LSK3DNet: Towards Effective and Efficient 3D Perception with Large Sparse Kernels

Tuo Feng[1], Wenguan Wang[2], Fan Ma[2]*, Yi Yang[2]
[1] ReLER, AAII, University of Technology Sydney [2] ReLER, CCAI, Zhejiang University
https://github.com/FengZicai/LSK3DNet

Abstract

Autonomous systems need to process large-scale, sparse, and irregular point clouds with limited compute resources. Consequently, it is essential to develop LiDAR perception methods that are both efficient and effective. Although naively enlarging 3D kernel size can enhance performance, it will also lead to a cubically-increasing overhead. Therefore, it is crucial to develop streamlined 3D large kernel designs that eliminate redundant weights and work effectively with larger kernels. In this paper, we propose an efficient and effective Large Sparse Kernel 3D Neural Network (LSK3DNet) that leverages dynamic pruning to amplify the 3D kernel size. Our method comprises two core components: Spatial-wise Dynamic Sparsity (SDS) and Channel-wise Weight Selection (CWS). SDS dynamically prunes and regrows volumetric weights from the beginning to learn a large sparse 3D kernel. It not only boosts performance but also significantly reduces model size and computational cost. Moreover, CWS selects the most important channels for 3D convolution during training and subsequently prunes the redundant channels to accelerate inference for 3D vision tasks. We demonstrate the effectiveness of LSK3DNet on three benchmark datasets and five tracks compared with classical models and large kernel designs. Notably, LSK3DNet achieves the state-of-the-art performance on SemanticKITTI (i.e.*, 75.6% on single-scan and 63.4% on multi-scan), with roughly 40% model size reduction and 60% computing operations reduction compared to the naive large 3D kernel model.*

1. Introduction

Autonomous systems are equipped with 3D LiDAR sensors and data processing platforms. The LiDAR sensor generates point clouds, which serve as the input for the processing platform. The platform performs perception tasks such as semantic segmentation [1–3] and object detection [4–6], providing a foundation for motion planning and decision-making [7] in autonomous systems. On the one hand, the processing platform is required to handle large-scale, sparse,

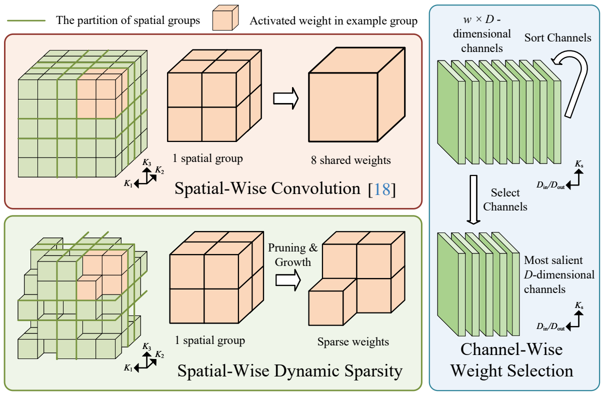
The partition of spatial groups
Activated weight in example group
1 spatial group
8 shared weights
Spatial-Wise Convolution [18]
K_1 K_2 K_3
1 spatial group
Pruning & Growth
Sparse weights
Spatial-Wise Dynamic Sparsity
$w \times D$ -dimensional channels
Sort Channels
Select Channels
Most salient D-dimensional channels
K_s
D_{in}/D_{out}
Channel-Wise Weight Selection

Figure 1. Illustrations on *Spatial-wise Group Convolution* [18], SDS, and CWS. The spatial dimensions K_s (*i.e.*, K_1, K_2, K_3) and channel dimensions (D_{in}/D_{out}) are shown. *Spatial-wise Group Convolution* shares the weights within each spatial group during training, leading to redundant model weights. In contrast, SDS removes non-salient weights and redundancies that are not sensitive to the input in each group, while CWS eliminates redundant weights in a channel-wise manner (Sec. 1).

and irregular point clouds. On the other hand, the platform's computing resources are limited. Therefore, it is crucial to develop effective and efficient LiDAR perception methods. Point-based methods [1, 8, 9] rely on computationally expensive, memory inefficient, and time-consuming point sampling strategy [10]. In contrast, sparse convolution [11–14] has been widely adopted for processing large-scale point clouds up to $160 \times 160 \times 20$ meters [5, 11, 12, 15–17]. Nevertheless, the application of such a technique in autonomous systems has also been hindered by limited computational resources [16]. PV-KD [19] attempts to reduce the model size with knowledge distillation, but its performance is bounded by its teacher model [15]. In opposition, large kernels can enhance performance through the advantages of large receptive fields [18, 20, 21]. However, naive 3D large kernels will significantly increase computational burdens. LargeKernel3D [18] explores large 3D kernels with *Spatial-wise Group Convolution* to circumvent optimization and efficiency problems caused by naive 3D large kernels. However, LargeKernel3D has a lot of redundant model weights during training as it shares the weights in each spatial group. Moreover, the performance of LargeKernel3D [18] drops

*Corresponding author.

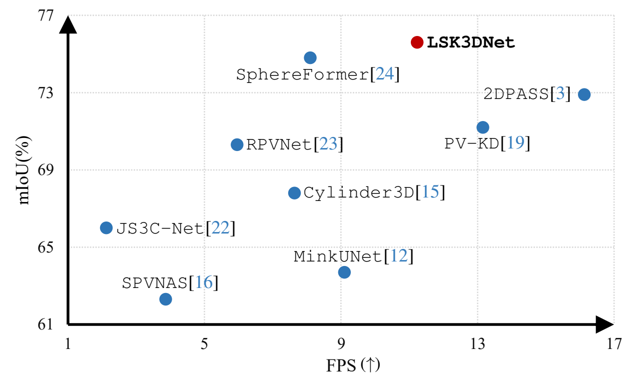

Figure 2. Performance (mIoU) *vs*. Inference Speed (FPS) on SemanticKITTI [25] single-scan challenge (Sec. 1).

when scaling up the kernel size over $7 \times 7 \times 7$.

In this study, we propose a Large Sparse Kernel 3D Neural Network (LSK3DNet), which is an efficient and effective 3D perception model. LSK3DNet incorporates two key components, namely *Spatial-wise Dynamic Sparsity* (SDS) and *Channel-wise Weight Selection* (CWS). These components enhance performance on perception tasks and overcome the challenges of high computational costs. SDS enlarges 3D kernels and reduces model parameters by learning sparse 3D kernels from scratch. CWS increases the model width and learns channel-wise importance during training, resulting in accelerated inference by pruning redundant channels [26]. As shown in Fig. 1, SDS can prune 3D kernels to remove redundant weights, while CWS can eliminate redundant channels in an online manner. Our SDS and CWS complement each other following a guideline of *"using spatial sparse groups, expanding width without more parameters"*, in contrast to *"using sparse groups, expanding more"* [21].

Specifically, SDS implements a remove-and-regrow process within each spatial group and preserves the sparsity in each group during dynamic training. This stands in stark contrast to the 2D approach [21], as it achieves dynamic sparsity in depth-wise groups. SDS allows to scale up the receptive fields with large kernel sizes, easily reaching or surpassing $9 \times 9 \times 9$, thereby achieving higher performance. In addition, CWS selectively identifies salient channels during training. It speeds up inference on the 3D vision tasks by pruning non-salient channel parameters. In this way, LSK3DNet can benefit from expanded width to achieve enhanced performance, but maintain the original model size during inference. Reducing the models' complexity (*e.g.*, model size) is a big benefit and a straightforward way to make them deployable on resource-constrained devices [27]. We conduct experiments on three benchmark datasets and five tracks. Our method achieves better performance compared to state-of-the-art methods [15, 23, 24] on SemanticKITTI [25] but with faster inference speed (Fig. 2). Moreover, LSK3DNet outperforms the prior 3D large kernel method of LargeKernel3D [18] on ScanNet v2 [28] and KITTI [29].

In a nutshell, our **contributions** are as follows:

- We propose LSK3DNet to enhance performance in perception tasks and mitigate high computational costs, surpassing state-of-the-art models while reducing model size by 40% and computing operations by 60%.
- We propose SDS to scale up 3D kernels. It learns large sparse kernels by dynamically pruning and regrowing weights from scratch, thereby reducing model size and computational operations.
- We develop CWS to improve performance by expanding the width. CWS assesses the importance of channels during training and speeds up inference by pruning redundant channel-wise parameters.

2. Related Work

Large-Kernel 3D Models. In 3D vision, research on 3D large kernel is very limited. LargeKernel3D [18] has demonstrated that sizeable kernels can be successfully employed and bring positive results for 3D networks. *Spatial-wise Group Convolution* [18] enables to achieve a kernel size of $7 \times 7 \times 7$. However, it shares the weights within each spatial group during training, leading to redundant model weights (See Fig. 1). Moreover, the performance of LargeKernel3D drops when scaling up the kernel size over $7 \times 7 \times 7$.

Large-Kernel 2D CNN Models. In the 2010s, various large-kernel settings were investigated. LR-Net [30], Inceptions [31], and GCNs [32] explored 2D large kernels of 7×7, 11×11, and 15×15 respectively. Due to the widespread adoption [33, 34] of VGG [35], research into large kernels was largely overlooked in favour of multiple smaller kernels (1×1 or 3×3) to obtain a larger receptive field [36–38] during the past decade. Recently, certain studies have reintroduced large kernels in CNNs. RepLKNet [20] examines the effects of large kernels in CNNs, and for the first time it is able to increase kernel size to 31×31. It achieves comparable results to those of Swin Transformer [39]. Following *"use sparse groups, expand more"*, SLaK [21] has achieved an impressive kernel size of 51×51. SLaK has to strike a trade-off between sparsity and model width since increased model width leads to an increase in model size, a phenomenon that becomes even more pronounced in 3D vision. However, CWS effectively decouples the two objectives of improved performance and reduced model size, enabling expanded width without increasing the model size.

3D Backbones. Generally speaking, there are three mainstreams of 3D backbones: *i) Point-based* methods [1, 8, 9, 40–55] approximate a permutation-invariant set function, and directly learn features from raw point clouds. Researchers have observed an unsatisfactory outcome for large-scale environments [56]. *ii) Projection-based* methods transform unstructured points into regular 2D images and thus employ CNNs [57–62]. However, the projection could leave out critical geometric details and cause unavoidable information loss. *iii) Voxel-based* methods have used sparse convolution [11, 12] to design more powerful networks for various 3D tasks [15, 63]. Sparse convolution facilitates convolution operations only on non-void voxels. Recently, *fusion-based* methods [3, 16, 23] have become upsurge. The point-based high-resolution branch mitigates the performance degradation, which is caused by aggressive downsampling [16, 64] (*i.e.*, regular sparse convolution) of the voxel branch.

Dynamic Sparse Training (DST). DST can train sparse neural networks from scratch, resulting in both a speedy training and prediction procedure. During training, DST [65–73] alters the location of non-zero weights with per predefined rules, thereby creating a sparse representation and cutting down the number of calculations. This paradigm commences without prior knowledge and simultaneously refines the non-zero locations and weights. The attractive aspect of DST is that it is sparse from the beginning, resulting in lower FLOPs and memory consumption for training and inference compared to a dense model. Generally, the pruning process [74] can be completed either by threshold-based pruning [65, 66, 70, 75] or by magnitude-based pruning [67, 76]. In addition, new weights are regrown with randomness growing [66, 70, 75], momentum growing [67], and gradient-based growing [68, 69, 76, 77]. Here, we efficiently integrate dynamic sparsity with 3D kernels in large kernel 3D networks. This design satisfies the anticipation for a larger parameter space, as a wide-ranging exploration of the parameter space is of significance to DST [71, 73, 78].

3. Methodology

In this section, we first formulate the problem in Sec. 3.1, then move on to a concise overview of submanifold sparse convolution in Sec. 3.2. Afterwards, the details of *Spatial-wise Dynamic Sparsity* (SDS) and *Channel-wise Weight Selection* (CWS) are described in Sec. 3.3 and 3.4. Lastly, network architecture is outlined in Sec. 3.5.

3.1. Problem Formulation

For 3D perception tasks, suppose that the input is a set of unordered point $\mathcal{X} = \{x_i \in \mathbb{R}^{D_{in}} : i = 1, 2, \dots, N\}$, and D_{in} is the dimensionality of the input features. $\mathcal{L}$ is a set of object/point labels, which varies according to different datasets. For segmentation task, LSK3DNet assigns each point x_i with a predicted label y_i, resulting in a point-wise prediction set $\mathcal{Y} = \{y_i : i = 1, 2, \dots, N\}$. In voxel-based methods, point clouds are converted to voxel representations. We denote the input sparse tensor with $X_s^{in} = [C, F]$. Coordinate matrix $C \in \mathbb{N}_+^{N\times 3}$ includes 3-dimensional discrete coordinates, and feature matrix $F \in \mathbb{R}^{N\times D_{in}}$ has D_{in}-dimensional features.

3.2. Review of Submanifold Sparse Convolution

Regular sparse convolution has widespread use in U-Net type 3D networks [11, 12]. However, it significantly reduces the sparsity level of point data [64], obfuscates feature distinctions [64], and leads to low-resolution and indistinguishable small objects [16]. In our LSK3DNet, we discard regular sparse convolution and exclusively employ submanifold sparse convolution for feature extraction.

In a 3-dimensional space $C \in \mathbb{N}_+^{N\times 3}$, the D_{in}-dimensional input feature at $c \in C$ can be denoted as $x_c^{in} \in \mathbb{R}^{D_{in}}$, and the 3D kernel is represented by $\boldsymbol{W} \in \mathbb{R}^{K^3 \times D_{\text{out}} \times D_{\text{in}}}$. We divide the kernel into K^3 spatial weights, each with a size of $D_{out} \times D_{in}$, and denote the divided weights as $\boldsymbol{W}_i$. The submanifold sparse convolution is formulated as follows:

$$x_c^{out} = \sum_{i \in \mathcal{V}^D(c,C)} \boldsymbol{W}_i \, x_{c+i}^{in}, \quad c \in C, \tag{1}$$

where x_c^{out} represents the current voxel on which the submanifold sparse convolution is applied; $\boldsymbol{W}_i$ corresponds to x_{c+i}^{in}, which represents the i-th adjacent input voxel ($[C_{c+i}, F_{c+i}]$); $\mathcal{V}^D$ is a set of offsets that define the shape of a kernel, and $\mathcal{V}^D(c, C) = \{i | c + i \in C, i \in \mathcal{V}^D\}$ is a set of offsets from the current center c that exist in C [12]. It should be noted that the input coordinates and output coordinates are equivalent; in other words, they share the same C. Due to this restriction, insufficient information flow exists while differentiating the distinct spatial characteristics [64]. Large receptive fields could be a potential solution to this problem.

3.3. Spatial-wise Dynamic Sparsity (SDS)

SDS incorporates two essential components: *Sparse Kernel Initialization* and *Sparse Weight Pruning and Growth*. SDS is applied to the 3D large kernel layers, while other layers are kept dense. When initialized with *Sparse Kernel Initialization*, all spatial groups of sparse tensors have the same level of sparsity s, where sparsity refers to the fraction of zeros in sparse kernels. We adjust the sparse position within spatial groups of 3D large kernel layers with an adaptation frequency f_a. During training, at regular intervals determined by f_a, the adjustable weights in the kernels are modified through a two-phase procedure, which includes *sparse weight pruning* and *sparse weight growth*. We maintain a constant number of non-zero parameters (*i.e.*, sparsity s) throughout the training process.

Sparse Kernel Initialization. Prior research [79] has explored the distribution of sparsity parameters across multiple layers. The proportion and adjustment of sparse weights are regulated by the layer-wise sparsity ratio, which determines how the weights are modified to seek more efficient sparse structures during training. However, this method is not suitable for *Spatial-wise Group Convolution*, since 3D large kernels are divided into multiple spatial groups. To ensure that each group has non-zero weights, we execute Erdős-Rényi (ER) based sparsity ratios [66, 69, 80] in each spatial group. Dense weights $\boldsymbol{W}_D$ are firstly initialized in a standard way (*i.e.*, kaiming uniform initialization [81]). The binary Mask

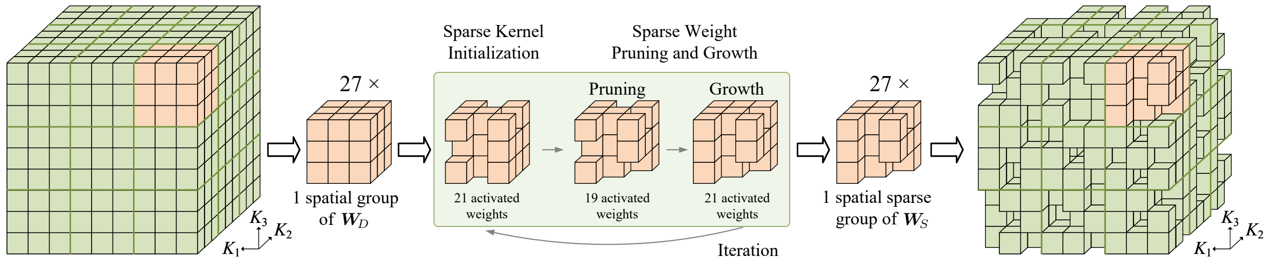

Figure 3. **Spatial-wise Dynamic Sparsity.** The utilization of SDS enables us to create and train sparse kernel 3D neural networks from the beginning. The sparse weights in each spatial group are firstly initialized by *Sparse Kernel Initialization* (Eq. (2)), and then regularly altered by discarding the least significant connections and introducing new ones (Eq. (3)). The sparse kernels are steadily improved by this dynamic process, which leads to a more thorough collection of local features. Note that different spatial sparse group has different sparse distribution. Here we take a sparsity of 22% for example (Sec. 3.3 and Alg. 1).

$\boldsymbol{M}$ is then applied to these dense weights; $\boldsymbol{M}$ is generated using an ER-based method with a sparsity of s. The zero-weight number in $\boldsymbol{M}$ is scaled by $1-\frac{D_{in}+D_{out}+K_1^g+K_2^g+K_3^g}{D_{in}\times D_{out}\times K_1^g\times K_2^g\times K_3^g}$, where K_1^g, K_2^g, and K_3^g denote the spatial group sizes. Therefore, the sparse weights $\boldsymbol{W}_S$ are initialized as follows:

$$\boldsymbol{W}_S \leftarrow \boldsymbol{W}_D \odot \boldsymbol{M}, \tag{2}$$

where $\odot$ represents the Hadamard product.

To design 3D networks with sparse groups, following the recipe of *"using spatial sparse groups, expanding width without more parameters"*, we have chosen *Spatial-wise Group Convolution* rather than *Depth-wise Group Convolution*. Our experiments have also shown that *Depth-wise Group Convolution* is unsuitable for large 3D kernels. This aligns with the findings in LargeKernel3D [18]. This experiment is provided in the supplementary materials.

Sparse Weight Pruning and Growth. Previous research [21] in 2D convolution has illustrated that dynamic sparsity can effectively enlarge kernel sizes and enhance scalability. But *Depth-wise Group Convolution* is unsuitable for large 3D kernels [18]. Therefore, we adapt sparse weights in each spatial group dynamically to accommodate the data. Specifically, a certain rate of connections (*i.e*., prune rate p) with the lowest magnitude are eliminated, and then we generate an equal number of new random connections. The regrow weights are randomly placed at zero positions within each spatial group. This is achieved by modifying the mask $\boldsymbol{M}$ (See Alg. 1). The positions of the eliminated connections and the newly generated connections are denoted as $\boldsymbol{E}$ and $\boldsymbol{G}$, respectively. The formula for pruning and regrowth can be expressed as follows:

$$\boldsymbol{W}_S \leftarrow \boldsymbol{W}_S \odot (\boldsymbol{M}-\boldsymbol{E}),\ \boldsymbol{W}_S \leftarrow \boldsymbol{W}_S \odot (\boldsymbol{M}+\boldsymbol{G}). \tag{3}$$

By doing so, the sparsity of each spatial group is kept steady. Moreover, the weights can be adjusted dynamically, allowing for improved local characteristics. Please refer to Fig. 3 for details. Once training is complete, mask $\boldsymbol{M}$ in spatial groups is also recorded. Unlike previous methods [21], we employ the *Spatial-wise Group Convolution* and partition dynamic sparse processes into separate spatial groups. Pruning and growth are conducted independently within each group without interfering with each other.

We can assume that weights are changing factors over time. Then, removing the least important weights is akin to the selection phase in natural evolution. Alternatively, the random addition of new weights is analogous to the alteration stage of evolutionary selection [66]. This phenomenon is also analogous to a biological process in the brain during sleep, known as synaptic shrinking. Researchers found that the weakest neural link in the brain weakens during slumber, while the vital neural connections remain how they are before. This shows that one of the main roles of sleeping is to reset the overall synaptic strength [82, 83].

3.4. Channel-wise Weight Selection (CWS)

Despite SLaK [21] employs DST to enlarge the 2D kernel size, it mitigates the performance degradation caused by sparsity in the way of expanding model width. It strives to strike a balance between sparsity and width. However, expanding width leads to the issue of increased computational burden. Therefore, SLaK [21] faces the dilemma that larger sparsity and smaller model size cannot be achieved simultaneously. Instead of naively increasing network width, we propose CWS to decouple sparsity and width. It selectively chooses the most salient channels and obtains improved performance while keeping the original model size during inference [26].

CWS operates in an online mode with a sorting frequency of f_s. A single weight sorting cycle involves multiple iterations of *Sparse Weight Pruning and Growth*, *i.e*., each cycle of weight sorting is a multiple of the adaptation rate f_a. The width factor w determines the extent to which the number of channels is augmented, *i.e*., extending D-dimension to $w\times D$-dimension. Once the expanded model has been created, we consider the D-dimensional small model to be embedded within the $w\times D$-dimensional augmented model [26]. When the number of training iterations reaches the sorting condition, the channels are sorted based on their L1 value (*i.e*., SortChannels), prompting the model to focus on the most relevant channels. During validation, we select the most important D-dimensional channels from $w\times D$-dimensional

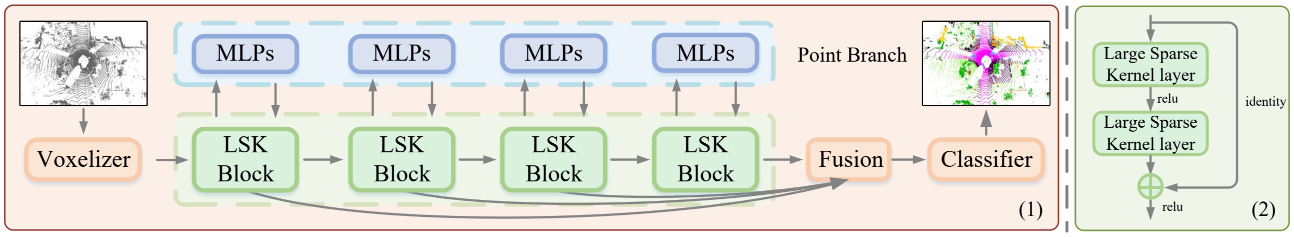

Figure 4. **LSK3DNet and LSK Block.** (1) LSK3DNet: Point clouds are fed into voxelizer for voxel-wise features. Then we extract features with LSK Block and Point Branch. The final prediction is a point-wise output. Here MLPs is the standard Multi-Layer Perceptrons. (2) Large Sparse Kernel Block (LSK Block) (Sec.3.5).

channels [26] (*i.e.*, SelectChannels). The pseudo-code for the entire algorithm is provided in Alg. 1. This operation effectively achieves higher performance while maintaining the parameters within the expected size. This is especially important in deployments where memory usage and computational efficiency are critical factors.

3.5. Network Architecture

Segmentation Baseline. U-Net type 3D networks (such as SparseConvNet [11] and MinkUNet [12]) employ aggressive downsampling (*i.e.*, regular sparse convolution) to increase the receptive field but at the cost of reduced resolution. However, with a low resolution, several points or tiny objects could be combined into one single grid and become indistinguishable [16]. So SPVCNN [16] is equipped with a high-resolution point-based branch. Subsequently, 2DPASS [3] further modifies multi-representation branches by omitting the regular sparse convolution. This is because it dilates all sparse features and blurs valuable information, increasing the burden for following layers [64]. Only submanifold sparse convolution is employed for feature extraction in Modified SPVCNN [3]. By limiting the output feature positions to the input positions, submanifold sparse convolution is able to circumvent the computation burden. We use Modified SPVCNN as a baseline for our segmentation network. This 3D network is compact yet powerful and can generate high-resolution representations from sparse point clouds in large-scale scenes. However, one challenge we face is that the restricted area of submanifold sparse convolution limits the information flow and makes it difficult to distinguish different spatial characteristics of the scene. To address this challenge, we introduce LSK Block, which stands for Large Sparse Kernel Block. This block increases the kernel sizes of submanifold sparse convolution and expands the receptive field to facilitate information flow. LSK Block has a standard residual structure that adds the output of identity mapping to that of two stacked large kernel convolutions (Fig.4(2)). Our network does not need the parallel convolutional branch with dilatation convolution [18]. The details of our segmentation network architecture are listed below:

- **Segmentation Network on SemanticKITTI [25].** LSK3DNet (See Fig.4(1)) divides the entire scene into voxels, each with a size of 5 cm. It has four scales of {2, 4, 8, 16}.

Algorithm 1 Pseudo-code of SDS and CWS

```
1:  set adaptation frequency $f_a$, sparsity $s$;
2:  set sorting frequency $f_s$ and width factor $w$;
3:  expand model width form $D$ dimension to $w \times D$ dimension;
4:  initialize dense model $\boldsymbol{W}_D^{w\times D}$;
5:  initialize sparse layers $\boldsymbol{W}_S^{w\times D}$;
6:  for each training iteration $i$ do
7:      $\boldsymbol{W}_S^{wxD} \leftarrow$NormalTraining($\boldsymbol{W}_S^{wxD}$);
8:      if ($i\%f_a$) equals to 0 then
9:          $\boldsymbol{W}_S^{wxD} \leftarrow \boldsymbol{W}_S^{wxD} \odot (\boldsymbol{M} - \boldsymbol{E})$;
10:         $\boldsymbol{W}_S^{wxD} \leftarrow \boldsymbol{W}_S^{wxD} \odot (\boldsymbol{M} + \boldsymbol{G})$;
11:     end if
12:     if ($i\%f_s$) equals to 0 then
13:         $\boldsymbol{W}_S^{wxD} \leftarrow$SortChannels($\boldsymbol{W}_S^{wxD}$);
14:     end if
15:     if validation then
16:         $\boldsymbol{W}_S^{wxD} \leftarrow$SortChannels($\boldsymbol{W}_S^{wxD}$);
17:         $\boldsymbol{W}_S^{D} \leftarrow$SelectChannels($\boldsymbol{W}_S^{wxD}$);
18:         NormalValidation($\boldsymbol{W}_S^{D}$);
19:     end if
20: end for
```

The hiden size D of the entire network is 64. We deploy LSK Blocks in *SparseBasicBlock*[1]. Following [3, 15, 85], we employ the weighted cross-entropy loss to optimize point accuracy and utilize the lovasz-softmax [86] loss to maximize the intersection-over-union.

- **Segmentation Network on ScanNet v2 [28].** This segmentation network has the same architecture as that on SemanticKITTI [25]. The entire scene is split with a voxel size of 2 cm, with scales of {2, 4, 8, 16, 16}. The hidden size D is 128. Following [87], we take the weighted cross-entropy loss as the objective function.

Detection Network on KITTI. We take Voxel R-CNN [5] as a baseline. Specifically, we retain the backbone of Voxel R-CNN and substitute plain sparse convolutional block with LSK Block in the first three stages. Other settings remain the same as [18]. The input spatial shape is {1600, 1408, 41}, and the channels of stages are {16, 16, 32, 64, 64}.

4. Experiment

4.1. Setups and Implementations

Dataset. To evaluate our method, we perform experiments

[1]`https://github.com/yanx27/2DPASS`

Table 1. Results of LSK3DNet on SemanticKITTI [25] single-scan *test* (Sec. 4.2). Regarding input data format, P denotes points, V represents voxelizations, R signifies range images, and 2DPASS incorporates additional 2D data. "ms/s" means "milliseconds per scene".

Methods	Input	**mIoU**	road	sidewalk	parking	other-ground	building	car	truck	bicycle	motorcycle	other-vehicle	vegetation	trunk	terrain	person	bicyclist	motorcyclist	fence	pole	traffic sign	speed (ms/s)
PointNet++ [41]	P	20.1	72.0	41.8	18.7	5.6	62.3	53.7	0.9	1.9	0.2	0.2	46.5	13.8	30.0	0.9	1.0	0.0	16.9	6.0	8.9	5900
TangentConv [57]	R	40.9	83.9	63.9	33.4	15.4	83.4	90.8	15.2	2.7	16.5	12.1	79.5	49.3	58.1	23.0	28.4	8.1	49.0	35.8	28.5	3000
PolarNet [62]	R	54.3	90.8	74.4	61.7	21.7	90.0	93.8	22.9	40.3	30.1	28.5	84.0	65.5	67.8	43.2	40.2	5.6	61.3	51.8	57.5	62
RandLA-Net [10]	P	55.9	90.5	74.0	61.8	24.5	89.7	94.2	43.9	29.8	32.2	39.1	83.8	63.6	68.6	48.4	47.4	9.4	60.4	51.0	50.7	880
SqueezeSegV3 [61]	R	55.9	91.7	74.8	63.4	26.4	89.0	92.5	29.6	38.7	36.5	33.0	82.0	58.7	65.4	45.6	46.2	20.1	59.4	49.6	58.9	238
KPConv [42]	P	58.8	90.3	72.7	61.3	31.5	90.5	95.0	33.4	30.2	42.5	44.3	84.8	69.2	69.1	61.5	61.6	11.8	64.2	56.4	47.4	-
JS3C-Net [22]	V	66.0	88.9	72.1	61.9	31.9	92.5	95.8	54.3	59.3	52.9	46.0	84.5	69.8	67.9	69.5	65.4	39.9	70.8	60.7	68.7	471
SPVNAS [16]	PV	67.0	90.2	75.4	67.6	21.8	91.6	97.2	56.6	50.6	50.4	58.0	86.1	73.4	71.0	67.4	67.1	50.3	66.9	64.3	67.3	259
Cylinder3D [15]	V	68.9	92.2	77.0	65.0	32.3	90.7	97.1	50.8	67.6	63.8	58.5	85.6	72.5	69.8	73.7	69.2	48.0	66.5	62.4	66.2	131
RPVNet [23]	RPV	70.3	93.4	80.7	70.3	33.3	93.5	97.6	44.2	68.4	68.7	61.1	86.5	75.1	71.7	75.9	74.4	43.4	72.1	64.8	61.4	168
(AF)^2-S3Net [84]	V	70.8	92.0	76.2	66.8	45.8	92.5	94.3	40.2	63.0	81.4	40.0	78.6	68.0	63.1	76.4	81.7	77.7	69.6	64.0	73.3	-
PV-KD [19]	V	71.2	91.8	77.5	70.9	41.0	92.4	97.0	53.5	67.9	69.3	60.2	86.5	73.8	71.9	75.1	73.5	50.5	69.4	64.9	65.8	76
2DPASS [3]	PV	72.9	89.7	74.7	67.4	40.0	93.5	97.0	61.1	63.6	63.4	61.5	86.2	73.9	71.0	77.9	81.3	74.1	72.9	65.0	70.4	62
SphereFormer [24]	V	74.8	91.8	78.2	69.7	41.3	93.8	97.5	59.6	70.1	70.5	67.7	86.7	75.1	72.4	79.0	80.4	75.3	72.8	66.8	72.9	123
LSK3DNet	PV	**75.6**	92.2	78.9	70.2	41.8	92.7	97.3	61.0	71.4	75.6	64.2	86.4	72.7	71.9	81.2	80.6	85.2	72.0	67.0	74.6	89

Table 2. Results of LSK3DNet on SemanticKITTI [25] multi-scan *test* set. The arrow below classes indicate moving classes (Sec. 4.2).

Method	Input	**mIoU**	**Acc**	car $\uparrow$	truck $\uparrow$	other-vehicle $\uparrow$	person $\uparrow$	bicyclist $\uparrow$	motorcyclist $\uparrow$
LatticeNet [89]	P	45.2	89.3	54.8	3.5	0.6	49.9	44.6	64.3
TLSeg [90]	R	47.0	89.6	68.2	2.1	12.4	40.4	42.8	12.9
KPConv [42]	P	51.2	89.3	69.4	5.8	4.7	67.5	67.4	47.2
Cylinder3D [15]	V	52.5	91.0	74.9	0.0	0.1	65.7	68.3	11.9
(AF)^2-S3Net [84]	V	56.9	88.1	65.3	5.6	3.9	67.6	66.4	59.6
2DPASS [3]	PV	62.4	91.4	82.1	16.1	3.8	80.3	71.2	73.1
LSK3DNet	PV	**63.4**	**92.2**	84.4	7.2	40.9	77.4	69.9	72.1

on three benchmark datasets and five tracks:

- **SemanticKITTI** [25] has 43,551 traffic point cloud scenes with fine annotations, split into 19,130/4,071/20,350 scenes for `train`/`val`/`test`. The dataset has 28 semantic classes, but only 19 classes are used for single-scan track and 25 classes for multi-scan track.
- **ScanNet v2** [28] contains 1,201/312/100 indoor scenes for `train`, `val`, and `test` splits, respectively. There are 20 semantic categories for ScanNet20 track and 200 categories for ScanNet200 track.
- **KITTI** [29] has 7,481/7,518 samples for `train` and `test`. We follow the frequently used `train`/`val` split [88] to divide the training samples into *train* split (3,712 samples) and *val* split (3,769 samples).

Training and Testing Details. We utilize AdamW optimizer with OneCycleLR scheduler starting with a learning rate of 5e-3. Data augmentation is also employed, such as random flipping, scaling, rotation around the gravity axis, spatial translation. We apply instance CutMix [23] and Test Time Augmentation (TTA) [3] to the SemanticKITTI *test* benchmark, and enhance the model with extra training epochs.

Metrics. We employ mean class intersection over union (mIoU) and overall accuracy (Acc) metrics as the evaluation

Table 3. Results of LSK3DNet and other tate-of-the-art methods on ScanNet v2 [28] (Sec. 4.2). S-Net stands for ScanNet.

Method	Input	S-Net20	S-Net20	S-Net200
		Val	*Test*	*Val*
PointNet++ [41]	P	53.5	55.7	-
3DMV [91]	P	-	48.4	-
PanopticFusion [92]	P	-	52.9	-
PointCNN [93]	P	-	45.8	-
PointConv [94]	P	61.0	66.6	-
JointPointBased [95]	P	69.2	63.4	-
PointASNL [96]	P	63.5	66.6	-
SegGCN [97]	P	-	58.9	-
RandLA-Net [85]	P	-	64.5	-
KPConv [42]	P	69.2	68.6	-
JSENet [98]	P	-	69.9	-
FusionNet [99]	P	-	68.8	-
Point Transformer [8]	P	70.6	-	-
Fast Point Transformer [100]	P	72.1	-	-
Stratified Transformer [9]	P	74.3	73.7	-
Point Transformer v2 [87]	P	75.4	75.2	31.9
SparseConvNet [11, 87]	V	69.3	72.5	28.8
MinkUNet [12]	V	72.2	73.6	-
LargeKernel3D [18]	V	73.5	73.9	-
LSK3DNet	PV	**75.7**	**75.5**	**33.1**

criterion for 3D semantic segmentation tasks, as outlined in [25]. What is more, we calculate Average Precision (AP) by recalling 11 positions for 3D object detection.

4.2. 3D Semantic Segmentation

Results on SemanticKITTI. We test LSK3DNet on both single-scan and multi-scan tracks [25]. Tab. 1 presents the quantitative results of SemanticKITTI single-scan track. LSK3DNet outperforms 2DPASS [3] in terms of both mIoU and IoU scores for most categories. Moreover, SDS reduces the model size of naive 3D kernels and CWS keeps the model size within the desired size. So, LSK3DNet has a faster running speed than most prior methods, but it's performance significantly benefits from the large kernels. Tab. 2 shows

Table 4. Results of 3D object detection methods on the car class of KITTI *val* set [29] (Sec. 4.3).

Method	Input	3D AP (IoU=0.7)		
		Easy	Moderate	Hard
VoxelNet [101]	V	81.97	65.46	62.85
PointPillars [102]	R	86.62	76.06	68.91
SECOND [63]	V	88.61	78.62	77.22
Point R-CNN [103]	P	88.88	78.63	77.38
Part-A^2 [104]	V	89.47	79.47	78.54
PV-RCNN [4]	PV	89.35	83.69	78.70
Focals Conv [64]	V	89.52	84.93	79.18
Voxel R-CNN [5]	V	89.41	84.52	78.93
LargeKernel3D [18]	V	89.52	85.07	79.32
LSK3DNet	V	**90.16**	**85.61**	**79.53**

the results of SemanticKITTI multi-scan track. The mIoU and Acc are computed over all 25 classes. Due to page limitations, we only report the per-class IoUs for moving categories. Under this challenging setting, LSK3DNet surpasses 2DPASS [3] in both mIoU and Acc. Notably, LSK3DNet achieves state-of-the-art performance on both single-scan and multi-scan tracks in SemanticKITTI. Surpassing over the SOTA 2D-3D method 2DPASS is more valuable, which utilizes both 2D and 3D data while our LSK3DNet takes as input only the 3D data. Visualization results are in the supplementary materials.

Results on ScanNet V2. We compare LSK3DNet with previous state-of-the-art methods on ScanNet v2 [28], a large-scale dataset for 3D indoor scene segmentation. ScanNet v2 has two tracks: ScanNet20 and ScanNet200, which use 20 and 200 semantic classes respectively. Tab. 3 presents the quantitative results of our model and other methods on both tracks. LSK3DNet achieves higher performance than previous methods in both tracks. Our performance is superior to transformer-based methods [8, 9, 87, 100], including Point Transformer v2. Moreover, LSK3DNet has a clear advantage over LargeKernel3D [18], improving the mIoU by 2.2% and 1.6% on the ScanNet v2 *val* and *test* set, respectively.

4.3. 3D Object Detection

We have also evaluated the detection performance of LSK3DNet on the *val* split for car category, as shown in Tab. 4. We report the average precision metric for a 3D bounding box with 11 recall positions (Recall 11). Based on Voxel R-CNN [5], we showcase the effectiveness of the LSK Block by comparing it with LargeKernel3D [18], a previous 3D large-kernel method. LSK3DNet achieves better results in three difficulty levels, compared with [5, 18]. Visualization comparison can be found in the supplementary material.

4.4. Ablation Studies

We first explore the effect of kernel size on performance, and then the overall architecture design. Next, we explain the hyperparameter choices of SDS and CWS. All ablation experiments are performed on the *val* set of SemancKITTI.

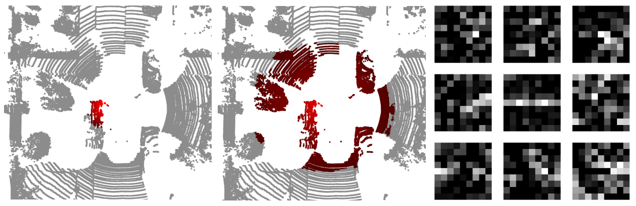

Figure 5. Effective Receptive Fields (ERFs) of Baseline and LSK3DNet. LSK3DNet has a larger ERF size. Additionally, we provide visualization of learned sparse kernels, where all weight values have taken the absolute value and normalization operations. The black areas indicate positions with weight values of zero.

Kernel Size. We use Modified SPVCNN as the baseline and then explore 3D kernel sizes under two settings: naive dense large kernel and our LSK3DNet. In the former setting, the dense 3D kernel is straightforwardly expanded. In the latter one, LSK3DNet is trained simultaneously with SDS and CWS, meaning that our LSK3DNet can learn a large sparse kernel model from the beginning.

The baseline does not use aggressive downsampling which is common in most U-type 3D networks [11, 12]. However, submanifold sparse convolution with small kernels may lose important information flow, especially for the spatially disconnected features. We solve this problem by enlarging the size of the 3D convolution kernel to obtain a large receptive field. This agrees with the performance improvement in Tab. 5, when the kernel size increases. In contrast to this observation, LargeKernel3D [18] shows the opposite trends; we empirically attribute this to the fact that LargeKernel3D is based on U-type 3D networks (Sec. 3.5). For a better understanding of Effective Receptive Fields (ERFs) size, Fig. 5 illustrates the comparison between the Baseline and LSK3DNet. Compared to the baseline, the high-contribution points of LSK3DNet are distributed over a larger input range, indicating a larger ERF. Additionally, to enhance comprehension of learned kernels, we offer kernel visualization in Fig. 5, providing insights into sparse training.

Compared to the "naive dense large kernel" of each kernel size, LSK3DNet exhibits superior performance in its corresponding size. LSK3DNet not only leverages SDS to expand its receptive field but also starts from scratch to learn 3D sparse kernels. Concurrently, CWS widens the network during training while pruning redundant channels to accelerate inference. Furthermore, SDS and CWS address the issues of overparameterization and overfitting that arise from simply increasing the kernel's size and expanding the model's width. The best performance of LSK3DNet is achieved at the 9×9×9 size, and the following experiments are conducted with $9\times9\times9$ size. Moreover, due to the extra dimension in 3D kernels compared to 2D kernels, naively enlarging the 3D kernel size will lead to a cubically-increasing overhead. Our LSK3DNet can effectively reduce 19.1M parameters and approximately 60% of computing operations when compared

Table 5. Segmentation performance of Modified SPVCNN with various large kernel settings. "Dense" refers to directly enlarging the kernel size. "Ours" refers to training with SDS and CWS (Sec. 4.4). The unit for "Param" is in million (M), "FLOPs" is in billion (G), and "Speed" is measured in milliseconds per scene.

	Kernel Size	mIoU	Param	FLOPs	Speed ↓
Dense ($D=64$)	$3,3,3$	65.2	1.9	78.4	36
	$5,5,5$	65.6	8.4	127.8	57
	$7,7,7$	66.2	22.6	381.2	65
	$9,9,9$	67.5	47.9	1916.3	93
	$11,11,11$	66.6	87.4	3192.0	134
Dense ($1.8\times D$)	$5,5,5$	66.2	27.0	444.0	84
	$7,7,7$	66.5	73.1	1551.2	123
	$9,9,9$	67.2	154.7	3030.0	177
	$11,11,11$	66.3	282.1	7139.2	250
Ours ($1.8\times D$)	$5,5,5$	66.8	5.1	203.8	46
	$7,7,7$	68.1	13.6	249.4	63
	$\mathbf{9,9,9}$	**70.2**	28.8	763.6	89
	$11,11,11$	70.1	52.5	1847.4	116

Table 6. Overall architecture design with $9\times9\times9$ size. Here, we report Param, FLOPs, and Speed for the inference model (Sec. 4.4).

Method	mIoU	Param	FLOPs	Speed ↓
Dense ($1.0\times D$)	67.5	47.9	1916.3	93
Dense ($1.8\times D$)	67.2	154.7	3030.0	177
SDS ($1.8\times D$)	69.3	93.0	2201.4	173
CWS ($1.8\times D$)	69.1	47.9	1916.3	93
LSK3DNet ($\mathbf{1.8\times D}$)	**70.2**	**28.8**	**763.6**	**89**

to the naive large kernel network. This aspect significantly enhances the value of SDS and CWS for 3D kernels.

Overall Architecture Design. In Tab. 6, Dense ($1.8\times D$) does not yield improvement due to overparameterization and overfitting. Although SDS maintains a model width of $1.8\times D$ during inference, it achieves improved performance due to sparse training and the large receptive field. CWS effectively addresses overfitting by selecting salient channels, thereby enhancing performance. By combining SDS and CWS, LSK3DNet can achieve the best performance while significantly reducing the model size and computational cost.

Spatial-wise Dynamic Sparsity. To control the degree of sparse weight adaptation in LSK3DNet, we have adjusted three hyperparameters: the adaptation frequency f_a, sparsity rate s, and prune rate p (Sec. 3.3). Adaptation frequency f_a indicates how often we update the sparse weights during training. Sparsity rate s specifies how sparse the 3D large kernels are. Prune rate p shows the fraction of updated weights in one adaptation. We have conducted experiments with different values of these hyperparameters and report the results in Tab. 7 and Tab. 8. Based on our empirical findings, we have chosen $f_a=2000$, $s=0.4$, and $p=0.3$ as the optimal settings for LSK3DNet.

Channel-wise Weight Selection. Another aspect of our method that we investigated is the CWS. This technique involves two hyperparameters: the sorting frequency f_s and the width factor w (Sec. 3.4). The former determines how frequently we sort and select channels for large kernels during training. It is expressed as a multiple of f_a, the adaptation frequency, meaning that we perform channel sorting after a certain number of weight adaptation cycles. When $f_s=6\times f_a$, the optimal performance is achieved. The latter controls the times of network width compared to the target model. We find that increasing the width factor up to $1.8\times$ improves the model's performance, but beyond that point, there is no significant gain. Therefore, we opt for a width of $1.8\times$ to avoid more extra compute resources that a wider width would require during training.

Table 7. Training and inference speed analysis, speed is measured in milliseconds per scene. The kernels have $9\times9\times9$ size, "T" and "I" represent training and inference speed, respectively (Sec. 4.4).

Sparsity	Dense ($1.0\times D$)			SDS ($1.8\times D$)			LSK3DNet ($1.8\times D$)		
s	mIoU	T ↓	I ↓	mIoU	T ↓	I ↓	mIoU	T ↓	I ↓
0	67.5	451	93	-	-	-	-	-	-
0.1	-	-	-	68.5	489	176	69.1	491	92
0.2	-	-	-	68.9	478	176	69.7	482	91
0.4	-	-	-	69.3	463	173	**70.2**	467	89
0.6	-	-	-	69.0	450	171	68.8	454	87
0.8	-	-	-	68.7	439	171	67.2	443	87

Table 8. Ablation studies of *Spatial-wise Dynamic Sparsity* on adaptation frequency f_a and prune rate p (Sec. 4.4).

Adaptation f_a	mIoU
100	68.8
1000	69.6
2000	**70.2**
3000	70.0
4000	69.8

Pruning p	mIoU
0.10	69.9
0.20	70.1
0.30	**70.2**
0.50	70.0
0.70	69.7

Table 9. Ablation studies of *Channel-wise Weight Selection* on sorting frequency f_s and width factor w (Sec. 4.4).

Sorting f_s	mIoU
$1\times f_a$	68.2
$2\times f_a$	69.1
$4\times f_a$	69.8
$\mathbf{6\times f_a}$	**70.2**
$10\times f_a$	69.9

Width w	mIoU
$1.1\times$	68.4
$1.5\times$	69.6
$\mathbf{1.8\times}$	**70.2**
$2.1\times$	69.9
$2.5\times$	70.1

5. Conclusion

We propose *Spatial-wise Dynamic Sparsity* to scale up 3D kernels beyond $9\times9\times9$, which prunes the volumetric weight and reduces the parameter size of large kernel layers. Our LSK3DNet can benefit from a large receptive field without increasing the computational cost compared to a naive 3D large kernel. *Channel-wise Weight Selection* expands the model width during training, and then sorts and selects important channels during validation to get a model of the expected size. In this way, we achieve *"using spatial sparse groups, expanding width without more parameters"*. We evaluate our method on the SemanticKITTI and achieve state-of-the-art performance. Our LSK3DNet also surpasses previous 3D large kernel methods on ScanNet v2 and KITTI.

Acknowledgments: This work was supported by a China Scholarship Council (CSC) scholarship.

References

[1] Haoran Zhou, Yidan Feng, Mingsheng Fang, Mingqiang Wei, Jing Qin, and Tong Lu. Adaptive graph convolution for point cloud analysis. In *ICCV*, 2021. 1, 2

[2] Tuo Feng, Wenguan Wang, Xiaohan Wang, Yi Yang, and Qinghua Zheng. Clustering based point cloud representation learning for 3d analysis. In *ICCV*, 2023.

[3] Xu Yan, Jiantao Gao, Chaoda Zheng, Chao Zheng, Ruimao Zhang, Shuguang Cui, and Zhen Li. 2dpass: 2d priors assisted semantic segmentation on lidar point clouds. In *ECCV*, 2022. 1, 2, 3, 5, 6, 7

[4] Shaoshuai Shi, Chaoxu Guo, Li Jiang, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. Pv-rcnn: Point-voxel feature set abstraction for 3d object detection. In *CVPR*, 2020. 1, 7

[5] Jiajun Deng, Shaoshuai Shi, Peiwei Li, Wengang Zhou, Yanyong Zhang, and Houqiang Li. Voxel r-cnn: Towards high performance voxel-based 3d object detection. In *AAAI*, 2021. 1, 5, 7

[6] Junbo Yin, Jianbing Shen, Runnan Chen, Wei Li, Ruigang Yang, Pascal Frossard, and Wenguan Wang. Is-fusion: Instance-scene collaborative fusion for multimodal 3d object detection. In *CVPR*, 2024. 1

[7] Tuo Feng, Ruijie Quan, Xiaohan Wang, Wenguan Wang, and Yi Yang. Interpretable3d: An ad-hoc interpretable classifier for 3d point clouds. In *AAAI*, 2024. 1

[8] Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip Torr, and Vladlen Koltun. Point transformer. In *ICCV*, 2021. 1, 2, 6, 7

[9] 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 *CVPR*, 2022. 1, 2, 6, 7

[10] Qingyong Hu, Bo Yang, Linhai Xie, Stefano Rosa, Yulan Guo, Zhihua Wang, Niki Trigoni, and Andrew Markham. Randla-net: Efficient semantic segmentation of large-scale point clouds. In *CVPR*, 2020. 1, 6

[11] Benjamin Graham, Martin Engelcke, and Laurens van der Maaten. 3d semantic segmentation with submanifold sparse convolutional networks. In *CVPR*, 2018. 1, 3, 5, 6, 7

[12] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *CVPR*, 2019. 1, 2, 3, 5, 6, 7

[13] Junbo Yin, Jin Fang, Dingfu Zhou, Liangjun Zhang, Cheng-Zhong Xu, Jianbing Shen, and Wenguan Wang. Semi-supervised 3d object detection with proficient teachers. In *ECCV*, 2022.

[14] Junbo Yin, Dingfu Zhou, Liangjun Zhang, Jin Fang, Cheng-Zhong Xu, Jianbing Shen, and Wenguan Wang. Proposal-contrast: Unsupervised pre-training for lidar-based 3d object detection. In *ECCV*, 2022. 1

[15] Xinge Zhu, Hui Zhou, Tai Wang, Fangzhou Hong, Yuexin Ma, Wei Li, Hongsheng Li, and Dahua Lin. Cylindrical and asymmetrical 3d convolution networks for lidar segmentation. In *CVPR*, 2021. 1, 2, 3, 5, 6

[16] Haotian Tang, Zhijian Liu, Shengyu Zhao, Yujun Lin, Ji Lin, Hanrui Wang, and Song Han. Searching efficient 3d architectures with sparse point-voxel convolution. In *ECCV*, 2020. 1, 2, 3, 5, 6

[17] Tianwei Yin, Xingyi Zhou, and Philipp Krahenbuhl. Center-based 3d object detection and tracking. In *CVPR*, 2021. 1

[18] Yukang Chen, Jianhui Liu, Xiaojuan Qi, Xiangyu Zhang, Jian Sun, and Jiaya Jia. Scaling up kernels in 3d cnns. *CVPR*, 2023. 1, 2, 4, 5, 6, 7

[19] Yuenan Hou, Xinge Zhu, Yuexin Ma, Chen Change Loy, and Yikang Li. Point-to-voxel knowledge distillation for lidar semantic segmentation. In *CVPR*, 2022. 1, 2, 6

[20] Xiaohan Ding, Xiangyu Zhang, Jungong Han, and Guiguang Ding. Scaling up your kernels to 31x31: Revisiting large kernel design in cnns. In *CVPR*, 2022. 1, 2

[21] Shiwei Liu, Tianlong Chen, Xiaohan Chen, Xuxi Chen, Qiao Xiao, Boqian Wu, Mykola Pechenizkiy, Decebal Mocanu, and Zhangyang Wang. More convnets in the 2020s: Scaling up kernels beyond 51x51 using sparsity. *arXiv preprint arXiv:2207.03620*, 2022. 1, 2, 4

[22] Xu Yan, Jiantao Gao, Jie Li, Ruimao Zhang, Zhen Li, Rui Huang, and Shuguang Cui. Sparse single sweep lidar point cloud segmentation via learning contextual shape priors from scene completion. In *AAAI*, 2021. 2, 6

[23] Jianyun Xu, Ruixiang Zhang, Jian Dou, Yushi Zhu, Jie Sun, and Shiliang Pu. Rpvnet: A deep and efficient range-point-voxel fusion network for lidar point cloud segmentation. In *ICCV*, 2021. 2, 3, 6

[24] Xin Lai, Yukang Chen, Fanbin Lu, Jianhui Liu, and Jiaya Jia. Spherical transformer for lidar-based 3d recognition. In *CVPR*, 2023. 2, 6

[25] Jens Behley, Martin Garbade, Andres Milioto, Jan Quenzel, Sven Behnke, Cyrill Stachniss, and Jurgen Gall. Semantickitti: A dataset for semantic scene understanding of lidar sequences. In *ICCV*, 2019. 2, 5, 6

[26] Han Cai, Chuang Gan, Ji Lin, and Song Han. Network augmentation for tiny deep learning. In *ICLR*, 2022. 2, 4, 5

[27] Vidya Kamath and A Renuka. Deep learning based object detection for resource constrained devices-systematic review, future trends and challenges ahead. *Neurocomputing*, 2023. 2

[28] Angela Dai, Angel X Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *CVPR*, 2017. 2, 5, 6, 7

[29] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013. 2, 6, 7

[30] Han Hu, Zheng Zhang, Zhenda Xie, and Stephen Lin. Local relation networks for image recognition. In *ICCV*, 2019. 2

[31] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, 2017. 2

[32] Chao Peng, Xiangyu Zhang, Gang Yu, Guiming Luo, and Jian Sun. Large kernel matters–improve semantic segmentation by global convolutional network. In *CVPR*, 2017. 2

[33] Ruijie Quan, Xuanyi Dong, Yu Wu, Linchao Zhu, and Yi Yang. Auto-reid: Searching for a part-aware convnet for person re-identification. In *ICCV*, 2019. 2

[34] Wenguan Wang, Guolei Sun, and Luc Van Gool. Looking beyond single images for weakly supervised semantic segmentation learning. *IEEE TPAMI*, 2022. 2

[35] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 2

[36] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 2

[37] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[38] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *CVPR*, 2017. 2

[39] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *ICCV*, 2021. 2

[40] 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

[41] 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. 6

[42] 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. 6

[43] Mutian Xu, Runyu Ding, Hengshuang Zhao, and Xiaojuan Qi. Paconv: Position adaptive convolution with dynamic kernel assembling on point clouds. In *CVPR*, 2021.

[44] Siqi Fan, Qiulei Dong, Fenghua Zhu, Yisheng Lv, Peijun Ye, and Fei-Yue Wang. Scf-net: Learning spatial contextual features for large-scale point cloud segmentation. In *CVPR*, 2021.

[45] Haoxi Ran, Wei Zhuo, Jun Liu, and Li Lu. Learning inner-group relations on point clouds. In *ICCV*, 2021.

[46] Tiange Xiang, Chaoyi Zhang, Yang Song, Jianhui Yu, and Weidong Cai. Walk in the cloud: Learning curves for point clouds shape analysis. In *ICCV*, 2021.

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

[48] Qiangeng Xu, Xudong Sun, Cho-Ying Wu, Panqu Wang, and Ulrich Neumann. Grid-gcn for fast and scalable point cloud learning. In *CVPR*, 2020.

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

[50] Yanchao Lian, Tuo Feng, and Jinliu Zhou. A dense pointnet++ architecture for 3d point cloud semantic segmentation. In *IGARSS*, 2019.

[51] Guocheng Qian, Abdulellah Abualshour, Guohao Li, Ali Thabet, and Bernard Ghanem. Pu-gcn: Point cloud upsampling using graph convolutional networks. In *CVPR*, 2021.

[52] Hehe Fan, Xin Yu, Yi Yang, and Mohan Kankanhalli. Deep hierarchical representation of point cloud videos via spatio-temporal decomposition. *IEEE TPAMI*, 44(12):9918–9930, 2021.

[53] Hehe Fan, Yi Yang, and Mohan Kankanhalli. Point spatio-temporal transformer networks for point cloud video modeling. *IEEE TPAMI*, 45(2):2181–2192, 2022.

[54] Hehe Fan, Xin Yu, Yuhang Ding, Yi Yang, and Mohan Kankanhalli. Pstnet: Point spatio-temporal convolution on point cloud sequences. In *ICLR*, 2020.

[55] Hehe Fan, Yi Yang, and Mohan Kankanhalli. Point 4d transformer networks for spatio-temporal modeling in point cloud videos. In *CVPR*, 2021. 2

[56] Qingyong Hu, Bo Yang, Sheikh Khalid, Wen Xiao, Niki Trigoni, and Andrew Markham. Towards semantic segmentation of urban-scale 3d point clouds: A dataset, benchmarks and challenges. In *CVPR*, 2021. 2

[57] Maxim Tatarchenko, Jaesik Park, Vladlen Koltun, and Qian-Yi Zhou. Tangent convolutions for dense prediction in 3d. In *CVPR*, 2018. 2, 6

[58] Bichen Wu, Alvin Wan, Xiangyu Yue, and Kurt Keutzer. Squeezeseg: Convolutional neural nets with recurrent crf for real-time road-object segmentation from 3d lidar point cloud. In *ICRA*, 2018.

[59] Bichen Wu, Xuanyu Zhou, Sicheng Zhao, Xiangyu Yue, and Kurt Keutzer. Squeezesegv2: Improved model structure and unsupervised domain adaptation for road-object segmentation from a lidar point cloud. In *ICRA*, 2019.

[60] Tiago Cortinhal, George Tzelepis, and Eren Erdal Aksoy. Salsanext: fast, uncertainty-aware semantic segmentation of lidar point clouds for autonomous driving. *arXiv preprint arXiv:2003.03653*, 2020.

[61] Chenfeng Xu, Bichen Wu, Zining Wang, Wei Zhan, Peter Vajda, Kurt Keutzer, and Masayoshi Tomizuka. Squeezesegv3: Spatially-adaptive convolution for efficient point-cloud segmentation. In *ECCV*, 2020. 6

[62] Yang Zhang, Zixiang Zhou, Philip David, Xiangyu Yue, Zerong Xi, Boqing Gong, and Hassan Foroosh. Polarnet: An improved grid representation for online lidar point clouds semantic segmentation. In *CVPR*, 2020. 2, 6

[63] Yan Yan, Yuxing Mao, and Bo Li. Second: Sparsely embedded convolutional detection. *Sensors*, 18(10):3337, 2018. 3, 7

[64] Yukang Chen, Yanwei Li, Xiangyu Zhang, Jian Sun, and Jiaya Jia. Focal sparse convolutional networks for 3d object detection. In *CVPR*, 2022. 3, 5, 7

[65] Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert Legenstein. Deep rewiring: Training very sparse deep networks. *arXiv preprint arXiv:1711.05136*, 2017. 3

[66] Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*, 9(1):2383, 2018. 3, 4

[67] Tim Dettmers and Luke Zettlemoyer. Sparse networks from scratch: Faster training without losing performance. *arXiv preprint arXiv:1907.04840*, 2019. 3

[68] Shiwei Liu, Decebal Constantin Mocanu, Amarsagar Reddy Ramapuram Matavalam, Yulong Pei, and Mykola Pechenizkiy. Sparse evolutionary deep learning with over one million artificial neurons on commodity hardware. *Neural Computing and Applications*, 33:2589–2604, 2021. 3

[69] Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In *ICML*, 2020. 3

[70] Hesham Mostafa and Xin Wang. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In *ICML*, 2019. 3

[71] Siddhant Jayakumar, Razvan Pascanu, Jack Rae, Simon Osindero, and Erich Elsen. Top-kast: Top-k always sparse training. In *NeurIPS*, 2020. 3

[72] Tianlong Chen, Yu Cheng, Zhe Gan, Lu Yuan, Lei Zhang, and Zhangyang Wang. Chasing sparsity in vision transformers: An end-to-end exploration. In *NeurIPS*, 2021.

[73] Shiwei Liu, Decebal Constantin Mocanu, Yulong Pei, and Mykola Pechenizkiy. Selfish sparse rnn training. In *ICML*, 2021. 3

[74] Song Han, Jeff Pool, Sharan Narang, Huizi Mao, Enhao Gong, Shijian Tang, Erich Elsen, Peter Vajda, Manohar Paluri, John Tran, et al. Dsd: Dense-sparse-dense training for deep neural networks. In *ICLR*, 2016. 3

[75] Decebal Constantin Mocanu et al. *Network computations in artificial intelligence*. Technische Universiteit Eindhoven, 2017. 3

[76] Xiaoliang Dai, Hongxu Yin, and Niraj K Jha. Nest: A neural network synthesis tool based on a grow-and-prune paradigm. *IEEE Transactions on Computers*, 68(10):1487–1497, 2019. 3

[77] Xiaoliang Dai, Hongxu Yin, and Niraj K Jha. Grow and prune compact, fast, and accurate lstms. *IEEE Transactions on Computers*, 69(3):441–452, 2019. 3

[78] Md Aamir Raihan and Tor Aamodt. Sparse weight activation training. In *NeurIPS*, 2020. 3

[79] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*, 2018. 3

[80] Shiwei Liu, Tianlong Chen, Xiaohan Chen, Li Shen, Decebal Constantin Mocanu, Zhangyang Wang, and Mykola Pechenizkiy. The unreasonable effectiveness of random pruning: Return of the most naive baseline for sparse training. In *ICLR*, 2022. 3

[81] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015. 3

[82] Graham H Diering, Raja S Nirujogi, Richard H Roth, Paul F Worley, Akhilesh Pandey, and Richard L Huganir. Homer1a drives homeostatic scaling-down of excitatory synapses during sleep. *Science*, 355(6324):511–515, 2017. 4

[83] Luisa De Vivo, Michele Bellesi, William Marshall, Eric A Bushong, Mark H Ellisman, Giulio Tononi, and Chiara Cirelli. Ultrastructural evidence for synaptic scaling across the wake/sleep cycle. *Science*, 355(6324):507–510, 2017. 4

[84] Ran Cheng, Ryan Razani, Ehsan Taghavi, Enxu Li, and Bingbing Liu. (af)2-s3net: Attentive feature fusion with adaptive feature selection for sparse semantic segmentation network. In *CVPR*, 2021. 6

[85] Qingyong Hu, Bo Yang, Linhai Xie, Stefano Rosa, Yulan Guo, Zhihua Wang, Niki Trigoni, and Andrew Markham. Randla-net: Efficient semantic segmentation of large-scale point clouds. In *CVPR*, 2020. 5, 6

[86] Maxim Berman, Amal Rannen Triki, and Matthew B Blaschko. The lovász-softmax loss: A tractable surrogate for the optimization of the intersection-over-union measure in neural networks. In *CVPR*, 2018. 5

[87] Xiaoyang Wu, Yixing Lao, Li Jiang, Xihui Liu, and Hengshuang Zhao. Point transformer v2: Grouped vector attention and partition-based pooling. In *NeurIPS*, 2022. 5, 6, 7

[88] Qinghao Meng, Wenguan Wang, Tianfei Zhou, Jianbing Shen, Luc Van Gool, and Dengxin Dai. Weakly supervised 3d object detection from lidar point cloud. In *ECCV*, 2020. 6

[89] Radu Alexandru Rosu, Peer Schütt, Jan Quenzel, and Sven Behnke. Latticenet: Fast point cloud segmentation using permutohedral lattices. *arXiv preprint arXiv:1912.05905*, 2019. 6

[90] Fabian Duerr, Mario Pfaller, Hendrik Weigel, and Jürgen Beyerer. Lidar-based recurrent 3d semantic segmentation with temporal memory alignment. In *3DV*, 2020. 6

[91] Angela Dai and Matthias Nießner. 3dmv: Joint 3d-multi-view prediction for 3d semantic scene segmentation. In *ECCV*, 2018. 6

[92] Gaku Narita, Takashi Seno, Tomoya Ishikawa, and Yohsuke Kaji. Panopticfusion: Online volumetric semantic mapping at the level of stuff and things. In *IROS*, 2019. 6

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

[94] Wenxuan Wu, Zhongang Qi, and Li Fuxin. Pointconv: Deep convolutional networks on 3d point clouds. In *CVPR*, 2019. 6

[95] Hung-Yueh Chiang, Yen-Liang Lin, Yueh-Cheng Liu, and Winston H Hsu. A unified point-based framework for 3d segmentation. In *3DV*, 2019. 6

[96] Xu Yan, Chaoda Zheng, Zhen Li, Sheng Wang, and Shuguang Cui. Pointasnl: Robust point clouds processing using nonlocal neural networks with adaptive sampling. In *CVPR*, 2020. 6

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

[98] Zeyu Hu, Mingmin Zhen, Xuyang Bai, Hongbo Fu, and Chiew-lan Tai. Jsenet: Joint semantic segmentation and edge detection network for 3d point clouds. In *ECCV*, 2020. 6

[99] Feihu Zhang, Jin Fang, Benjamin Wah, and Philip Torr. Deep fusionnet for point cloud semantic segmentation. In *ECCV*, 2020. 6

[100] Chunghyun Park, Yoonwoo Jeong, Minsu Cho, and Jaesik Park. Fast point transformer. In *CVPR*, 2022. 6, 7

[101] Charles R Qi, Or Litany, Kaiming He, and Leonidas J Guibas. Deep hough voting for 3d object detection in point clouds. In *ICCV*, 2019. 7

[102] Alex H Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *CVPR*, 2019. 7

[103] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. Pointrcnn: 3d object proposal generation and detection from point cloud. In *CVPR*, 2019. 7

[104] Shaoshuai Shi, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. From points to parts: 3d object detection from point cloud with part-aware and part-aggregation network. *IEEE TPAMI*, 43(8):2647–2664, 2020. 7