Supplementary Material for

PAConv: Position Adaptive Convolution with Dynamic Kernel Assembling on Point Clouds

Outline

This supplementary document is arranged as follows:
(1) Sec. A benchmarks the performance of the most recent point convolutional **operators** under the same backbone architecture and the same data augmentation strategies;
(2) Sec. B investigates the effects of ScoreNet on PAConv;
(3) Sec. C elaborates on network configurations and implementation strategies for different down-stream tasks;
(4) Sec. D presents a CUDA implementation of PAConv;
(5) Sec. E lists detailed semantic segmentation results with per-category scores;
(6) Sec. F visualizes the results of the baseline (*i.e.* PointNet++) and ours (*i.e.* PointNet++ equipped with our PAConv) to facilitate the comparisons.

A. Comparison of Point Convolutional Operators

In this section, we focus on comparing the capability of PAConv with other different point convolutional **operators**. We compare with the most recent convolutional operators – PointConv [27] and KPConv [22]. To minimize the influence of the network architectures, we choose the classical MLP-based network PointNet++ [18] as the backbone and integrate other convolution operators by directly replacing the MLPs, following how we embed our PAConv into PointNet++ as mentioned in Sec. C.2.

Specifically, we replace the MLPs in PointNet++ with the PointConv * [27] and KPConv † [22] operators, while ensure not making any changes on the original network architecture and feature-dims of the original PointNet++. All the experiments are conducted on the S3DIS [1] dataset, and adopt the Area-5 evaluation protocol, under the same data augmentation strategies for fair comparisons.

As summarized in Table. I, our PAConv improves the mIoU of PointNet++ by 9.31%↑ with decent efficiency. However, PointConv only promotes the mIoU of PointNet++ with 2.7%↑ while the inference is time-consuming with tremendous amount of FLOPs.

Note: Since the radius to initialize kernel points in KPConv [22] need to be specifically tuned for different point cloud scales, it is tricky to adjust this radius in our implementation. Specifically, following the official code † of KPConv, we have tried exhaustive search to set the radius ranging from $0.07 \sim 0.6$, which is multiplied by 2 at each downsampling layer. However, the mIoU can only reach $11\% \sim 26\%$. This result is not comparable with its original version, thus it is not reported here.

Method	mIoU
PointNet++ [18]	57.27
PointConv * [27]	59.97
KPConv * [22]	-
PAConv *	**66.58**

Table I. Segmentation results (%) of PointNet++ [18], PointConv [27] and our PAConv on S3DIS Area-5. * indicates plugging the corresponding convolution operator to the original PointNet++ [18] network **without** changing network configurations. KPConv [22] is not reported due to the result reproduced by our implementation is not comparable with its original version.

Compared with KPConv, our PAConv does not require either complicated design of network architecture or handcrafted adjustment of kernel point space, which is a more flexible and efficient point convolutional operator, adaptable to different applications.

B. More Explorations on ScoreNet

ScoreNet Depth. The ScoreNet in PAConv consists of several fully connected layers with feature dim $[f_1, ..., f_d]$. Here we aim to figure out how the depth of ScoreNet influences the performance of our PAConv. Same with the ablation studies in the main paper, this experiment is conducted on PAConv without adding correlation loss on the S3IDS dataset.

Table. II shows the result, where the corresponding time complexity (floating point operations/sample) of each setting is also enumerated for developers to balance the performance and efficiency. We clearly see that a

* https://github.com/DylanWusee/pointconv_pytorch
† https:///github.com/HuguesTHOMAS/KPConv-PyTorch

ScoreNet Layer	mIoU	FLOPs/sample(M)
[16]	64.42	1178
[16, 16]	65.29	1215
[16, 16, 16]	**65.63**	1253

Table II. Segmentation results (%) and #FLOPs/sample (M) of PAConv on the S3DIS dataset using different ScoreNet depth settings. Area-5 evaluation is adaopted. Deeper ScoreNet brings better performance but has lower efficiency.

deeper ScoreNet brings better performance while has lower efficiency.

Score Distribution in the Network. Furthermore, we visualize the average score coefficients of each weight matrix B_m for all points at different network layer depths on S3DIS Area-5 segmentation task, aiming to figure out how scores distribute in the network.

As illustrated in Fig. II, the scores of different weight matrices are diversely distributed (*i.e.*, non-uniform and not only focus on single weight matrix) at all layers, which means nearly all the weight matrices in the weight bank have the possibility to be chosen for assembling point convolution kernels. This proves that the weight matrices in our PAConv are fully utilized, bringing more flexibility in the dynamic kernel assembling.

Score Distribution in 3D Space. Following Sec.6.1 of the main paper, we provide more visualizations to demonstrate the spatial distribution of scores. As demonstrated in Fig. I, different weight matrices capture different position relations in 3D space.

C. Network Configurations and Implementations

Our PAConv is implemented using Pytorch [16]. A data-parallel training scheme is adopted on several Nvidia GeForce GTX 2080 Ti GPUs. The details of networks and training strategies for different tasks are illustrated below.

C.1. Object-Level Tasks

Network Configurations. As mentioned in the main paper, our PAConv is purely embedded into simple classical MLP-based point cloud networks ***without*** any modifications on network architectures or parameters (*i.e.* feature dimensions).

We employ PointNet [17] and DGCNN [26] as the backbones for object-level tasks (*i.e.* object classification and part segmentation). Due to the similar architecture design of DGCNN and PointNet, we only provide the network architecture of DGCNN in Fig. III. It clearly shows that the scale/resolution of the point cloud is fixed across whole networks.

The feature dimensions are the same as the official code of DGCNN [26] ([‡] for classification and [§] for part segmentation). Several MLPs with a dropout probability of 0.5 are employed at the last feature layers of the network. The dimensions of the fully connected layers are (512, 256, C_{out}) for generating final classification scores, and (256, 256, 128, C_{out}) to obtain final per-point segmentation scores for part segmentation. All layers include ReLU and batch normalization except for the last score prediction layer. As for the part segmentation, the one-hot encoding (16-d) of the object label is concatenated to the last feature layer.

We set the number of neighbors in KNN search to 20 for classification and 30 for part segmentation when building neighborhood in Euclidean space at each PAConv.

Training. We follow the official code of DGCNN [26] to train the network.

For the classification task [‡], we use SGD with learning rate 0.1 and reduce it to 0.001 with cosine annealing. The momentum is 0.9 and the weight decay is 10^{-4}. The batch size is set to 32.

For the segmentation task [§], Adam with learning rate 0.003 is employed and is divided by 2 after every 40 epochs. The weight decay is 0 and the batch size is 32. Both classification and part segmentation networks converge in 350 epochs.

C.2. Scene-Level Task

Network Configurations. For scene level tasks, we choose PointNet++ [18] with encoding (downsampling) and decoding (upsampling) layers as the backbone. As shown in Fig. IV, we directly replace the PointNet modules (*i.e.* MLPs) with our PAConv for local pattern learning in the encoding layers of PointNet++ ***without*** changing any other network configurations.

Each encoding layer takes an $N \times (d + C)$ matrix as input that is from N points with d-dim coordinates and C-dim point feature. It outputs an $N^{'} \times (d + C^{'})$ matrix of $N^{'}$ subsampled points with d-dim coordinates and new $C^{'}$-dim feature vectors summarizing local context.

Our decoding layers are totally the same as PointNet++. Concretely, for each query point at each layer in the decoder, the point feature set is first upsampled through a nearest-neighbor interpolation based on the inverse distance weighted averagely among k nearest neighbors of the query point. Next, the upsampled feature maps are concatenated with the intermediate feature maps produced by encoding layers through skip connections, after which a shared MLP is applied to the concatenated feature maps.

[‡] https://github.com/WangYueFt/dgcnn/blob/master/pytorch

[§] https://github.com/WangYueFt/dgcnn/blob/master/tensorflow/part_seg

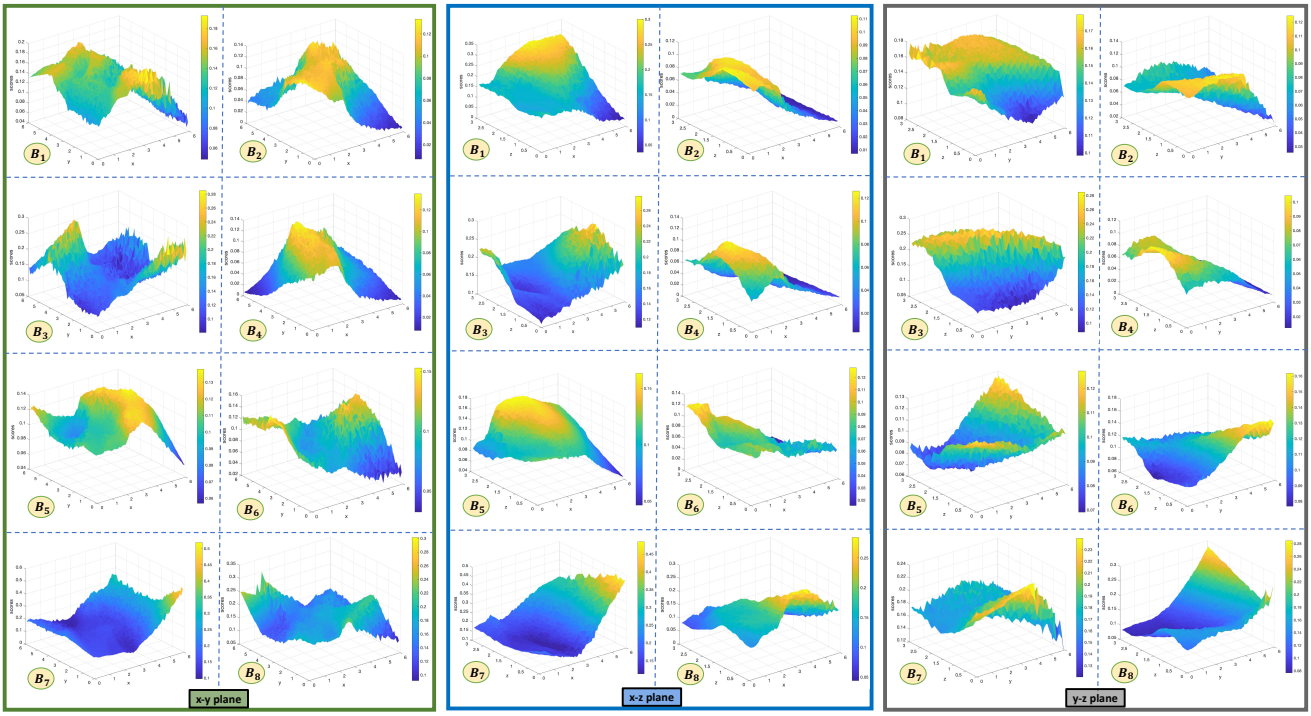

Figure I. The spatial distribution of scores, where the input points are randomly initialized and are sent to a trained ScoreNet. When the corresponding height of a point is higher (or the color is closer to yellow), the output score of this point is larger. It illustrates the relation between spatial positions and score distributions for each weight matrix B_m.

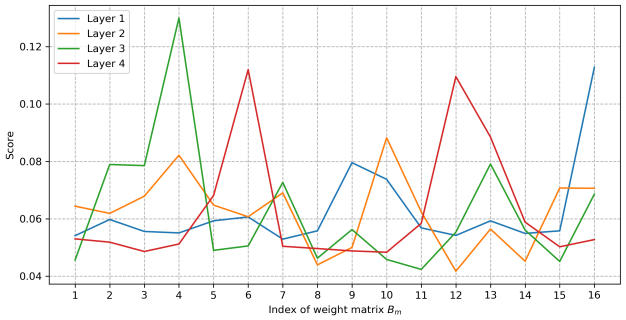

Figure II. Average score coefficient of each weight matrix B_m at different layer depths. The corresponding score of each weight matrix is diversely distributed, indicating that all the weight matrices are fully utilized for assembling point convolution kernels.

Same with PointNet++ [18], we use the following notations to describe our network architecture. EN$(N, [l_1, ..., l_d])$ is an encoding layer with N query points using d consecutive PAConv with feature dim $l_i (i = 1, ..., d)$. DE$(l_1, ..., l_d)$ is a decoding layer with d MLPs, which is used for updating features concatenated from interpolation and skip link. With these notations, the network can be represented as:

$$\begin{aligned}
&\text{EN}(1024, [32, 32, 64]) \\
&\downarrow \\
&\text{EN}(256, [64, 64, 128]) \\
&\downarrow \\
&\text{EN}(64, [128, 128, 256]) \\
&\downarrow \\
&\text{EN}(16, [256, 256, 512]) \\
&\downarrow \\
&\text{DE}(256, 256) \\
&\downarrow \\
&\text{DE}(256, 256) \\
&\downarrow \\
&\text{DE}(256, 128) \\
&\downarrow \\
&\text{DE}(128, 128, 128)
\end{aligned}$$

Additionally, in each PAConv, we choose the nearest 32 points ($K = 32$) in Euclidean space as the neighboring points for each query point. Each layer is followed by ReLU and batch normalization except for the last score prediction layer. At last, several MLPs (128, 128, C_{out}) with dropout ratio 0.5 is utilized to output the final point-wise scores for semantic segmentation.

Training. We employ SGD optimizer with initial learning rate 0.05 and divide it by 10 at the 60*th* and 80*th* epoch. The

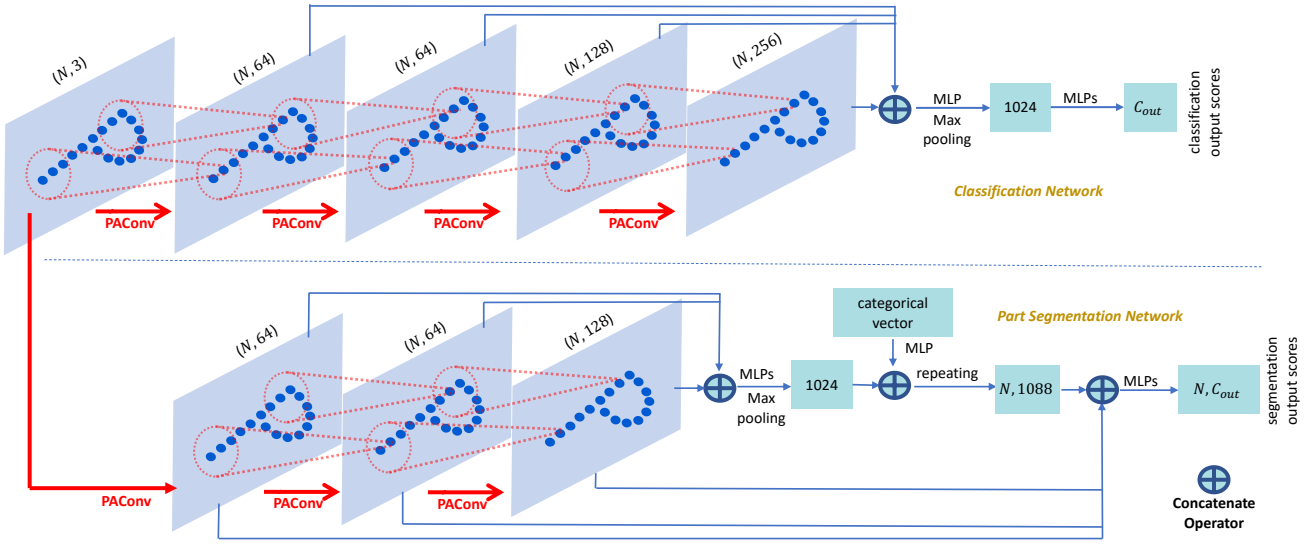

Figure III. Network based on DGCNN [26] for object classification and shape part segmentation. The architecture is **totally same** with the official source code of DGCNN, where the *EdgeConv* of DGCNN is replaced by our PAConv. We observe that the scale/resolution of the point cloud is fixed across the whole network.

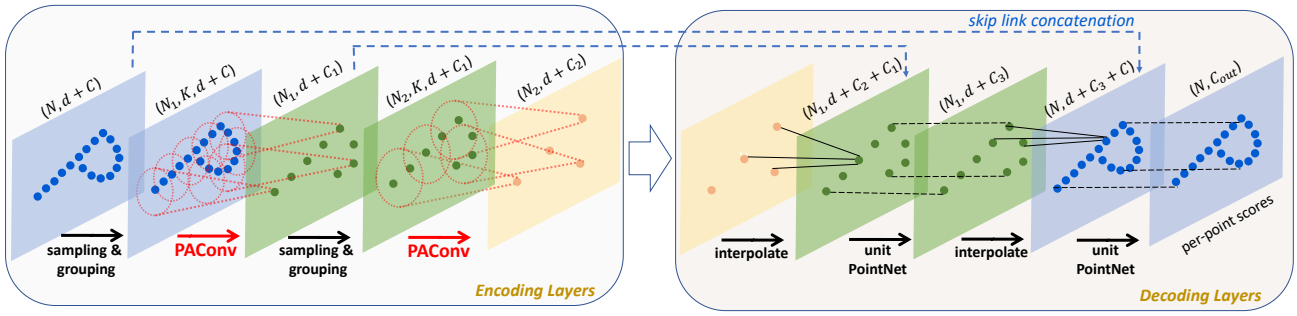

Figure IV. Large scene segmentation network built on PointNet++ [18]. We directly replace the PointNet (*i.e.* MLPs) operations with our PAConv for local feature representation in the encoding layers of PointNet++ **without** changing any other network configurations.

momentum is 0.9 and the weight decay is 10^{-4}. The batch size is 16 and the total number of epochs is 100.

D. CUDA Implementation

By re-formulating Eq. (3) of the main paper to $g_i = \Lambda_{j \in \mathcal{N}_i} \sum_{m=1}^{M}((B_m f_j) S_{ij}^m)$, PAConv can be realized equivalently by first transforming features using weight matrices, then assembling transformed features with scores. We implement a CUDA layer to assemble neighbor features by querying neighbor indices on-the-fly without storing large intermediate matrix (Fig. V). This reduces memory usage from **10G+ to 5600M** with 65,536 points. The CUDA code is also released.

Note that the performance on S3DIS semantic segmentation is slightly different between CUDA version and the original version. In CUDA version, since we only maintain one feature for each point on-the-fly regardless of the local area it belongs to, it is necessary to apply neighboring feature aggregation after each PAConv layer. However, our original version exactly follows PointNet++ [18], where neighboring point features are firstly respectively refined in each local area by three continuous PAConv layers, and are aggregated then.

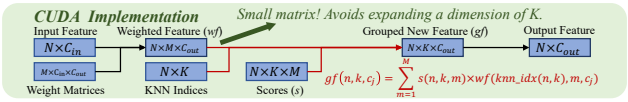

Figure V. The CUDA implementation of our PAConv.

E. More Segmentation Results

We provide more detailed segmentation results of our PAConv and all the other methods listed in the main paper.

First, we summarize the segmentation results of each category on S3DIS [1], plus mean of class-wise accuracy (mAcc). As shown in Table. III, our PAConv with $\mathcal{L}_{corr}$ achieves the **best mAcc** among all the approaches. Without adopting the computation and memory intensive grid sampling strategy, our approach compares on par with KPConv with deformable design.

Next, Table. IV reports the results on S3DIS with 6-fold cross validation (calculating the metrics with results from different folds merged), where our method achieves comparable performance with the state-of-the-arts. To be noted, we do not use grid sampling.

Last, Table V enumerates the mIoU of each class on ShapeNet Part [29] for shape part segmentation task.

F. Visualization of Result Comparisons on S3DIS

Since we employ PointNet++ [18] as the backbone for the indoor scene segmentation on S3DIS [1], we provide the visualization results to intuitively compare the performance between original PointNet++ and PointNet++ after integrating PAConv. As shown in Fig. VI, PointNet++ equipped with PAConv achieves conspicuously stronger performance than original PointNet++ on various scenes or areas.

Method	Pre.	mIoU	mAcc	ceil.	floor	wall	beam	col.	wind.	door	chair	table	book	sofa	board	clut.
PointNet [17]	*BLK*	41.09	48.98	88.80	97.33	69.80	0.05	3.92	46.26	10.76	52.61	58.93	40.28	5.85	26.38	33.22
SegCloud [21]	*BLK*	48.92	57.35	90.06	96.05	69.86	0.00	18.37	38.35	23.12	75.89	70.40	58.42	40.88	12.96	41.60
TangentConv [20]	*BLK*	52.6	62.2	90.5	97.7	74.0	0.0	20.7	39.0	31.3	69.4	77.5	38.5	57.3	48.8	39.8
PointCNN [7]	*BLK*	57.26	63.86	92.31	98.24	79.41	0.00	17.60	22.77	62.09	80.59	74.39	66.67	31.67	62.05	56.74
ParamConv [25]	*BLK*	58.27	67.01	92.26	96.20	75.89	0.27	5.98	69.49	63.45	66.87	65.63	47.28	68.91	59.10	46.22
PointWeb [5]	*BLK*	60.28	66.64	91.95	98.48	79.39	0.00	21.11	59.72	34.81	88.27	76.33	69.30	46.89	64.91	52.46
PointEdge [4]	*BLK*	61.85	68.30	91.47	98.16	81.38	0.00	23.34	65.30	40.02	87.70	75.46	67.78	58.45	65.61	49.36
GACNet [24]	*BLK*	62.85	-	92.28	98.27	81.90	0.00	20.35	59.07	40.85	85.80	78.54	70.75	61.70	74.66	52.82
Point2Node [3]	*BLK*	62.96	70.02	93.88	98.26	83.30	0.00	35.65	55.31	58.78	84.67	79.51	71.13	44.07	58.72	55.17
KPConv *rigid* [22]	*Grid*	65.4	70.9	92.6	97.3	81.4	0.0	16.5	54.5	69.5	90.1	80.2	74.6	66.4	63.7	58.1
KPConv *deform*[22]	***Grid***	**67.1**	**72.8**	92.8	97.3	82.4	0.0	23.9	58.0	69.0	91.0	81.5	75.3	75.4	66.7	58.9
FPConv [8]	*BLK*	62.8	-	94.6	98.5	80.9	0.0	19.1	60.1	48.9	88.0	80.6	68.4	53.2	68.2	54.9
SegGCN [6]	*BLK*	63.6	70.44	93.7	98.6	80.6	0.0	28.5	42.6	74.5	88.7	80.9	71.3	69.0	44.4	54.3
PosPool [13]	*Grid*	66.7	-	-	-	-	-	-	-	-	-	-	-	-	-	-
PointNet++ [18]	*BLK*	57.27	63.54	91.31	96.92	78.73	0.00	15.99	54.93	31.88	83.52	74.62	67.24	49.31	54.15	45.89
PAConv w/o $\mathcal{L}_{corr}$ (**PN2*)	*BLK*	65.63	72.75	93.10	98.42	82.64	0.00	22.59	61.30	63.31	87.95	78.49	73.46	64.51	70.10	57.26
PAConv w/ $\mathcal{L}_{corr}$ (**PN2*)	***BLK***	**66.58**	**73.00**	94.55	98.59	82.37	0.00	26.43	57.96	59.96	89.73	80.44	74.32	69.80	73.50	57.72

Table III. Segmentation results (%) on S3DIS Area-5. *BLK* and *Grid* signify using block sampling and grid sampling in data pre-processing, respectively. **PN2* refers to applying PointNet++ [18] as the backbone. The best results under block and grid sampling are respectively highlighted.

Method	Pre.	mIoU	mAcc	ceil.	floor	wall	beam	col.	wind.	door	chair	table	book	sofa	board	clut.
PointNet [17]	*BLK*	47.6	66.2	88.0	88.7	69.3	42.4	23.1	47.5	51.6	42.0	54.1	38.2	9.6	29.4	35.2
SegCloud [21]	*BLK*	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
TangentConv [20]	*BLK*	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
PointCNN [7]	*BLK*	65.39	75.61	94.78	97.30	75.82	63.25	51.71	58.38	57.18	69.12	71.63	61.15	39.08	52.19	58.59
ParamConv [25]	*BLK*	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
PointWeb [5]	*BLK*	66.73	76.19	93.54	94.21	80.84	52.44	41.33	64.89	68.13	67.05	71.35	62.68	50.34	62.20	58.49
PointEdge [4]	*BLK*	67.83	76.26	-	-	-	-	-	-	-	-	-	-	-	-	-
GACNet [24]	*BLK*	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Point2Node [3]	*BLK*	70.00	79.10	94.08	97.28	83.42	62.68	52.28	72.31	64.30	70.78	75.77	49.83	65.73	60.26	60.90
KPConv *rigid* [22]	*Grid*	69.6	78.1	93.7	92.0	82.5	62.5	49.5	65.7	77.3	57.8	64.0	68.8	71.7	60.1	59.6
KPConv *deform*[22]	*Grid*	70.6	79.1	93.6	92.4	83.1	63.9	54.3	66.1	76.6	57.8	64.0	69.3	74.9	61.3	60.3
FPConv [8]	*BLK*	68.7	-	94.8	97.5	82.6	42.8	41.8	58.6	73.4	81.0	71.0	61.9	59.8	64.2	64.2
SegGCN [6]	*BLK*	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
PosPool [13]	*Grid*	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
PointNet++ [18]	*BLK*	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
PAConv w/o $\mathcal{L}_{corr}$ (**PN2*)	*BLK*	69.31	78.65	94.30	93.46	82.80	56.88	45.74	65.21	74.90	59.74	74.60	67.41	61.78	65.79	58.36

Table IV. Segmentation results (%) on S3DIS with 6-fold cross validation. *BLK* and *Grid* signify using block sampling or grid sampling in data pre-processing, respectively. **PN2* refers to applying PointNet++ [18] as the backbone.

Method (time order)	Class mIoU	Inst. mIoU	aero	bag	cap	car	chair	ear phone	guitar	knife	lamp	lap top	motor	mug	pistol	rocket	skate board	table
PointNet [17]	80.4	83.7	83.4	78.7	82.5	74.9	89.6	73.0	91.5	85.9	80.8	95.3	65.2	93.0	81.2	57.9	72.8	80.6
PointNet++ [18]	81.9	85.1	82.4	79.0	87.7	77.3	90.8	71.8	91.0	85.9	83.7	95.3	71.6	94.1	81.3	58.7	76.4	82.6
SynSpecCNN [30]	82.0	84.7	81.6	81.7	81.9	75.2	90.2	74.9	93.0	86.1	84.7	95.6	66.7	92.7	81.6	60.6	82.9	82.1
SPLATNet [19]	83.7	85.4	83.2	84.3	89.1	80.3	90.7	75.5	92.1	87.1	83.9	96.3	75.6	95.8	83.8	64.0	75.5	81.8
PCNN [2]	81.8	85.1	82.4	80.1	85.5	79.5	90.8	73.2	91.3	86.0	85.0	95.7	73.2	94.8	83.3	51.0	75.0	81.8
SpiderCNN [28]	82.4	85.3	83.5	81.0	87.2	77.5	90.7	76.8	91.1	87.3	83.3	95.8	70.2	93.5	82.7	59.7	75.8	82.8
SpecGCN [23]	-	85.4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
PointCNN [7]	84.6	86.1	84.1	86.5	86.0	80.8	90.6	79.7	92.3	88.4	85.3	96.1	77.2	95.3	84.2	64.2	80.0	83.0
PointConv [27]	82.8	85.7	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Point2Seq [10]	82.2	85.2	82.6	81.8	87.5	77.3	90.8	77.1	91.1	86.9	83.9	95.7	70.8	94.6	79.3	58.1	75.2	82.8
PVCNN [14]	-	86.2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
RS-CNN [12]	84.0	86.2	83.5	84.8	88.8	79.6	91.2	81.1	91.6	88.4	86.0	96.0	73.7	94.1	83.4	60.5	77.7	83.6
InterpCNN [15]	84.0	86.3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
KPConv *rigid* [22]	85.0	86.2	83.8	86.1	88.2	81.6	91.0	80.1	92.1	87.8	82.2	96.2	77.9	95.7	86.8	65.3	81.7	83.6
KPConv *deform* [22]	**85.1**	**86.4**	84.6	86.3	87.2	81.1	91.1	77.8	92.6	88.4	82.7	96.2	78.1	95.8	85.4	69.0	82.0	83.6
DensePoint [11]	84.2	86.4	84.0	85.4	90.0	79.2	91.1	81.6	91.5	87.5	84.7	95.9	74.3	94.6	82.9	64.6	76.8	83.7
3D-GCN [9]	82.1	85.1	83.1	84.0	86.6	77.5	90.3	74.1	90.0	86.4	83.8	95.6	66.8	94.8	81.3	59.6	75.7	82.8
DGCNN [26]	82.3	85.2	84.0	83.4	86.7	77.8	90.6	74.7	91.2	87.5	82.8	95.7	66.3	94.9	81.1	63.5	74.5	82.6
PAConv (**DGC*)	84.6	86.1	84.3	85.0	90.4	79.7	90.6	80.8	92.0	88.7	82.2	95.9	73.9	94.7	84.7	65.9	81.4	84.0

Table V. Segmentation results (%) on ShapeNet Part dataset. **DGC* indicates using DGCNN [26] as the backbone.

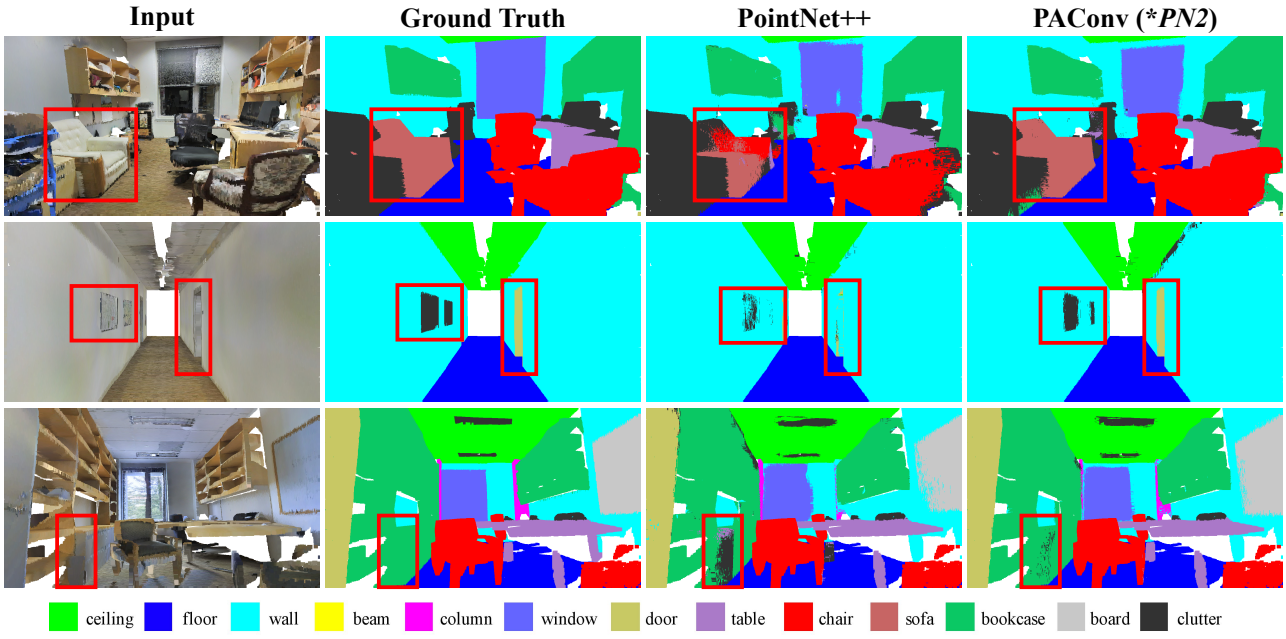

Figure VI. Visualization of semantic segmentation results on S3DIS Area-5. The first column is original scene inputs, the second column is the ground truth of the segmentation, the third row is the scenes segmented by the backbone PointNet++ [18], and the last column shows the segmentation results of plugging our PAconv into PointNet++. Each row denotes a scene in S3DIS Area-5. The red bounding boxes indicate the specific areas, where our PAConv has significantly better performance than PointNet++.

References

[1] I. Armeni, O. Sener, A. R. Zamir, H. Jiang, I. Brilakis, M. Fischer, and S. Savarese. 3d semantic parsing of large-scale indoor spaces. In *CVPR*, 2016. 1, 5

[2] Matan Atzmon, Haggai Maron, and Yaron Lipman. Point convolutional neural networks by extension operators. *ACM Trans. Graph.*, 2018. 7

[3] Wenkai Han, Chenglu Wen, Cheng Wang, Xin Li, and Qing Li. Point2node: Correlation learning of dynamic-node for point cloud feature modeling. In *AAAI*, 2020. 6

[4] L. Jiang, H. Zhao, S. Liu, X. Shen, C. Fu, and J. Jia. Hierarchical point-edge interaction network for point cloud semantic segmentation. In *ICCV*, 2019. 6

[5] Li Jiang, Hengshuang Zhao, Shu Liu, Xiaoyong Shen, Chi-Wing Fu, and Jiaya Jia. Hierarchical point-edge interaction network for point cloud semantic segmentation. In *ICCV*, 2019. 6

[6] Huan Lei, Naveed Akhtar, and Ajmal Mian. Seggcn: Efficient 3d point cloud segmentation with fuzzy spherical kernel. In *CVPR*, 2020. 6

[7] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. Pointcnn: Convolution on x-transformed points. In *NeurIPS*. 2018. 6, 7

[8] Yiqun Lin, Zizheng Yan, Haibin Huang, Dong Du, Ligang Liu, Shuguang Cui, and Xiaoguang Han. Fpconv: Learning local flattening for point convolution. In *CVPR*, 2020. 6

[9] Zhi-Hao Lin, Sheng-Yu Huang, and Yu-Chiang Frank Wang. Convolution in the cloud: Learning deformable kernels in 3d graph convolution networks for point cloud analysis. In *CVPR*, 2020. 7

[10] 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 *AAAI*, 2019. 7

[11] Yongcheng Liu, Bin Fan, Gaofeng Meng, Jiwen Lu, Shiming Xiang, and Chunhong Pan. Densepoint: Learning densely contextual representation for efficient point cloud processing. In *ICCV*, 2019. 7

[12] Yongcheng Liu, Bin Fan, Shiming Xiang, and Chunhong Pan. Relation-shape convolutional neural network for point cloud analysis. In *CVPR*, 2019. 7

[13] Ze Liu, Han Hu, Yue Cao, Zheng Zhang, and Xin Tong. A closer look at local aggregation operators in point cloud analysis. In *ECCV*, 2020. 6

[14] Zhijian Liu, Haotian Tang, Yujun Lin, and Song Han. Point-voxel cnn for efficient 3d deep learning. In *NeurIPS*, 2019. 7

[15] Jiageng Mao, Xiaogang Wang, and Hongsheng Li. Interpolated convolutional networks for 3d point cloud understanding. In *ICCV*, 2019. 7

[16] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019. 2

[17] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *CVPR*, 2017. 2, 6, 7

[18] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *NeurIPS*. 2017. 1, 2, 3, 4, 5, 6, 7

[19] Hang Su, Varun Jampani, Deqing Sun, Subhransu Maji, Evangelos Kalogerakis, Ming-Hsuan Yang, and Jan Kautz. SPLATNet: Sparse lattice networks for point cloud processing. In *CVPR*, 2018. 7

[20] M. Tatarchenko, J. Park, V. Koltun, and Q. Zhou. Tangent convolutions for dense prediction in 3d. In *CVPR*, 2018. 6

[21] Lyne P. Tchapmi, Christopher B. Choy, Iro Armeni, JunYoung Gwak, and Silvio Savarese. Segcloud: Semantic segmentation of 3d point clouds. In *3DV*, 2017. 6

[22] Hugues Thomas, Charles R. Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J. Guibas. Kpconv: Flexible and deformable convolution for point clouds. In *ICCV*, 2019. 1, 6, 7

[23] Chu Wang, Babak Samari, and Kaleem Siddiqi. Local spectral graph convolution for point set feature learning. In *ECCV*, 2018. 7

[24] Lei Wang, Yuchun Huang, Yaolin Hou, Shenman Zhang, and Jie Shan. Graph attention convolution for point cloud semantic segmentation. In *CVPR*, 2019. 6

[25] S. Wang, S. Suo, W. Ma, A. Pokrovsky, and R. Urtasun. Deep parametric continuous convolutional neural networks. In *CVPR*, 2018. 6

[26] 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 Trans. Graph.*, 2019. 2, 4, 7

[27] Wenxuan Wu, Zhongang Qi, and Li Fuxin. Pointconv: Deep convolutional networks on 3d point clouds. In *CVPR*, 2019. 1, 7

[28] Yifan Xu, Tianqi Fan, Mingye Xu, Long Zeng, and Yu Qiao. Spidercnn: Deep learning on point sets with parameterized convolutional filters. In *ECCV*, 2018. 7

[29] 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 Trans. Graph.*, 2016. 5

[30] Li Yi, Hao Su, Xingwen Guo, and Leonidas J. Guibas. Syncspeccnn: Synchronized spectral cnn for 3d shape segmentation. In *CVPR*, 2017. 7