architecture components and implementations may vary significantly. The effectiveness of designing components in some operators using deeper residual architectures is also unknown.

3 Overview of Local Aggregation Operators

In this section, we present a general formulation for local aggregation operators as well as a categorization of them.

General Formulation In general, for each point i, a local aggregation layer first transforms a neighbor point j's feature $\mathbf{f}_j \in \mathbb{R}^{d \times 1}$ and its relative location $\Delta\mathbf{p}_{ij} = \mathbf{p}_j - \mathbf{p}_i \in \mathbb{R}^{3 \times 1}$ into a new feature by a function $G(\cdot, \cdot)$, and then aggregate all transformed neighborhood features to form point i's output feature by a reduction function R (typically using MAX, AVG or SUM), as

$$\mathbf{g}_i = R\left(\{G(\Delta\mathbf{p}_{ij}, \mathbf{f}_j) | j \in \mathcal{N}(i)\}\right), \tag{1}$$

where $\mathcal{N}(i)$ represents the neighborhood of point i. Alternatively, edge features $\{\mathbf{f}_i, \Delta\mathbf{f}_{ij}\}$ ($\Delta\mathbf{f}_{ij} = \mathbf{f}_j - \mathbf{f}_i$) can be used as input instead of $\Delta\mathbf{p}_{ij}$ [35].

According to the family to which the transformation function $G(\cdot, \cdot)$ belongs, existing local aggregation operators can be roughly categorized into three types: point-wise MLP based, pseudo grid feature based, and adaptive weight based.

Point-wise MLP based Methods The pioneer work of point-wise MLP based method, PointNet++ [22], applies several point-wise transformation (fully connected) layers on a concatenation of relative position and point feature to achieve

transformation:

$$G(\Delta\mathbf{p}_{ij}, \mathbf{f}_j) = \text{MLP}\left(\text{concat}(\Delta\mathbf{p}_{ij}, \mathbf{f}_j)\right). \tag{2}$$

There are also variants by using an alternative edge feature $\{\mathbf{f}_i, \Delta\mathbf{f}_{ij}\}$ as input [35, 13], or by using a special neighborhood strategy [11]. The reduction function $R(\cdot)$ is usually set as MAX [22, 35, 13].

The multiple point-wise layers after concatenation operation can approximate any continuous function about the relative coordinates and point feature [20, 22]. However, a drawback lies in its large computation complexity, considering the fact that the multiple fully connected (FC) layers are applied to all neighboring points when computing each point's output. Specifically, the FLOPs is $\mathcal{O}(\text{time}) = ((2d+3)+(h-2)d/2)\cdot d/2\cdot nK$, for a point cloud with n points, neighborhood size of K, FC layer number of h, and inter-mediate dimension of $d/2$, when $h \geq 2$. The space complexity is $\mathcal{O}(\text{space}) = ((2d+3)+(h-2)d/2)\cdot d/2$. For $h = 1$, there exists efficient implementation by computation sharing (see Section 4.2).

Pseudo Grid Feature based Methods The pseudo grid feature based methods generate pseudo features on several sampled regular grid points, such that regular convolution methods can be applied. A representative method is KPConv [30], where equally distributed spherical grid points are sampled and the pseudo features on the k^{th} grid point is computed as

$$\mathbf{f}_{i,k} = \sum_{j\in\mathcal{N}(i)} \max(0, 1 - \frac{\|\Delta\mathbf{p}_{jk}\|_2}{\sigma}) \cdot \mathbf{f}_j. \tag{3}$$

The index of each grid point k will have strict mapping with the relative position to center point $\Delta\mathbf{p}_{ik}$. Hence, a (depth-wise) convolution operator with parametrized weights $\mathbf{w}_k \in \mathbb{R}^{d\times 1}$ defined on each grid point can be used to achieve feature transformation:

$$G(\Delta\mathbf{p}_{ik}, \mathbf{f}_{i,k}) = \mathbf{w}_k \odot \mathbf{f}_{i,k}. \tag{4}$$

Different pseudo grid feature based methods mainly differ each other by the definition of grid points [9, 17, 38, 12, 27] or index order [14]. When depth-wise convolution is used, the space and time complexity are $\mathcal{O}(\text{space}) = dM$ and $\mathcal{O}(\text{time}) = ndKM$, respectively, where M is the number of grid points.

Adaptive Weight based Methods The adaptive weight based methods define convolution filters over arbitrary relative positions, and hence can compute aggregation weights on all neighbor points:

$$G(\Delta\mathbf{p}_{ij}, \mathbf{f}_j) = H\left(\Delta\mathbf{p}_{ij}\right) \odot \mathbf{f}_j, \tag{5}$$

where H is typically implemented by several point-wise transformation (fully connected) layers [34, 5]; $\odot$ is an element-wise multiplication operator; R is typically set as SUM.

Some methods adopt more position related variables [16], point density [36], or edge features [32] as the input to compute adaptive weights. More sophisticated function other than fully connected (FC) layers are also used, for example,

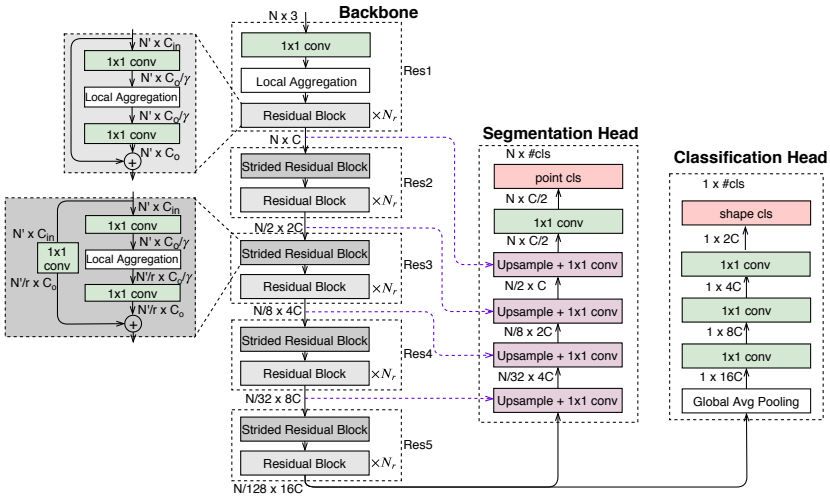


Fig. 1. A common deep residual architecture used to evaluate different local aggregation operators. In evaluation, we adjust the model complexity by changing architecture depth (or block repeating factor N_r), base width C and bottleneck ratio γ. Note the point numbers drawn in this figure is an approximation to indicate the rough complexity but not an accurate number. Actually, the points on each stage are generated by a subsampling method [29] using a fixed grid size and the point number on different point cloud instances can vary

Taylor approximation [14] and an additional SoftMax function to normalize aggregation weights over neighborhood [32].

The space and time complexity of this method are $\mathcal{O}(\text{space}) = ((h-2)d/2 + d + 3) \cdot d/2$ and $\mathcal{O}(\text{time}) = ((h-2)d/2 + d + 5) \cdot d/2 \cdot nK$, respectively, when an inter-mediate dimesion of $d/2$ is used and the number of FC layers $h \geq 2$. When $h = 1$, the space and computation complexity is much smaller, as $\mathcal{O}(\text{space}) = 3d$ and $\mathcal{O}(\text{time}) = 5dnK$, respectively.

Please see Appendix A6 for detailed analysis of the space and time complexity for the above 3 operators.

4 Benchmarking Local Aggregation Operators in Common Deep Architecture

While most local aggregation operators described in Section 3 are reported using specific shallow architectures, it is unknown whether their designing components perform also sweet using a deep residual architecture. In addition, these operators usually use different backbone architectures and different implementation details, making a fair comparison between them difficult.

In this section, we first present a deep residual architecture, as well as implementation details regarding point sampling and neighborhood selection. Then we evaluate the designing components of representative operators using common architectures, implementation details and benchmarks. The appropriate settings

within each method type using the common deep residual architectures are suggested and discussed.

4.1 Common Experimental Settings

Architecture To investigate different local aggregation operators on a same, deep and modern ground, we select a 5-stage deep residual network, similar to the standard ResNet model [7] in image analysis. Residual architectures have been widely adopted in different fields to facilitate the training of deep networks [7, 31]. However, in the point cloud field, until recently, there are some works [30, 13] starting to use deep residual architectures, probably because the unnecessary use of deep networks on several small scale benchmarks. Nevertheless, our investigation shows that on larger scale and more challenging datasets such as PartNet [19], deep residual architectures can bring significantly better performance, for example, with either local aggregation operator type described in Section 3, the deep residual architectures can surpass previous best methods by more than 3 mIoU. On smaller scale datasets such as ModelNet40, they also seldom hurt the performance. The deep residual architecture would be a reasonable choice for practitioners working on point cloud analysis.

Fig. 1 shows the residual architecture used in this paper. It consists 5 stages of different point resolution, with each stage stacked by several bottleneck residual blocks. Each bottleneck residual block is composed successively by a 1×1 point-wise transformation layer, a local aggregation layer, and another 1×1 point-wise transformation layer. At the block connecting two stages, a stridded local aggregation layer is applied where the local neighborhood is selected at a higher resolution and the output adopts a lower resolution. Batch normalization and ReLU layers are applied after each 1×1 layer to facilitate training. For head networks, we use a 4-layer classifier and a U-Net style encoder-decoder [24] for classification and semantic segmentation, respectively.

In evaluation of a local aggregation operator, we use this operator to instantiate all local aggregation layers in the architecture. We also consider different model capacity by varying network depth (block repeating factor N_r), width (C) and bottleneck ratio (γ).

Point Sampling and Neighborhoods. To generate point sets for different resolution levels, we follow [29, 30] to use a subsampling method with different grid sizes to generate point sets in different resolution stages. Specifically, the whole 3D space is divided by grids and one point is randomly sampled to represent a grid if multiple points appear in the grid. This method can alleviate the varying density problem [29, 30]. Given a base grid size at the highest resolution of Res1, the grid size for different resolutions are multiplied by $2\times$ stage-by-stage. The base grid size for different datasets are detailed in Section 6.

To generate a point neighborhood, we follow the ball radius method [22, 8, 16], which in general result in more balanced density than the location or feature kNN methods [34, 2, 35]. The ball radius is set as $2.5\times$ of the base grid size.

Table 1. The performance of baseline operators, sweet spots of point-wise MLP based, pseudo grid feature based and adaptive weight based operators, and the proposed PosPool operators on three benchmark datasets. Baseline* denotes Eq. (6) and baseline† (AVG/MAX) denotes Eq. (7) AVG/MAX, respectively. PosPool and PosPool* denote the operators in Eq. (8) and (10), respectively. (S) after each method denotes a smaller configuration of this method ($N_r = 1$, $\gamma = 2$ and $C = 36$), which is about 16× more efficient than the regular configuration (the other row) of $N_r = 1$, $\gamma = 2$ and $C = 144$. Previous best performing methods on three benchmarks in literature are shown in the first block of this table

method	ModelNet40			S3DIS			PartNet			
	acc	param	FLOP	mIoU	param	FLOP	val	test	param	FLOP
DensePoint [15]	93.2	0.7M	0.7G	-	-	-	-	-	-	-
KPConv [30]	92.9	15.2M	1.7G	65.7	15.0M	6.5G	-	-	-	-
PointCNN [14]	92.5	0.6M	25.3G	65.4	4.4M	36.7G	-	46.4	4.4M	23.1G
baseline*	91.4	19.4M	1.8G	51.5	18.4M	7.2G	42.5	44.6	18.5M	6.7G
baseline† (AVG, S)	90.7	1.2M	0.1G	50.3	1.1M	0.5G	39.5	40.6	1.1M	0.4G
baseline† (AVG)	91.4	19.4M	1.8G	51.0	18.4M	7.2G	44.2	45.8	18.5M	6.7G
baseline† (MAX, S)	91.5	1.2M	0.1G	57.4	1.1M	0.5G	39.8	41.2	1.1M	0.4G
baseline† (MAX)	91.8	19.4M	1.8G	58.4	18.4M	7.2G	45.4	47.4	18.5M	6.7G
point-wise MLP (S)	92.6	1.7M	0.2G	56.7	1.6M	0.8G	45.3	47.0	1.6M	0.7G
point-wise MLP	92.8	26.5M	2.7G	66.2	25.5M	9.8G	48.1	51.5	25.6M	9.1G
pseudo grid (S)	92.3	1.2M	0.3G	64.3	1.2M	1.0G	44.2	45.2	1.2M	0.9G
pseudo grid	93.0	19.5M	2.0G	65.9	18.5M	9.3G	50.8	53.0	18.5M	8.5G
adapt weights (S)	92.1	1.2M	0.2G	61.9	1.2M	0.6G	44.1	46.1	1.2M	0.5G
adapt weights	93.0	19.4M	2.3G	66.5	18.4M	7.8G	50.1	53.5	18.5M	7.2G
PosPool (PPNet-S)	92.5	1.2M	0.1G	64.2	1.1M	0.5G	44.6	47.2	1.1M	0.5G
PosPool (PPNet)	92.9	19.4M	1.8G	66.5	18.4M	7.3G	50.0	53.4	18.5M	6.8G
PosPool* (PPNet-S*)	92.6	1.2M	0.1G	61.3	1.1M	0.5G	46.1	47.2	1.1M	0.5G
PosPool* (PPNet*)	93.2	19.4M	1.8G	66.7	18.4M	7.3G	50.6	53.8	18.5M	6.8G

Datasets We consider three datasets with varying scales of training data, task outputs (classification and semantic segmentation) and scenarios (CAD models and real scenes): ModelNet40 [37], S3DIS [1] and PartNet [19]. More details about datasets are described in Section 6.

Performance of Two Baseline Operators For point cloud modeling, the architectures without local aggregation operators also perform well to some extent, e.g. PointNet [20]. To investigate what local aggregation operators perform beyond, we present two baseline functions to replace the local aggregation operators described in Section 3:

$$\mathbf{g}_i = \mathbf{f}_i, \tag{6}$$

$$\mathbf{g}_i = R\left(\{\mathbf{f}_j | j \in \mathcal{N}(i)\}\right). \tag{7}$$

The former is an identity function, without encoding neighborhood points. The latter is an AVG/MAX pool layer without regarding their relative positions.

Table 2. Evaluating different settings of the point-wise MLP method. The option ∇, $\triangle$, $\square$ and $\diamond$ denote input features using $\{\Delta\mathbf{p}_{ij}, \mathbf{f}_j\}$, $\{\mathbf{f}_i, \Delta\mathbf{f}_{ij}\}$, $\{\Delta\mathbf{p}_{ij}, \mathbf{f}_i, \Delta\mathbf{f}_{ij}\}$, and $\{\Delta\mathbf{p}_{ij}, \mathbf{f}_i, \mathbf{f}_j, \Delta\mathbf{f}_{ij}\}$, respectively. "Sweet spot" denotes balanced settings regarding both efficacy and efficiency. The accuracy on PartNet test set is not tested in ablations to avoid the tuning of test set

method	γ	input ∇	input $\triangle$	input $\square$	input $\diamond$	#FC	$R(\cdot)$	ModelNet40	S3DIS	PartNet (val/test)
PointNet++ [22]	-	✓				3	MAX	90.7	-	-/42.5
PointNet++*	-	✓				3	MAX	91.6	55.3	43.1/45.3
sweet spot	8		✓			1	MAX	92.8	62.9	48.2/50.8
	2		✓			1	MAX	92.8	66.2	48.1/51.2
FC num	8		✓			2	MAX	92.5	59.5	47.9/-
	8		✓			3	MAX	92.0	59.9	48.7/-
input	8	✓				1	MAX	92.6	59.8	47.1/-
	8		✓			1	MAX	92.5	61.4	47.6/-
	8			✓		1	MAX	92.7	51.0	47.9/-
reduction $R(\cdot)$	8		✓			1	AVG	92.3	55.1	46.8/-
	8		✓			1	SUM	92.2	44.7	46.7/-

Table 1 shows the accuracy of these two baseline operators using the common architecture in Fig. 1 on three benchmarks. It can be seen that these baseline operators mostly perform marginally worse than the previous best performing methods on the three datasets. The baseline† operator using a MAX pooling layer even slightly outperforms the previous state-of-the-art with smaller computation FLOPs (47.4 mIoU, 6.7G FLOPs vs. 46.4 mIoU, 23.1G FLOPs).

In the following, we will revisit different designing components in the point-wise MLP based methods and the adaptive weight based methods using the common deep residual architecture in Fig. 1. For the pseudo grid feature methods, we choose a representative operator, KPConv [30], with depth-wise convolution kernel and its default grid settings ($M = 15$) for comparison. There are not much hyper-settings for it and we will omit the detailed tuning.

4.2 Performance Study on Point-wise MLP based Method

We start the investigation of this type of methods from a representative method, PointNet++ [22]. We first reproduce this method using its own specific overall architecture and with other implementation details the same as ours. Table 2 (denoted as PointNet++*) shows our reproduction is fairly well, which achieves slightly better accuracy than that reported by the authors [22, 19] on ModelNet40 and PartNet.

We re-investigate several design components for this type of methods using the deep architecture in Fig. 1, including the number of fully connected (FC) layers in an MLP, the choice of input features and the reduction function. Table 2 shows the ablation study on these aspects, with architecture hyper-parameters as: block repeat factor $N_r = 1$, base width $C = 144$ and bottleneck ratio $\gamma = 8$.

We can draw the following conclusions:

- *Number of FC layers.* In literature of this method type, 3 layers are usually used by default to approximate complex functions. Surprisingly, in our experiments, *using 1 FC layer without non-linearity significantly outperforms that using 2 or 3 FC layers on S3DIS*, and it is also competitive on ModelNet40 and PartNet. We hypothesize that the fitting ability by multiple FC layers applied on the concatenation of point feature and relative position may be partly realized by the point-wise transformation layers (the first and the last layers in a residual block) applied on point feature alone. Less FC layers also ease optimization. Using 1 FC layer is also favorable considering the efficiency issue: the computation can be significantly reduced when 1 FC layer is adopted, through computing sharing as explained below.
- *Input Features.* The relative position and edge feature perform similarly on ModelNet40 and PartNet, and combining them has no additional gains. However, on S3DIS datasets, combining both significantly outperforms the variants using each alone.
- *Reduction function.* MAX pooling performs the best, which is in accord with that in literature.

An efficient implementation when 1 FC layer is used. Denote the weight matrix of this only FC layer as $W = [W^1, W^2] \in \mathbb{R}^{d\times(d+3)}$ where $W^1 \in \mathbb{R}^{d\times 3}$ and $W^2 \in \mathbb{R}^{d\times d}$. We have $G = W \cdot \text{concat}(\Delta\mathbf{p}_{ij}, \mathbf{f}_j) = W^1\Delta\mathbf{p}_{ij} + W^2\mathbf{f}_j$. Noting the computation of the second term $W^2\mathbf{f}_j$ can be shared when point j appears in different neighborhoods, the computation complexity of this operator is significantly reduced from $(d+3)ndK$ to $nd^2 + 3ndK$.

Sweet spots for point-wise MLP methods. Regarding both the efficacy and efficiency, the sweet spot settings are applying 1 FC layer to an input combination of relative position and edge features. Table 2 also shows that using $\gamma = 2$ for this method can approach or surpass the state-of-the-art on all three datasets.

4.3 Performance Study on Adaptive Weight based Method

Table 3 shows the ablations over several designing components within this method type, including the number of fully connected (FC) layers, choice of input features, the reduction function and whether to do weight normalization. We adopt architecture hyper-parameters as: block repeat factor $N_r = 1$, base width $C = 144$, and bottleneck ratio $\gamma = 8$.

We can draw the following conclusions:

- *Number of FC layers.* Using 1 FC layer performs noticeably better than that using 2 or 3 layers on S3DIS, and is comparable on ModelNet40 and PartNet.
- *Input features.* Using relative positions alone performs best on all datasets. The accuracy slightly drops with additional position features [16]. The edge features harm the performance, probably because it hinders the effective learning of adaptive weights from relative positions.

Table 3. Evaluating different settings of the adaptive weight based methods. $d\mathbf{p}*$ denotes the 9-dimensional position vector as in [16]. "Sweet spot" denotes balanced settings regarding both efficacy and efficiency. The accuracy on PartNet test set is not tested for ablations to avoid tuning the test set.

method	γ	input				#FC	$R(\cdot)$	S.M.	ModelNet	S3DIS	PartNet (val)
		$\{d\mathbf{p}\}$	$\{d\mathbf{f}\}$	$\{d\mathbf{p}, d\mathbf{f}\}$	$\{d\mathbf{p}^*\}$						
PConv[34]	-	✓				2	SUM		-	58.3	-
FlexConv[5]	-	✓				1	SUM		90.2	56.6	-
sweet spot	8	✓				1	AVG		92.7	62.6	50.0
sweet spot*	2	✓				1	AVG		93.0	66.5	50.1
FC num	8	✓				2	AVG		92.6	61.3	49.9
	8	✓				3	AVG		92.5	58.5	49.6
input	8		✓			1	AVG		85.3	46.6	46.9
	8			✓		1	AVG		82.2	55.7	46.4
	8				✓	1	AVG		92.1	57.0	49.1
reduction	8	✓				1	SUM		92.6	61.7	49.1
	8	✓				1	MAX		92.4	62.3	49.7
SoftMax	8	✓				1	AVG	✓	91.7	55.9	45.8

– *Reduction function.* MAX and AVG functions perform slightly better than SUM function, probably because the MAX and AVG functions are more insensitive to varying neighbor size. We use AVG function by default.

– *SoftMax normalization.* The accuracy significantly drops by SoftMax normalization, probably because the positive weights after normalization let kernels act as low-pass filters and may cause the over-smoothing problem [13].

Sweet spots for adaptive weight based methods. The best performance is achieved by applying 1 FC layer without SoftMax normalization on relative positions alone to compute the adaptive weights. This method also approaches or surpasses the state-of-the-art on all three datasets using a deep residual network.

Discussions Table 1 indicates that the three local aggregation operator types with appropriate settings all achieve the state-of-the-art performance on three representative datasets using the same deep residual architectures. With 16× less parameters and computations (marked by "S"), they also perform competitive compared with the previous state-of-the-art. The sweet spots of different operators also favor a simplicity principle, that the relative position alone and 1 FC layer perform well in most scenarios.

While recent study in point cloud analysis mainly lies in inventing new local aggregation operators, the above results indicate that some of them may worth re-investigation under deeper and residual architectures. These results also stimulate a question: could a much simpler local aggregation operator achieve similar accuracy as the sophisticated ones? In the following section, we will try to answer this question by presenting an extremely simple local aggregation operator.

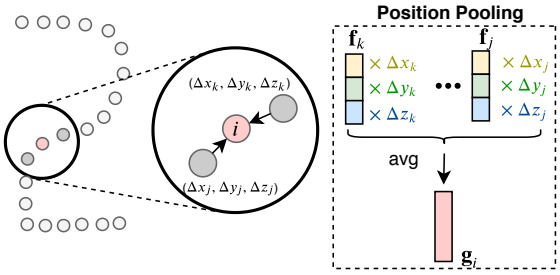


Fig. 2. Illustration of the proposed position pooling (PosPool) operator.

5 PosPool: An Extremely Simple Local Aggregation Operator

In this section, we present a new local aggregation operator, which is extremely simple with no learnable weights.

The new operator is illustrated in Fig. 2. For each neighboring point j, it combines the relative position $\Delta\mathbf{p}_{ij}$ and point feature $\mathbf{f}_j$ by element-wise multiplication. Considering the dimensional difference between the 3-dimensional $\Delta\mathbf{p}_{ij}$ and d-dimensional $\mathbf{f}_j$, the multiplication is applied group-wise that $\Delta\mathbf{p}_{ij}$'s scalars $[\Delta x_{ij}, \Delta y_{ij}, \Delta z_{ij}]$ are multiplied to 1/3 channels of $\mathbf{f}_j$, respectively, as

$$G(\Delta\mathbf{p}_{ij}, \mathbf{f}_j) = \text{Concat}\left[\Delta x_{ij}\mathbf{f}_j^0; \Delta y_{ij}\mathbf{f}_j^1; \Delta z_{ij}\mathbf{f}_j^2\right], \tag{8}$$

where $\mathbf{f}_j^{0,1,2}$ are the 3 sub-vectors equally split from $\mathbf{f}_j$, as $\mathbf{f}_j = \left[\mathbf{f}_j^0; \mathbf{f}_j^1; \mathbf{f}_j^2\right]$.

The operator is named position pooling (PosPool), featured by its property of no learnable weight. It also reserves the *permutation/translation invariance* property which is favorable for point cloud analysis.

A Variant. We also consider a variant of position pooling operator which is slightly more complex, but maintains the no learnable weight property. Instead of using 3-d relative coordinates, we embed the coordinates into a vector with the same dimension as point feature $\mathbf{f}_{ij}$ using cosine/sine functions, similar as in [31]. The embedding is concatenated from $d/6$ group of 6-dimensional vectors, with the m^{th} 6-d vector representing the cosine/sine functions with a wave length of $1000^{6m/d}$ on relative locations x, y, z:

$$\begin{aligned}\mathcal{E}^m(x,y,z) = &[sin(100x/1000^{6m/d}, cos(100x/1000^{6m/d}),\\ &sin(100y/1000^{6m/d}, cos(100y/1000^{6m/d}),\\ &sin(100z/1000^{6m/d}, cos(100z/1000^{6m/d})].\end{aligned} \tag{9}$$

Then an element-wise multiplication operation is applied on the embedding $\mathcal{E}$ and the point feature $\mathbf{f}_{ij}$:

$$G(\Delta\mathbf{p}_{ij}, \mathbf{f}_j) = \mathcal{E} \odot \mathbf{f}_{ij}. \tag{10}$$

The resulting operator also does not have any learnable weights, and is set as a variant of position pooling layer. We find this variant performs slightly better

than the direct multiplication in Eq. (8) in some scenarios. We will show more variants in Appendix A3.

Complexity Analysis The space complexity $\mathcal{O}(\text{space}) = 0$, as there are no learnable weights. The time complexity is also small $\mathcal{O}(\text{time}) = ndK$. Due to the no learnable weight nature, it may also potentially ease the hardware implementation, which does not require an adaption to different learnt weights.

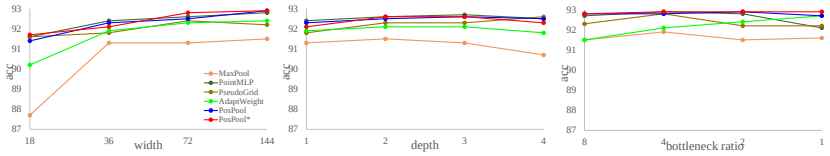
6 Experiments

6.1 Benchmark Settings

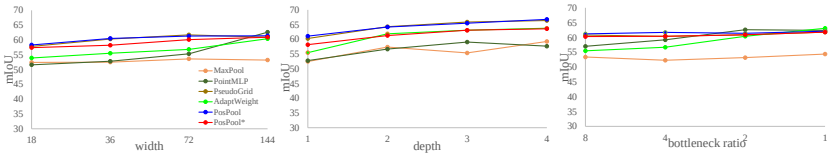
In this section, we detailed the three benchmark datasets with varying scales of training data, task outputs (classification and semantic segmentation) and scenarios (CAD models and real scenes).

- *ModelNet40* [37] is a 3D classification benchmark. This dataset consists of 12,311 meshed CAD models from 40 classes. We follow the official data splitting scheme in [37] for training/testing. We adopt an input resolution of 5,000 and a base grid size of 2cm.
- *S3DIS* [1] is a real indoor scene segmentation dataset with 6 large scale indoor areas captured from 3 different buildings. 273 million points are annotated and classified into 13 classes. We follow [28] and use Area-5 as the test scene and all others for training. In both training and test, we segment small sub-clouds in spheres with radius of 2m. In training, the spheres are randomly selected in scenes. In test, we select spheres regularly in the point clouds. We adopt a base grid size of 4cm.
- *PartNet* [19] is a more recent challenging benchmark for large-scale fine-grained part segmentation. This dataset consists of pre-sampled point clouds of $26,671$ 3D object models in 24 object categories, with each object containing 18 parts on average. This dataset is officially split into three parts: 70% training, 10% validation, and 20% test sets. We train our model with official training dataset and then conduct the comparison study on the validation set on 17 categories with fine-grained annotation. We also report the best accuracies of different methods on the test set. We use the 10,000 points provided with the datasets as input, and the base grid size is set as 2cm.

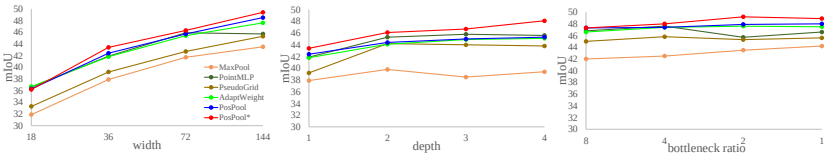
The training/inference settings are detailed in Appendix A1. Note for PartNet datasets, while in [19] independent networks are trained for 17 different shapes, we adopt a shared backbone and independent 3 fully connected layers for part segmentation of different categories and train all the categories together, which significantly facilitate the evaluation on this dataset. We note using the shared backbone network achieves similar accuracy than the methods training different shapes independently.



(a) Evaluation on ModelNet40 datasets with different width, depth and bottleneck ratio



(b) Evaluation on S3DIS datasets with different width, depth and bottleneck ratio



(c) Evaluation on PartNet datasets with different width, depth and bottleneck ratio

Fig. 3. Accuracy of different methods with varying width (C), depth ($N_r + 1$) and bottleneck ratio (γ) on three benchmark datasets.

6.2 Comparing Operators with Varying Architecture Capacity

Fig. 3 shows comparison of different local aggregation operators using architectures with different model capacity on three benchmarks, by varying the network width, depth and bottleneck ratio. Detailed experimental settings are presented in Appendix A2. It can be seen: the PosPool operators achieve top or close-to-top performances using varying network hyper-parameters on all datasets, showing its strong stability and adaptability. While the other more sophisticated operators may achieve similar accuracy with the PosPool layers on some datasets or settings, their performance are less stable across scenarios and model capacity. For example, the accuracy of the "AdaptWeight" method will drop significantly on S3DIS when the model capacity is reduced by either the width, depth or bottleneck ratio.

7 Conclusion

This paper studies existing local aggregation operators in depth via a carefully designed common testbed that consists of a deep residual architecture and three representative benchmarks. Our investigation illustrates that with appropriate settings, all operators can achieve the state-of-the-art performance on three tasks. Motivated by this finding, we present a new extremely simple operator without learned weights, which performs as good as existing operators with sophisticated design. We hope our study and new design can encourage further rethinking and understanding on the role of local aggregation operators and shed new light to future network design.

References

1. Armeni, I., Sax, A., Zamir, A.R., Savarese, S.: Joint 2D-3D-Semantic Data for Indoor Scene Understanding. ArXiv e-prints (Feb 2017)
2. Atzmon, M., Maron, H., Lipman, Y.: Point convolutional neural networks by extension operators. ACM Transactions on Graphics **37**(4), 1–12 (Jul 2018)
3. Feng, Y., Zhang, Z., Zhao, X., Ji, R., Gao, Y.: Gvcnn: Group-view convolutional neural networks for 3d shape recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 264–272 (2018)
4. Gadelha, M., Wang, R., Maji, S.: Multiresolution tree networks for 3d point cloud processing. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 103–118 (2018)
5. Groh, F., Wieschollek, P., Lensch, H.P.: Flex-convolution. In: Asian Conference on Computer Vision. pp. 105–122. Springer (2018)
6. Guo, H., Wang, J., Gao, Y., Li, J., Lu, H.: Multi-view 3d object retrieval with deep embedding network. IEEE Transactions on Image Processing **25**(12), 5526–5537 (2016)
7. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition (2015)
8. Hermosilla, P., Ritschel, T., Vázquez, P.P., Vinacua, A., Ropinski, T.: Monte carlo convolution for learning on non-uniformly sampled point clouds. ACM Transactions on Graphics **37**(6), 1–12 (Dec 2018)
9. Hua, B.S., Tran, M.K., Yeung, S.K.: Pointwise convolutional neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 984–993 (2018)
10. Klokov, R., Lempitsky, V.: Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 863–872 (2017)
11. Komarichev, A., Zhong, Z., Hua, J.: A-cnn: Annularly convolutional neural networks on point clouds. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 7421–7430 (2019)
12. Lan, S., Yu, R., Yu, G., Davis, L.S.: Modeling local geometric structure of 3d point clouds using geo-cnn. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 998–1008 (2019)
13. Li, G., Müller, M., Thabet, A., Ghanem, B.: Can gcns go as deep as cnns? arXiv preprint arXiv:1904.03751 (2019)
14. Li, Y., Bu, R., Sun, M., Wu, W., Di, X., Chen, B.: Pointcnn: Convolution on x-transformed points. In: Advances in Neural Information Processing Systems. pp. 820–830 (2018)
15. Liu, Y., Fan, B., Meng, G., Lu, J., Xiang, S., Pan, C.: Densepoint: Learning densely contextual representation for efficient point cloud processing. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 5239–5248 (2019)
16. Liu, Y., Fan, B., Xiang, S., Pan, C.: Relation-shape convolutional neural network for point cloud analysis. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 8895–8904 (2019)
17. Mao, J., Wang, X., Li, H.: Interpolated convolutional networks for 3d point cloud understanding. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 1578–1587 (2019)
18. Maturana, D., Scherer, S.: Voxnet: A 3d convolutional neural network for real-time object recognition. In: 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 922–928. IEEE (2015)

19. Mo, K., Zhu, S., Chang, A.X., Yi, L., Tripathi, S., Guibas, L.J., Su, H.: Partnet: A large-scale benchmark for fine-grained and hierarchical part-level 3d object understanding. In: CVPR (2019)
20. Qi, C.R., Su, H., Mo, K., Guibas, L.J.: Pointnet: Deep learning on point sets for 3d classification and segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 652–660 (2017)
21. Qi, C.R., Su, H., Nießner, M., Dai, A., Yan, M., Guibas, L.J.: Volumetric and multi-view cnns for object classification on 3d data. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 5648–5656 (2016)
22. Qi, C.R., Yi, L., Su, H., Guibas, L.J.: Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In: NIPS (2017)
23. Riegler, G., Osman Ulusoy, A., Geiger, A.: Octnet: Learning deep 3d representations at high resolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 3577–3586 (2017)
24. Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation. MICCAI p. 234–241 (2015)
25. Su, H., Maji, S., Kalogerakis, E., Learned-Miller, E.: Multi-view convolutional neural networks for 3d shape recognition. In: Proceedings of the IEEE international conference on computer vision. pp. 945–953 (2015)
26. Tatarchenko, M., Dosovitskiy, A., Brox, T.: Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 2088–2096 (2017)
27. Tatarchenko, M., Park, J., Koltun, V., Zhou, Q.Y.: Tangent convolutions for dense prediction in 3d. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 3887–3896 (2018)
28. Tchapmi, L., Choy, C., Armeni, I., Gwak, J., Savarese, S.: Segcloud: Semantic segmentation of 3d point clouds. In: 2017 International Conference on 3D Vision (3DV). pp. 537–547. IEEE (2017)
29. Thomas, H., Goulette, F., Deschaud, J.E., Marcotegui, B., LeGall, Y.: Semantic classification of 3d point clouds with multiscale spherical neighborhoods. 2018 International Conference on 3D Vision (3DV) (Sep 2018)
30. Thomas, H., Qi, C.R., Deschaud, J.E., Marcotegui, B., Goulette, F., Guibas, L.J.: Kpconv: Flexible and deformable convolution for point clouds. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 6411–6420 (2019)
31. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need (2017)
32. Wang, L., Huang, Y., Hou, Y., Zhang, S., Shan, J.: Graph attention convolution for point cloud semantic segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 10296–10305 (2019)
33. Wang, P.S., Liu, Y., Guo, Y.X., Sun, C.Y., Tong, X.: O-cnn: Octree-based convolutional neural networks for 3d shape analysis. ACM Transactions on Graphics (TOG) **36**(4), 72 (2017)
34. Wang, S., Suo, S., Ma, W.C., Pokrovsky, A., Urtasun, R.: Deep parametric continuous convolutional neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2589–2597 (2018)
35. Wang, Y., Sun, Y., Liu, Z., Sarma, S.E., Bronstein, M.M., Solomon, J.M.: Dynamic graph cnn for learning on point clouds. ACM Transactions on Graphics (TOG) **38**(5), 1–12 (2019)
36. Wu, W., Qi, Z., Fuxin, L.: Pointconv: Deep convolutional networks on 3d point clouds. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 9621–9630 (2019)

37. Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., Xiao, J.: 3d shapenets: A deep representation for volumetric shapes. In: CVPR (2015)
38. Zhang, Z., Hua, B.S., Yeung, S.K.: Shellnet: Efficient point cloud convolutional neural networks using concentric shells statistics. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 1607–1616 (2019)