KPConvX: Modernizing Kernel Point Convolution with Kernel Attention

Supplementary Material

Abstract

This supplementary material is divided into the following sections.

- *Appendix A presents the kernel point initialization method.*
- *Appendix B details our network architectures and training parameters.*
- *Appendix C discusses a double shortcut block design.*
- *Appendix D describes how we create stochastic depth in our networks.*
- *Appendix E present full results on S3DIS dataset, and discusses the data preprocessing as full scenes or rooms.*
- *Appendix F gathers additional ablation and parameter studies on S3DIS dataset.*
- *Appendix G present full results on Scannet dataset for our four architectures.*

Our implementation can be found in the following GitHub repository: *https://github.com/apple/ml-kpconvx*

A. Kernel Points Initialization

As explained in the main paper, we use kernel points similarly to [18] and adopt a shell definition of the kernel point positions as proposed in [10]. The original KPConv [18] defined the kernel points regularly on a sphere with an optimization scheme, but did not have the option to have more than one shell. SPConv [10] proposed to initialize multiple kernels with different radii independently, and then merge them together. On the contrary, we choose to initialize all the kernels together, in a unified optimization scheme similar to the one used in KPConv but with constraints enforced on the radius of each shell.

Let s be the number of shells and $[1, N_1, ..., N_s]$ be the number of points per shell. Note that we do not count the center point as a shell as it will always be alone and centered in the sphere. The first step in our kernel initialization method is to compute the shell radii r_j ($j \leq s$). We distribute them regularly along the radius of the kernel sphere r, as shown in Fig. 3 of the main paper:

$$\forall j \in \mathbb{N}, \quad 1 \leq j \leq s, \quad r_j = \frac{2j}{2s+1} r \; . \tag{1}$$

For each kernel point $\widetilde{x}_k$, we apply the same repulsive potential as in [18]:

$$\forall x \in \mathbb{R}^3, \quad E_k^{rep}(x) = \frac{1}{\|x - \widetilde{x}_k\|} \; , \tag{2}$$

but without any attractive potential. Therefore we are trying to minimize

$$E^{tot} = \sum_{k<K} \sum_{l \neq k} E_k^{rep}(\widetilde{x}_l) \; . \tag{3}$$

However, we enforce the constraint that every point can only move on the sphere defined by its shell radius:

$$\forall j \leq s, \quad K_j \leq k < K_{j+1}, \quad \|\widetilde{x}_k\| = r_j \; , \tag{4}$$

where $K_j = 1 + N_1 + ... + N_{j-1}$. We use the same gradient descent algorithm as [18], therefore, the constraint can be applied directly to the gradients, similarly to the other constraints that were already in place, such as the one fixing the first point at the center of the sphere. The whole process is illustrated in Fig. 1

B. Training Parameters and Augmentations

This section details all the training parameters and the augmentation used for our experiments.

S3DIS and ScanNetv2. We share the same parameters for all our architectures. We use the following number of channels for each layer: $[64, 96, 128, 192, 256]$. As explained in the main paper, we start with 64 features and expand with a ratio of $\sqrt{2}$, while still ensuring that the number of channels remains divisible by 16. We train for 180 epochs with 300 steps per epoch. With an effective batch size of 24, we thus see approximately 7200 input point clouds per epoch. We use an initial learning rate of $5e^{-3}$ and reduce the learning rate exponentially at each epoch, at a rate of 0.1 every 60 epochs. We use a weight decay of 0.01 in AdamW, and

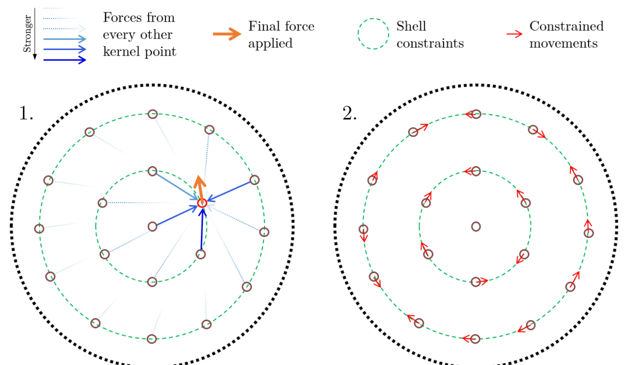

Figure 1. Illustration of our optimization function for kernel point disposition. First, the force applied to each kernel point is computed (1). Then the resulting movements are constrained to the shell spheres (2).

standard values $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $eps = 1e^{-8}$. Concerning augmentations, we follow [16] with some modifications and use (in this order):

- $RandomScale$ ($s_{min} = 0.9, s_{max} = 1.1$)
- $RandomFlip$ ($axis = 0, p = 0.5$)
- $RandomJitter$ ($\sigma = 0.005$)
- $RandomRotate$ ($axis = 2$)
- $ChromaticAutoContrast$ ($p = 0.2$)
- $ChromaticNormalize$ ()
- $RandomDropColor$ ($p = 0.2$)

ScanObjectNN. For ScanObjectNN, we use a 7-channel input feature containing a constant one feature, the point coordinates before augmentation, and the point coordinates after augmentation. The feature channels are $[64, 96, 128, 192, 256]$, following the same $\sqrt{2}$ expansion rule. Our networks are trained for 180 epochs, with enough steps to cover all the input shapes at each epoch, given a batch size of 64. The other training parameters are the same as the ones used for S3DIS. We augment the data with:

- $UnitSphereScale$ ($R = 1.0$)
- $RandomScale$ ($s_{min} = 0.9, s_{max} = 1.1$)
- $RandomFlip$ ($axis = 0, p = 0.5$)
- $RandomRotate$ ($axis = 2$)

C. Discussion on Double Shortcut Blocks

The common practice for deep convolutional networks and transformers is to alternate between local feature extraction (convolution or self-attention) and linear layers. It is considered more efficient to reduce the complexity of the local feature extractor, by making it depthwise and trusting the linear layers to combine features in the channel dimension. The gain in memory consumption usually allows networks to be deeper and reach better performance [12]. Since the original ResNet bottleneck block [4] was designed, it has been the base for the development of newer blocks, with a shortcut connection to solve the vanishing gradient issue.

More recently, the inverted bottleneck design has become more popular [12, 16]. Compared to the bottleneck design that starts with a downsampling MLP, follows with a local extractor, and finishes with an upsampling MLP, the inverted design places the local extractor at the beginning of the block and has two MLPs that upsample and then downsample the features. Although the two designs might seem very different, they are, in fact, nearly the same. If we take a step back and look at the succession of operations, it is always the same: downsampling MLP, local extractor, upsampling MLP, downsampling MLP, etc. The only thing that changes in the series is the placement of the shortcut connection, as shown in Fig. 2. In the first block, the shortcut propagates the high-dimensional features, and in the second, it propagates the low-dimensional features.

During our experiments, we tested with a novel design that combines both types of shortcuts in the same computation graph. We still define blocks in the manner of the inverted bottleneck design, but we add a shortcut between consecutive blocks. This provides an additional path to help propagate the gradients for the high-dimensional features.

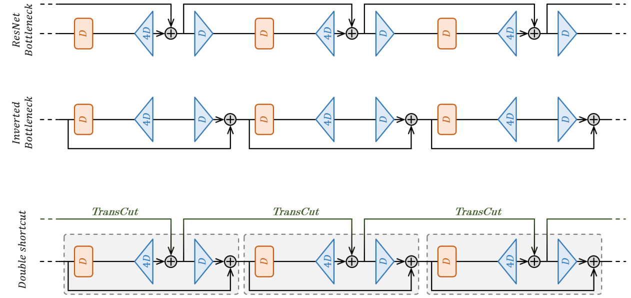

Figure 2. Illustration of our double shortcut block design, compared to ResNet bottleneck and inverted bottleneck blocks. The basic operations are the same, only the features' path changes.

Nevertheless, the improvement brought by the double shortcut design was not significant enough. Furthermore, additional experiments on image datasets would be necessary to validate this design, which is why we decided to keep this idea in the supplementary material.

D. Stochastic Depth and DropPath Implementation

Stochastic depth [5] was proposed as a way to improve the training of deep residual networks by randomly dropping layers. For inference, all the layers are used to harness the full power of the network, allowing for better information and gradient flow. This technique highlights the significant redundancy in deep residual networks. As shown in Fig. 3, the standard way to implement this is to drop (multiply by zeros) all the features from one element (point cloud) of the batch, right before the shortcut addition. This ensures that the features from the previous block are only propagated forward, as if this block did not exist. This operation is commonly referred to as DropPath.

In cases where all the batch elements are of the same length, implementing this technique is straightforward.

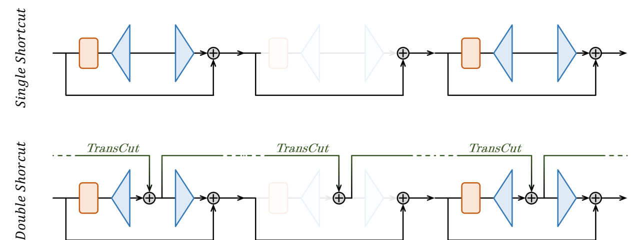

Figure 3. Stochastic depth is easily adapted to double shortcut design, by dropping the same batch elements before both shortcuts.

Table 1. Classwise IoU for S3DIS experiment. Best results are highlighted in **bold** and results within 1% of the best ones are underlined.

Method	Data	mIoU	mAcc	OA	ceiling	floor	wall	beam	column	window	door	table	chair	sofa	bookcase	board	clutter
PointNet [14]	full	41.1	49.0	-	88.8	97.3	69.8	0.1	3.9	46.3	10.8	59.0	52.6	5.9	40.3	26.4	33.2
SegCloud [17]	full	48.9	57.4	-	90.1	96.1	69.9	0.0	18.4	38.4	23.1	70.4	75.9	40.9	58.4	13.0	41.6
PointCNN [9]	full	57.3	63.9	85.9	92.3	98.2	79.4	0.0	17.6	22.8	62.1	74.4	80.6	31.7	66.7	62.1	56.7
SPGraph [7]	full	58.0	66.5	86.4	89.4	96.9	78.1	0.0	42.8	48.9	61.6	84.7	75.4	69.8	52.6	2.1	52.2
SegGCN [8]	full	63.6	70.4	88.2	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
MinkUNet [2]	rooms	65.4	71.7	-	91.8	**98.7**	86.2	0.0	34.1	48.9	62.4	81.6	89.8	47.2	74.9	74.4	58.6
PAConv [20]	rooms	66.6	73.0	-	94.6	98.6	82.4	0.0	26.4	58.0	60.0	89.7	80.4	74.3	69.8	73.5	57.7
KPConv [18]	full	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
PTv1 [21]	rooms	70.4	76.5	90.8	94.0	98.5	**86.3**	0.0	38.0	**63.4**	74.3	89.1	82.4	74.3	**80.2**	76.0	59.3
SPoTr [13]	rooms	70.8	76.4	90.7	-	-	-	-	-	-	-	-	-	-	-	-	-
PointNeXt [16]	rooms	70.5	76.8	90.6	94.2	98.5	84.4	0.0	37.7	59.3	74.0	83.1	91.6	77.4	77.2	78.8	60.6
PointMixer [1]	rooms	71.4	77.4	-	94.2	98.2	86.0	0.0	43.8	62.1	78.5	90.6	82.2	73.9	79.8	78.5	59.4
PTv2 [19]	rooms	71.6	77.9	91.1	-	-	-	-	-	-	-	-	-	-	-	-	-
StratTrans [6]	rooms	72.0	78.1	91.5	**96.2**	**98.7**	85.6	0.0	**46.1**	60.0	76.8	**92.6**	84.5	77.8	75.2	78.1	**64.0**
PointVector [3]	rooms	72.3	78.1	91.0	95.1	98.6	85.1	0.0	41.4	60.8	76.7	84.4	92.1	82.0	77.2	85.1	61.4
PtMetaBase [11]	rooms	72.3	-	91.3	-	-	-	-	-	-	-	-	-	-	-	-	-
KPConvX-L (ours)	rooms	**73.5**	**78.7**	**91.7**	94.9	98.5	86.2	**0.1**	40.4	63.0	**84.1**	84.0	**92.4**	**82.5**	79.0	**86.8**	63.1

However, in our case where the batch elements are stacked along the first dimension, we need to obtain a mask indicating the dropped batch elements. Furthermore, when using our double shortcut design, we need to perform the DropPath operation before both shortcuts, using the same mask for the same batch elements. We provide a custom implementation of the common DropPath operation in our open-source code, enabling future work to utilize it as well.

E. Full Results on S3DIS and Discussion about Data Preprocessing

As mentioned in the paper, we provide the full classwise IoU of our best model on S3DIS Area5 in Tab. 1. We also want to highlight an important factor in the results of this dataset that is often overlooked in previous work: there are two ways to preprocess the data of S3DIS.

On the one hand, it is possible to load entire areas as very large scenes and sample subsets (spheres or cubes) for training [15, 18]. On the other hand, it is also possible to load single rooms as smaller scenes and use them as input, optionally dropping some points [16, 21]. We observe a significant improvement when using the rooms, and we choose this strategy in this paper. For transparency, we indicate which data preprocessing (rooms or full scenes) was used for each method in the state of the art in Tab. 1.

Table 2. Architecture study for KPConvX. Best results are highlighted in **bold** and results within 1% of the best ones are underlined.

	mIoU (5-try avg)	TP	GPU	params
Architecture	mean±std	ins/s	GB	M
$[4, 4, 12, 20, 4] + 1$	72.3 ± 0.6	38.1	6.9	19.7
$[5, 5, 13, 21, 4] + 0$	71.6 ± 0.7	42.8	4.6	19.7
$[4, 4, 12, 20, 4] + 0$	72.1 ± 0.7	48.2	4.6	18.7
$[3, 3, 9, 12, 3] + 1$ *	$\mathbf{72.4} \pm 0.9$	47.7	6.8	13.5
$[4, 4, 4, 12, 4] + 1$	72.2 ± 0.6	46.1	6.8	13.4
$[3, 3, 3, 9, 3] + 1$	72.1 ± 0.6	52.6	6.8	10.4
$[2, 2, 2, 8, 2] + 1$	71.7 ± 0.4	62.3	6.8	8.5
$[2, 2, 2, 6, 2] + 1$	71.5 ± 0.8	64.1	6.8	7.4
$[2, 2, 2, 2, 2] + 1$	70.9 ± 0.5	64.3	6.8	5.2
$[3, 3, 3, 3, 3] + 0$	70.0 ± 0.3	75.7	4.6	6.2
$[2, 2, 2, 2, 2] + 0$	68.8 ± 0.5	88.7	4.6	4.2

Table 3. Architecture study for KPConvD. Best results are highlighted in **bold** and results within 1% of the best ones are underlined.

	mIoU (5-try avg)	TP	GPU	params
Architecture	mean±std	ins/s	GB	M
$[4, 4, 12, 20, 4] + 1$	72.1 ± 0.4	59.5	4.6	11.3
$[5, 5, 13, 21, 4] + 0$	71.8 ± 0.3	57.1	4.6	11.3
$[4, 4, 12, 20, 4] + 0$	71.4 ± 0.5	68.9	4.6	10.8
$[3, 3, 9, 12, 3] + 1$ *	$\mathbf{72.2} \pm 0.7$	64.1	4.6	7.8
$[4, 4, 4, 12, 4] + 1$	71.3 ± 0.8	50.5	4.6	7.8
$[3, 3, 3, 9, 3] + 1$	71.4 ± 0.4	65.5	4.6	6.1
$[2, 2, 2, 8, 2] + 1$	71.3 ± 0.5	75.7	4.6	5.0
$[2, 2, 2, 6, 2] + 1$	71.2 ± 0.4	79.5	4.6	4.4
$[2, 2, 2, 2, 2] + 1$	70.6 ± 0.6	85.7	4.6	3.2
$[3, 3, 3, 3, 3] + 0$	70.2 ± 0.4	94.0	4.6	3.7
$[2, 2, 2, 2, 2] + 0$	69.5 ± 0.3	115.5	4.6	2.6

Table 4. Study of the kernel point shells. Best results are highlighted in **bold** and results within 1% of the best ones are underlined.

	mIoU (5-try avg)	TP	GPU	params
Kernel Point Shells	mean±std	ins/s	GB	M
$[1, 6]$	70.6 ± 0.4	65.9	3.2	9.3
$[1, 12]$	70.9 ± 0.9	61.4	3.6	10.0
$[1, 14]$	71.4 ± 1.1	59.7	3.7	10.2
$[1, 19]$	71.5 ± 0.7	57.4	4.1	10.8
$[1, 28]$	71.8 ± 0.5	52.5	5.0	11.9
$[1, 12, 14]$	71.7 ± 0.7	51.8	4.7	11.6
$[1, 12, 19]$	$\underline{72.3} \pm 0.5$	52.4	5.4	12.2
$[1, 12, 28]$	72.0 ± 0.4	50.0	6.6	13.3
$[1, 14, 19]$	71.9 ± 0.9	51.3	5.7	12.5
$[1, 14, 28]$*	$\mathbf{72.4} \pm 0.6$	47.7	6.8	13.5
$[1, 14, 35]$	$\underline{72.3} \pm 0.5$	46.1	7.7	14.3
$[1, 14, 42]$	$\underline{72.3} \pm 0.7$	42.9	8.6	15.1
$[1, 19, 28]$	72.0 ± 0.5	44.5	7.5	14.1
$[1, 19, 35]$	$\underline{72.2} \pm 0.5$	45.8	8.4	14.9
$[1, 19, 42]$	$\underline{72.4} \pm 0.3$	42.2	9.3	15.7

Table 5. Parameter study of the convolution radius. Best results are highlighted in **bold** and results within 1% of the best ones are underlined.

	mIoU (5-try avg)	TP	GPU	params
Convolution radius	mean±std	ins/s	GB	M
$r = 1.3$	71.7 ± 0.5	47.6	6.8	13.5
$r = 1.4$	$\underline{72.1} \pm 0.5$	47.8	6.8	13.5
$r = 1.5$	71.9 ± 0.4	47.5	6.8	13.5
$r = 1.6$	71.8 ± 0.4	47.9	6.8	13.5
$r = 1.7$	71.9 ± 0.3	47.8	6.8	13.5
$r = 1.8$	71.9 ± 0.7	47.2	6.8	13.5
$r = 1.9$	$\underline{72.1} \pm 0.4$	48.1	6.8	13.5
$r = 2$	$\underline{72.1} \pm 0.5$	47.4	6.8	13.5
$r = 2.1$ *	$\mathbf{72.4} \pm 0.6$	47.7	6.8	13.5
$r = 2.2$	71.9 ± 0.4	47.1	6.8	13.5
$r = 2.3$	72.0 ± 0.3	47.8	6.8	13.5
$r = 2.4$	$\underline{72.2} \pm 0.6$	47.8	6.8	13.5
$r = 2.5$	$\underline{72.0} \pm 0.6$	47.4	6.8	13.5
$r = 2.6$	$\underline{72.1} \pm 0.4$	47.9	6.8	13.5
$r = 2.7$	72.0 ± 0.4	47.9	6.8	13.5
$r = 2.8$	71.9 ± 0.3	47.8	6.8	13.5
$r = 2.9$	71.7 ± 0.5	47.2	6.8	13.5
$r = 3$	72.0 ± 0.5	47.2	6.8	13.5
$r = 3.1$	71.6 ± 0.5	46.7	6.8	13.5

F. Additional Ablation and Parameter Studies

In this section, we provide additional ablation and parameter studies that were not crucial for the paper but are interesting for the reader to gain more insight into the mechanisms of our approach. For this larger study, we provide the average score over 5 attempts.

First, we present full architecture studies for KPConvX and KPConvD, respectively. For the purpose of this experiment, we define our architectures as $[N_1, N_2, N_3, N_4, N_5] + N_{dec}$, where N_i is the number of blocks for layer i and N_{dec} is the number of decoder blocks used (same for each layer). For clarity, in the main paper, we chose to highlight two architectures: small ($[2, 2, 2, 8, 2] + 1$) and large ($[3, 3, 9, 12, 3] + 1$). In Tab. 2 and Tab. 3, we can find these two architectures along with other architecture variants. The architecture sizes vary from the original one used by KPConv ($[2, 2, 2, 2, 2] + 0$) to an extremely large architecture $[4, 4, 12, 20, 4] + 1$. We find that bigger architectures perform better than smaller architectures, but when reaching an extremely large size above KPConvX-L, the performance drops again. We also notice that adding a decoder layer improves the performance even compared to an architecture that has one more encoder layer to compensate.

Then, we showcase a study of the number of kernel points and shells for KPConvX in Tab. 4. The studied kernel point dispositions range from a very simple one-shell $[1, 6]$ disposition where each kernel point is placed in a cardinal direction, to a large two-shell $[1, 19, 42]$ disposition. We observe a general trend where larger kernels improve the results. However, similarly to the architecture study, we observe that the performance stops improving if the number of kernel points increases too much. We thus chose the $[1, 14, 28]$ disposition which led to the best results in this experiment.

Finally, we also study the radius of our convolution kernel in Tab. 5. This parameter does not affect the network's size or efficiency. The effect of changing the convolution radius is that it changes the position of the kernel points in space, scaling the radius of each shell accordingly. The kernel points will be associated with different neighbors depending on their position. If the radius is too small, further neighbors will not have any associated kernel points, and the kernel points placed near the center will be less likely to have any associated neighbors. If the radius is too large, the area covered by each kernel point will be bigger, and the kernel will thus be less descriptive, missing finer details in the input point patterns. Therefore, we find an optimal radius value of 2.1. As a reminder, the radius value is defined relative to the subsampling grid size at every layer. For example, with a 2.1 radius, the first convolution radius on S3DIS data, which is subsampled at 4cm, is 8.4cm.

G. Full Results on Scannet

We also provide the full classwise IoU for our 4 models on the Scannet validation set. As shown in Tab. 6. KPConvX-L is our best network on this dataset as well, followed closely by KPConvD-L. Note that, as opposed to S3DIS, Scannet input point clouds can only be defined as rooms.

Table 6. Classwise IoU for Scannet experiment. Best results are highlighted in **bold** and results within 1% of the best ones are <u>underlined</u>.

Model	mIoU	otherfurniture	bathtub	sink	toilet	shower	refridgerator	curtain	desk	counter	picture	bookshelf	window	door	table	sofa	chair	bed	cabinet	floor	wall
KPConvX-L	**76.3**	61.9	85.6	71.5	93.9	64.3	<u>64.5</u>	77.2	68.8	70.0	**40.5**	81.8	**74.2**	**72.8**	**79.6**	**85.6**	92.2	<u>84.4</u>	**74.0**	**95.9**	**87.2**
KPConvX-S	75.7	**63.9**	88.1	70.6	**95.0**	68.2	**64.6**	77.9	64.1	**71.1**	38.1	83.0	69.0	72.1	76.3	84.1	91.6	82.6	70.2	<u>95.7</u>	**87.2**
KPConvD-L	<u>76.2</u>	61.1	87.9	**72.4**	94.1	**74.0**	61.8	**80.1**	**69.9**	70.2	35.4	**84.7**	69.2	70.8	77.2	84.8	**92.8**	**84.7**	70.5	<u>95.7</u>	<u>86.8</u>
KPConvD-S	75.5	63.4	**89.9**	71.3	92.2	67.2	62.1	78.5	64.5	68.4	39.7	82.6	70.5	72.0	75.3	81.2	91.5	**84.7**	71.3	**95.9**	<u>87.0</u>

References

[1] Jaesung Choe, Chunghyun Park, Francois Rameau, Jaesik Park, and In So Kweon. Pointmixer: Mlp-mixer for point cloud understanding. In *European Conference on Computer Vision*, pages 620–640. Springer, 2022. 3

[2] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3075–3084, 2019. 3

[3] Xin Deng, WenYu Zhang, Qing Ding, and XinMing Zhang. Pointvector: A vector representation in point cloud analysis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9455–9465, 2023. 3

[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*, pages 770–778, 2016. 2

[5] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *European conference on computer vision*, pages 646–661. Springer, 2016. 2

[6] Xin Lai, Jianhui Liu, Li Jiang, Liwei Wang, Hengshuang Zhao, Shu Liu, Xiaojuan Qi, and Jiaya Jia. Stratified Transformer for 3D Point Cloud Segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8500–8509, 2022. 3

[7] Loic Landrieu and Martin Simonovsky. Large-scale point cloud semantic segmentation with superpoint graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4558–4567, 2018. 3

[8] Huan Lei, Naveed Akhtar, and Ajmal Mian. Seggcn: Efficient 3d point cloud segmentation with fuzzy spherical kernel. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11611–11620, 2020. 3

[9] 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. 3

[10] Yuyan Li, Chuanmao Fan, Xu Wang, and Ye Duan. Spnet: Multi-shell kernel convolution for point cloud semantic segmentation. In *International Symposium on Visual Computing*, pages 366–378. Springer, 2021. 1

[11] Haojia Lin, Xiawu Zheng, Lijiang Li, Fei Chao, Shanshan Wang, Yan Wang, Yonghong Tian, and Rongrong Ji. Meta architecture for point cloud analysis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17682–17691, 2023. 3

[12] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11976–11986, 2022. 2

[13] Jinyoung Park, Sanghyeok Lee, Sihyeon Kim, Yunyang Xiong, and Hyunwoo J Kim. Self-positioning point-based transformer for point cloud understanding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21814–21823, 2023. 3

[14] 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. 3

[15] Charles R. 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. 3

[16] Guocheng Qian, Yuchen Li, Houwen Peng, Jinjie Mai, Hasan Abed Al Kader Hammoud, Mohamed Elhoseiny, and Bernard Ghanem. PointNeXt: Revisiting PointNet++ with Improved Training and Scaling Strategies. *arXiv preprint arXiv:2206.04670*, 2022. 2, 3

[17] Lyne Tchapmi, Christopher Choy, Iro Armeni, JunYoung Gwak, and Silvio Savarese. Segcloud: Semantic segmentation of 3d point clouds. In *2017 International Conference on 3D Vision (3DV)*, pages 537–547. IEEE, 2017. 3

[18] 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 *Proceedings of the IEEE International Conference on Computer Vision*, pages 6411–6420, 2019. 1, 3

[19] Xiaoyang Wu, Yixing Lao, Li Jiang, Xihui Liu, and Hengshuang Zhao. Point transformer v2: Grouped vector attention and partition-based pooling. In *NeurIPS*, 2022. 3

[20] Mutian Xu, Runyu Ding, Hengshuang Zhao, and Xiaojuan Qi. Paconv: Position adaptive convolution with dynamic kernel assembling on point clouds. In *Proceedings of*

the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 3173–3182, 2021. 3

[21] Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip HS Torr, and Vladlen Koltun. Point transformer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 16259–16268, 2021. 3