SqueezeSegV3: Spatially-Adaptive Convolution for Efficient Point-Cloud Segmentation

Chenfeng Xu[1⋆], Bichen Wu[2], Zining Wang[3], Wei Zhan[3], Peter Vajda[2], Kurt Keutzer[3], and Masayoshi Tomizuka[3]

[1] Huazhong University of Science and Technology
[2] Facebook Inc
[3] University of California, Berkeley
xuchenfeng@hust.edu.cn,
{wbc, vajdap}@fb.com,
{wangzining,wzhan,keutzer}@berkeley.edu, tomizuka@me.berkeley.edu

Abstract. LiDAR point-cloud segmentation is an important problem for many applications. For large-scale point cloud segmentation, the *de facto* method is to project a 3D point cloud to get a 2D LiDAR image and use convolutions to process it. Despite the similarity between regular RGB and LiDAR images, we discover that the feature distribution of LiDAR images changes drastically at different image locations. Using standard convolutions to process such LiDAR images is problematic, as convolution filters pick up local features that are only active in specific regions in the image. As a result, the capacity of the network is under-utilized and the segmentation performance decreases. To fix this, we propose Spatially-Adaptive Convolution (SAC) to adopt different filters for different locations according to the input image. SAC can be computed efficiently since it can be implemented as a series of element-wise multiplications, im2col, and standard convolution. It is a general framework such that several previous methods can be seen as special cases of SAC. Using SAC, we build SqueezeSegV3 for LiDAR point-cloud segmentation and outperform all previous published methods by at least 3.7% mIoU on the SemanticKITTI benchmark with comparable inference speed. Code and pretrained model are avalibale at https://github.com/chenfengxu714/SqueezeSegV3.

Keywords: Point-Cloud Segmentation, Spatially-Adaptive Convolution

1 Introduction

LiDAR sensors are widely used in many applications [59], especially autonomous driving [9,56,1]. For level 4 & 5 autonomous vehicles, most of the solutions rely on LiDAR to obtain a point-cloud representation of the environment. LiDAR point clouds can be used in many ways to understand the environment, such as

⋆ This work was conducted during Chenfeng Xu's visit to University of California, Berkeley.

2D/3D object detection [65,3,41,34], multi-modal fusion [64,17], simultaneous localization and mapping [4,2] and point-cloud segmentation [56,58,35]. This paper is focused on point-cloud segmentation. This task takes a point-cloud as input and aims to assign each point a label corresponding to its object category. For autonomous driving, point-cloud segmentation can be used to recognize objects such as pedestrians and cars, identify drivable areas, detecting lanes, and so on. More applications of point-cloud segmentation are discussed in [59].

Recent work on point-cloud segmentation is mainly divided into two categories, focusing on small-scale or large-scale point-clouds. For small-scale problems, ranging from object parsing to indoor scene understanding, most of the recent methods are based on PointNet [35,36]. Although PointNet-based methods have achieved competitive performance in many 3D tasks, they have limited processing speed, especially for large-scale point clouds. For outdoor scenes and applications such as autonomous driving, typical LiDAR sensors, such as Velodyne HDL-64E LiDAR, can scan about $64 \times 3000 = 192,000$ points for each frame, covering an area of $160 \times 160 \times 20$ meters. Processing point clouds at such scale efficiently or even in real time is far beyond the capability of PointNet-based methods. Hence, much of the recent work follows the method based on spherical projection proposed by Wu *et al.* [56,58]. Instead of processing 3D points directly, these methods first transform a 3D LiDAR point cloud into a 2D LiDAR image and use 2D ConvNets to segment the point cloud, as shown in Figure 1. In this paper, we follow this method based on spherical projection.

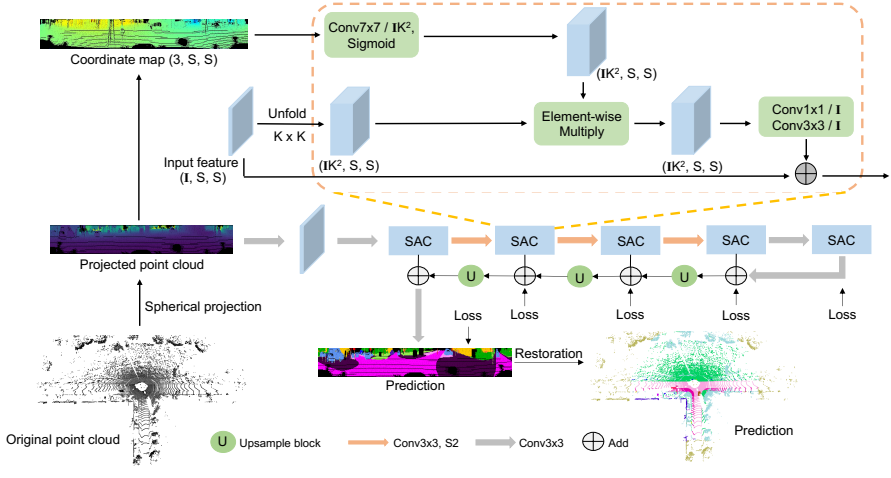

Fig. 1: The framework of SqueezeSegV3. A LiDAR point cloud is projected to generate a LiDAR image, which is then processed by spatially adaptive convolutions (SAC). The network outputs a point-wise prediction that can be restored to label the 3D point cloud. Other variants of SAC can be found in Figure 4.

To transform a 3D point-cloud into a 2D grid representation, each point in the 3D space is projected to a spherical surface. The projection angles of each

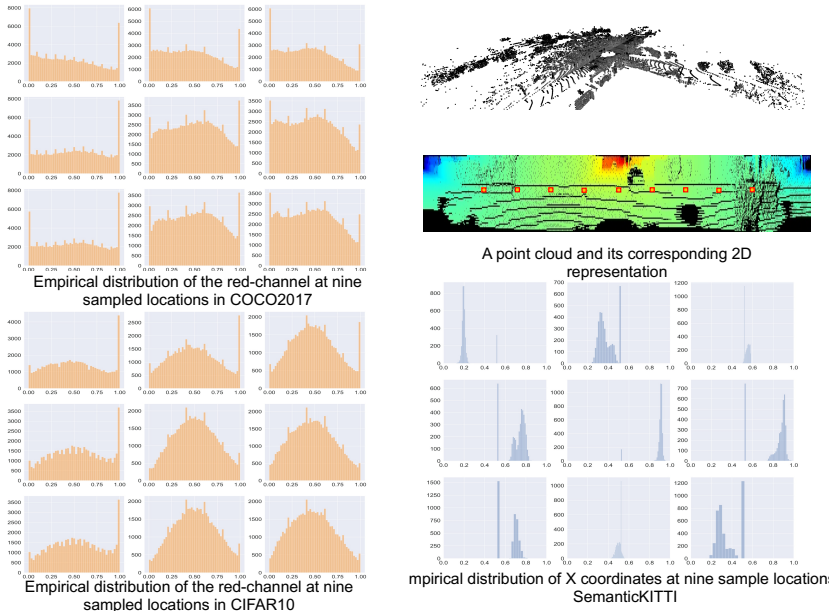

Fig. 2: Pixel-wise feature distribution at nine sampled locations from COCO2017 [25], CIFAR10 [21] and SemanticKITTI [1]. The left shows the distribution of the red channel across all images in COCO2017 and CIFAR10. The right shows the distribution of the X coordinates across all LiDAR images in SemanticKITTI.

point are quantized and used to denote the location of the pixel. Each point's original 3D coordinates are treated as features. Such representations of LiDAR are very similar to RGB images, therefore, it seems straightforward to adopt 2D convolution to process "LiDAR images". This pipeline is illustrated in Figure 1.

However, we discovered that an important difference exists between LiDAR images and regular images. For a regular image, the feature distribution is largely invariant to spatial locations, as visualized in Figure 2. For a LiDAR image, its features are converted by spherical projection, which introduces very strong spatial priors. As a result, the feature distribution of LiDAR images varies drastically at different locations, as illustrated in Figure 2 and Figure 3 (top). When we train a ConvNet to process LiDAR images, convolution filters may fit local features and become only active in some regions and are not used in other parts, as confirmed in Figure 3 (bottom). As a result, the capacity of the model is under-utilized, leading to decreased performance in point-cloud segmentation.

To tackle this problem, we propose Spatially-Adaptive Convolution (SAC), as shown in Figure 1. SAC is designed to be spatially-adaptive and content-aware. Based on the input, it adapts its filters to process different parts of the image. To ensure efficiency, we factorize the adaptive filter into a product of a static convolution weight and an attention map. The attention map is computed by a

one-layer convolution, whose output at each pixel location is used to adapt the static weight. By carefully scheduling the computation, SAC can be implemented as a series of widely supported and optimized operations including element-wise multiplication, im2col, and reshaping, which ensures the efficiency of SAC.

SAC is formulated as a general framework such that previous methods such as squeeze-and-excitation (SE) [14], convolutional block attention module (CBAM) [51], context-aggregation module (CAM) [58], and pixel-adaptive convolution (PAC) [42] can be seen as special cases of SAC, and experiments show that the more general SAC variants proposed in this paper outperform previous ones.

Using spatially-adaptive convolution, we build SqueezeSegV3 for LiDAR point-cloud segmentation. On the SemanticKITTI benchmark, SqueezeSegV3 outperforms all previously published methods by at least 3.7 mIoU with comparable inference speed, demonstrating the effectiveness of spatially-adaptive convolution.

2 Related work

2.1 Point-Cloud Segmentation

Recent papers on point-cloud segmentation can be divided into two categories - those that deal with small-scale point-clouds, and those that deal with large-scale point clouds. For small-scale point-cloud segmentation such as object part parsing and indoor scene understanding, mainstream methods are based on PointNet [35,36]. DGCNN [50] and Deep-KdNet [20] extend the hierarchical architecture of PointNet++ [36] by grouping neighbor points. Based on the PointNet architecture, [8,23,24] further improve the effectiveness of sampling, reordering and grouping to obtain a better representation for downstream tasks. PVCNN [27] improves the efficiency of PointNet-based methods [27,50] using voxel-based convolution with a contiguous memory access pattern. Despite these efforts, the efficiency of PointNet-based methods is still limited since they inherently need to process sparse data, which is more difficult to accelerate [27]. It is noteworthy to mention that the most recent RandLA-Net [15] significantly improves the speed of point cloud processing in the novel use of random sampling.

Large-scale point-cloud segmentation is challenging since 1) large-scale point-clouds are difficult to annotate and 2) many applications require real-time inference. Since a typical outdoor LiDAR (such as Velodyne HDL-64E) can collect about $200K$ points per scan, it is difficult for previous methods [22,38,47,26,37,31,29] to satisfy a real-time latency constraint. To address the data challenge, [56,48] proposed tools to label 3D bounding boxes and convert to point-wise segmentation labels. [56,58,62] proposed to train with simulated data. Recently, Behley *et al.* proposed SemanticKITTI [1], a densely annotated dataset for large-scale point-cloud segmentation. For efficiency, Wu *et al.* [56] proposed to project 3D point clouds to 2D and transform point-cloud segmentation to image segmentation. Later work [58,1,30] continued to improve the projection-based method, making it a popular choice for large-scale point-cloud segmentation.

2.2 Adaptive Convolution

Standard convolutions use the same weights to process input features at all spatial locations regardless of the input. Adaptive convolutions may change the weights according to the input and the location in the image. Squeeze-and-excitation and its variants [14,13,51] compute channel-wise or spatial attention to adapt the output feature map. Pixel-adaptive convolution (PAC) [42] changes the convolution weight along the kernel dimension with a Gaussian function. Wang *et al.* [49] propose to directly re-weight the standard convolution with a depth-aware Gaussian kernel. 3DNConv [5] further extends [49] by estimating depth through an RGB image and using it to improve image segmentation. In our work, we propose a more general framework such that channel-wise attention [14,13], spatial attention [51,58] and PAC [42] can be considered as special cases of spatially-adaptive convolution. In addition to adapting weights, deformable convolutions [6,66] adapt the location to pull features to convolution. DKN [19] combines both deformable convolution and adaptive convolution for joint-image filtering. However, deformable convolution is orthogonal to our proposed method.

2.3 Efficient Neural Networks

Many applications that involve point-cloud segmentation require real-time inference. To meet this requirement, we not only need to design efficient segmentation pipelines [58], but also efficient neural networks which optimize the parameter size, FLOPs, latency, power, and so on [52].

Many neural nets have been targeted to achieve efficiency, including SqueezeNet [16,10,54], MobileNets [12,39,11], ShiftNet [55,61], ShuffleNet [63,28], FBNet [53,57], ChamNet [7], MnasNet [44], and EfficientNet [45]. Previous work shows that using a more efficient backbone network can effectively improve efficiency in downstream tasks. In this paper, however, in order to rigorously evaluate the performance of spatially-adaptive convolution (SAC), we use the same backbone as RangeNet++ [30].

3 Spherical Projection of LiDAR Point-Cloud

To process a LiDAR point-cloud efficiently, Wu *et al.* [56] proposed a pipeline (shown in Figure 1) to project a sparse 3D point cloud to a 2D LiDAR image as

$$\begin{bmatrix} p \\ q \end{bmatrix} = \begin{bmatrix} \frac{1}{2}(1 - arctan(y,x)/\pi) \cdot w \\ (1 - (arcsin(z \cdot r^{-1}) + f_{up}) \cdot f^{-1}) \cdot h \end{bmatrix}, \tag{1}$$

where (x, y, z) are 3D coordinates, (p, q) are angular coordinates, (h, w) are the height and width of the desired projected 2D map, $f = f_{up} + f_{down}$ is the vertical field-of-view of the LiDAR sensor, and $r = \sqrt{x^2 + y^2 + z^2}$ is the range of each point. For each point projected to (p, q), we use its measurement of (x, y, z, r) and intensity as features and stack them along the channel dimension. This way, we can represent a LiDAR point cloud as a LiDAR image with the shape of

$(h, w, 5)$. Point-cloud segmentation can then be reduced to image segmentation, which is typically solved using ConvNets.

Despite the apparent similarity between LiDAR and RGB images, we discover that the spatial distribution of RGB features are quite different from (x, y, z, r) features. In Figure 2, we sample nine pixels on images from COCO [25], CIFAR10 [21] and SemanticKITTI [1] and compare their feature distribution. In COCO and CIFAR10, the feature distribution at different locations are rather similar. For SemanticKITTI, however, feature distribution at each locations are drastically different. Such spatially-varying distribution is caused by the spherical projection in Equation (1). In Figure 3 (top), we plot the mean of x, y, and z channels of LiDAR images. Along the width dimension, we can see the sinusoidal change of x and y channels. Along the height dimension, points projected to the top of the image have higher z-values than the ones projected to the bottom. As we will discuss later, such spatially varying distribution can degrade the performance of convolutions.

4 Spatially-Adaptive Convolution

4.1 Standard Convolution

Previous methods based on spherical projection [56,58,30] treat projected LiDAR images as RGB images and process them with standard convolution as

$$Y[m, p, q] = \sigma(\sum_{i,j,n} W[m, n, i, j] \times X[n, p + \hat{i}, q + \hat{j}]), \tag{2}$$

where $Y \in \mathbf{R}^{O \times S \times S}$ is the output tensor, $X \in \mathbf{R}^{I \times S \times S}$ denotes the input tensor, and $W \in \mathbf{R}^{O \times I \times K \times K}$ is the convolution weight. O, I, S, K are the output channel size, input channel size, image size, and kernel size of the weight, respectively. $\hat{i} = i - \lfloor K/2 \rfloor$, $\hat{j} = j - \lfloor K/2 \rfloor$. $\sigma(\cdot)$ is a non-linear activation function.

Convolution is based on a strong inductive bias that the distribution of visual features is invariant to image locations. For RGB images, this is a somewhat valid assumption, as illustrated in Figure 2. Therefore, regardless of the location, a convolution use the same weight W to process the input. This design makes the convolution operation very computationally efficient: First, convolutional layers are efficient in parameter size. Regardless of the input resolution S, a convolutional layer's parameter size remains the same as $O \times I \times K \times K$. Second, convolution is efficient to compute. In modern computer architectures, loading parameters into memory costs orders-of-magnitude higher energy and latency than floating point operations such as multiplications and additions [33]. For convolutions, we can load the parameter once and re-use for all the input pixels, which significantly improves the latency and power efficiency.

However, for LiDAR images, the feature distribution across the image are no longer identical, as illustrated in Figure 2 and 3 (top). Many features may only exist in local regions of the image, so the filters that are trained to process them are only active in the corresponding regions and are not useful elsewhere. To

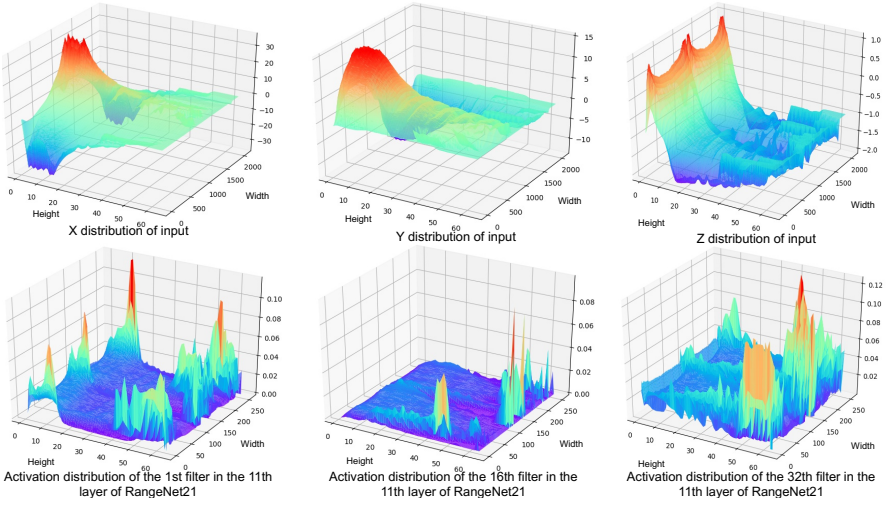

Fig. 3: Channel and filter activation visualization on the SemanticKITTI dataset. Top: we visualize the mean value of x, y, and z channels of the projected LiDAR images at different locations. Along the width dimension, we can see the sinusoidal change of the x and y channels. Along the height dimension, we can see z values are higher at the top of the image. Bottom: We visualize the mean activation value of three filters at the 11th layer of a pre-trained RangeNet21 [30]. We can see that those filters are sparsely activated only in certain areas.

confirm this, we analyze a trained RangeNet21 [30] by calculating the average filter activation across the image. We can see in Figure 3 (bottom) that convolutional filters are sparsely activated and remain zero in many regions. This validates that convolution filters are spatially under-utilized.

4.2 Spatially-Adaptive Convolution

To better process LiDAR images with spatially-varying feature distributions, we re-design convolution to achieve two goals: 1) It should be spatially-adaptive and content-aware. The new operator should process different parts of the image with different filters, and the filters should adapt to feature variations. 2) The new operator should be efficient to compute.

To achieve these goals, we propose Spatially-Adaptive Convolution (SAC), which can be described as the following:

$$Y[m,p,q] = \sigma(\sum_{i,j,n} W(X_0)[m,n,p,q,i,j] \times X[n, p+\hat{i}, q+\hat{j}]). \tag{3}$$

$W(\cdot) \in \mathbf{R}^{O\times I\times S\times S\times K\times K}$ is a function of the raw input X_0. It is spatially-adaptive, since W depends on the location (p,q). It is content-aware since W

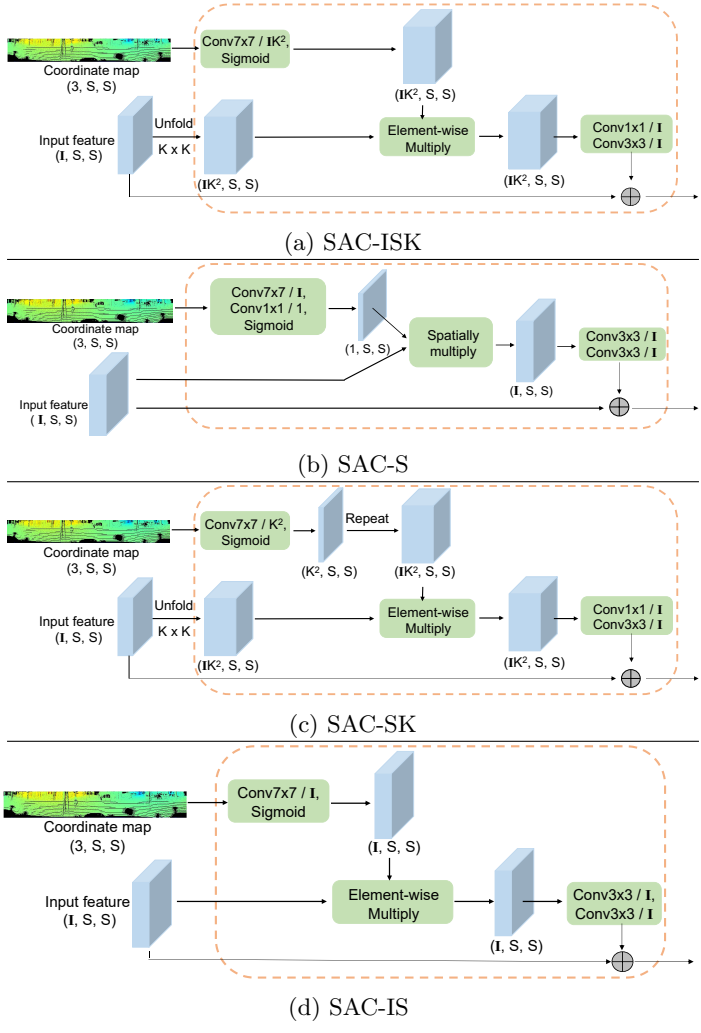

Fig. 4: Variants of spatially-adaptive convolution used in Figure 1.

is a function of the raw input X_0. Computing W in this general form is very expensive since W contains too many elements to compute.

To reduce the computational cost, we factorize W as the product of a standard convolution weight and a spatially-adaptive attention map as:

$$W[m,n,p,q,i,j] = \hat{W}[m,n,i,j] \times A(X_0)[m,n,p,q,i,j]. \tag{4}$$

$\hat{W} \in \mathbf{R}^{O\times I\times S\times S}$ is a standard convolution weight, and $A \in \mathbf{R}^{O\times I\times S\times S\times K\times K}$ is the attention map. To reduce the complexity, we collapse several dimensions of A to obtain a smaller attention map to make it computationally tractable.

Table 1: Extra parameters and MACs for different SAC variants in SqueezeSegV3-21

Method	O	I	S	K	Extra Params (%)	Extra MACs (%)
SAC-S	✗	✗	✓	✗	1.1	2.4
SAC-IS	✗	✓	✓	✗	2.2	6.2
SAC-SK	✗	✗	✓	✓	1.9	3.1
SAC-ISK	✗	✓	✓	✓	14.9	24.8

We denote the first dimension of A as the output channel dimension (O), the second as the input channel dimension (I), the 3rd and 4th dimensions as spatial dimensions (S), and the last two dimensions as kernel dimensions (K).

Starting from Equation (4), we name this form of SAC as SAC-OISK, and we re-write A as A_{OISK}, where the subscripts denote the dimensions that are not collapsed to 1. If we collapse the output dimension, we name the variant as SAC-ISK, and the attention map as $A_{ISK} \in \mathbf{R}^{1\times I\times S\times S\times K\times K}$. SAC-ISK adapts a convolution weight spatially as well as across the kernel and input channel dimensions, as shown in Figure 4a. We can further compress the kernel dimensions to obtain SAC-IS with $A_{IS} \in \mathbf{R}^{1\times I\times S\times S\times 1\times 1}$, (Figure 4d) and SAC-S with pixel-wise attention as $A_S \in \mathbf{R}^{1\times 1\times S\times S\times 1\times 1}$ (Figure 4b).

As long as we retain the spatial dimension A, SAC is able to spatially adapt a standard convolution. Experiments show that all variants of SAC effectively improve the performance on the SemanticKITTI dataset.

4.3 Efficient Computation of SAC

To efficiently compute an attention map, we feed the raw LiDAR image X_0 into a 7x7 convolution followed by a sigmoid activation. The convolution computes the values of the attention map at each location. The more dimensions to adapt, the more FLOPs and parameter size SAC requires. However, most of the variants of SAC are very efficient. Taking SqueezeSegV3-21 as an example, the cost of adding different SAC variants is summarized in Table 1. The extra FLOPs (2.4% - 24.8%) and parameters (1.1% - 14.9%) needed by SAC is quite small.

After obtaining the attention map, we need to efficiently compute the product of the convolution weight $\hat{W}$, attention map A, and the input X. One choice is to first compute the adaptive weight as Equation (4) and then process the input X. However, the adaptive weight varies per pixel, so we are no longer able to re-use the weight spatially to retain the efficiency of standard convolution.

So, instead, we first combine the attention map A with the input tensor X. For attention maps without kernel dimensions, such as A_S or A_{IS}, we directly perform element-wise multiplication (with broadcasting) between A and X. Then, we apply a standard convolution with weight W on the adapted input. The examples of SAC-S and SAC-IS are illustrated in Figures 4b and 4d respectively. Pseudo-code implementation is provided in the supplementary material.

For attention maps with kernel dimensions, such as A_{ISK} and A_{SK}, we first perform an unfolding (im2col) operation on X. At each location, we collect

nearby K-by-K features and stack them along the channel dimension to get $\tilde{X} \in \mathbf{R}^{K^2 I \times S \times S}$. Then, we can apply element-wise multiplication to combine the attention map A and input X. Next, we reshape weight $W \in \mathbf{R}^{O \times I \times K \times K}$ as $\tilde{W} \in \mathbf{R}^{O \times K^2 I \times 1 \times 1}$. Finally, the output of Y can be obtained by applying a 1-by-1 convolution with $\tilde{W}$ on $\tilde{X}$. The computation of SAC-ISK and SAC-SK is shown in Figures 4a and 4c respectively, and the pseudo-code implementation is provided in the supplementary material.

Overall, SAC can be implemented as a series of element-wise multiplications, im2col, reshaping, and standard convolution operations, which are widely supported and well optimized. This ensures that SAC can be computed efficiently.

4.4 Relationship with Prior Work

Several prior works can be seen as variants of a spatially-adaptive convolution, as described by Equations 3 and 4. Squeeze-and-Excitation (SE) [14,13] uses global average pooling and fully-connected layers to compute channel-wise attention to adapt the feature map, as illustrated in Figure 5a. It can be seen as the variant of SAC-I with a attention map of $A_I \in \mathbf{R}^{1 \times I \times 1 \times 1 \times 1 \times 1}$. The convolutional block attention module (CBAM) [51] can be see as applying A_I followed by an A_S to adapt the feature map, as shown in Figure 5b. SqueezeSegV2 [58] uses the context-aggregation module (CAM) to combat dropout noises in LiDAR images. At each position, it uses a 7x7 max pooling followed by 1x1 convolutions to compute a channel-wise attention map. It can be seen as the variant SAC-IS with the attention map of $A_{IS} \in \mathbf{R}^{1 \times I \times S \times S \times 1 \times 1}$ as illustrated in Figure 5c. Pixel-adaptive convolution (PAC) [42] uses a Gaussian function to compute kernel-wise attention for each pixel. It can be seen as the variant of SAC-SK, with the attention map of $A_{SK} \in \mathbf{R}^{1 \times 1 \times S \times S \times K \times K}$, as illustrated in Figure 5d. Our ablation studies compare variants of SAC, including ones proposed in our paper and in prior work. Experiments show our proposed SAC variants outperform previous baselines.

5 SqueezeSegV3

Using the spatially-adaptive convolution, we build SqueezeSegV3 for LiDAR point-cloud segmentation. The overview of the model is shown in Figure 1.

5.1 The Architecture of SqueezeSegV3

To facilitate rigorous comparison, SqueezeSegV3's backbone architecture is based on RangeNet [30]. RangeNet contains five stages of convolution, each stage contains several blocks. At the beginning of the stage, it performs downsampling. The output is then upsampled to recover the resolution. Each block of RangeNet contains two stacked convolutions. We replace the first one with SAC-ISK as in Figure 4a. We remove the last two downsampling. To keep the same FLOPs, we reduce the channels of last two stages. The output channel sizes from $Stage1$

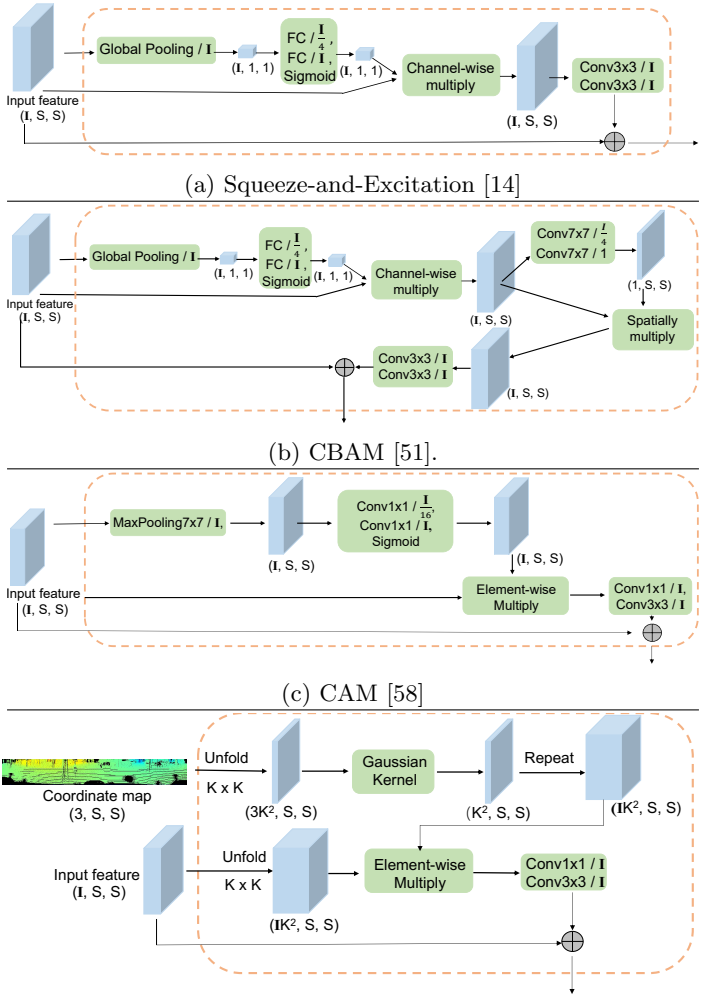

Fig. 5: Variants of spatially-adaptive convolution from previous work.

to *Stage*5 are 64, 128, 256, 256 and 256 respectively, while the output channel sizes in RangeNet [30] are 64, 128, 256, 512 and 1024. Due to the removal of the last two downsampling operations, we only adopt 3 upsample blocks using transposed convolution and convolution.

5.2 Loss Function

We introduce a multi-layer cross entropy loss to train the proposed network, which is also used in [40,18,60,32]. During training, from *stage*1 to *stage*5, we

add a prediction layer at each stage's output. For each output, we respectively downsample the groundtruth label map by 1x, 2x, 4x, 8x and 8x, and use them to train the output of *stage*1 to *stage*5. The loss function can be described as

$$L = \sum_{i=1}^{5} \frac{-\sum_{H_i,W_i} \sum_{c=1}^{C} w_c \cdot y_c \cdot log(\hat{y}_c)}{H_i \times W_i}. \tag{5}$$

In the equation, $w_c = \frac{1}{log(f_c+\epsilon)}$ is a normalization factor and f_c is the frequency of class c. H_i, W_i are the height and width of the output in i-th stage, y_c is the prediction for the c-th class in each pixel and $\hat{y}_c$ is the label. Compared to the single-stage cross-entropy loss used for the final output, the intermediate supervisions guide the model to form features with more semantic meaning. In addition, they help mitigate the vanishing gradient problem in training.

6 Experiments

6.1 Dataset and Evaluation Metrics

We conduct our experiments on the SemanticKITTI dataset [1], a large-scale dataset for LiDAR point-cloud segmentation. The dataset contains 21 sequences of point-cloud data with 43,442 densely annotated scans and total 4549 millions points. Following [1], sequences-{0-7} and {9, 10} (19130 scans) are used for training, sequence-08 (4071 scans) is for validation, and sequences-{11-21} (20351 scans) are for test. Following previous work [30], we use mIoU over 19 categories to evaluate the accuracy.

6.2 Implementation Details

We pre-process all the points by spherical projection following Equation (1). The 2D LiDAR images are then processed by SqueezeSegV3 to get a 2D predicted label map, which is then restored back to the 3D space. Following previous work [30,56,58], we project all points in a scan to a 64×2048 image. If multiple points are projected to the same pixel on the 2D image, we keep the point with the largest distance. Following RangeNet21 and RangeNet53 in [30], we propose SqueezeSegV3-21 (SSGV3-21) and SqueezeSegV3-53 (SSGV3-53). The model architecture of SSGV3-21 and SSGV3-53 are similar to RangeNet21 and RangNet53 [30], except that we replace regular convolution blocks with SAC blocks. Both models contain 5 stages, each of them has a different input resolution. In SSGV3-21, the 5 stages respectively contain 1, 1, 2, 2, 1 blocks and in SSGV3-53, the 5 stages contain 1, 2, 8, 8, 4 blocks, which are also same as RangeNet21 and RangeNet53, respectively.

We use the SGD optimizer to end-to-end train the whole model. During training, SSGV3-21 is trained with an initial learning rate of 0.01, SSGV3-53 is trained with an initial learning rate of 0.005. We use the warming up strategy to change the learning rate for 1 epoch. During inference, the original points will be projected and fed into SqueezeSegV3 to get a 2D prediction. Then we adopt the restoration operation to obtain the 3D prediction, as previous work [30,56,58].

Table 2: IoU [%] on test set (sequences 11 to 21). SSGV3-21 and SSGV3-53 are the proposed method. Their complexity corresponds to RangeNet21 and RangeNet53 respectively. * means KNN post-processing from RangeNet++ [30], and ‡ means the CRF post-processing from SqueezeSegV2 used [58]. The first group reports PointNet-based methods. The second reports projection-based methods. The third include our results

Method	car	bicycle	motorcycle	truck	other-vehicle	person	bicyclist	Motorcyclist	road	parking	sidewalk	other-ground	building	fence	vegetation	trunk	terrain	pole	traffic-sign	mean IoU	Scans/sec
PNet [35]	46.3	1.3	0.3	0.1	0.8	0.2	0.2	0.0	61.6	15.8	35.7	1.4	41.4	12.9	31.0	4.6	17.6	2.4	3.7	14.6	2
PNet++ [36]	53.7	1.9	0.2	0.9	0.2	0.9	1.0	0.0	72.0	18.7	41.8	5.6	62.3	16.9	46.5	13.8	30.0	6.0	8.9	20.1	0.1
SPGraph [22]	68.3	0.9	4.5	0.9	0.8	1.0	6.0	0.0	49.5	1.7	24.2	0.3	68.2	22.5	59.2	27.2	17.0	18.3	10.5	20.0	0.2
SPLAT [43]	66.6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	70.4	0.8	41.5	0.0	68.7	27.8	72.3	35.9	35.8	13.8	0.0	22.8	1
TgConv [46]	86.8	1.3	12.7	11.6	10.2	17.1	20.2	0.5	82.9	15.2	61.7	9.0	82.8	44.2	75.5	42.5	55.5	30.2	22.2	35.9	0.3
RLNet [15]	**94.0**	19.8	21.4	**42.7**	**38.7**	**47.5**	**48.8**	4.6	90.4	56.9	67.9	15.5	81.1	49.7	78.3	**60.3**	59.0	44.2	38.1	50.3	22
SSG [56]	68.8	16.0	4.1	3.3	3.6	12.9	13.1	0.9	85.4	26.9	54.3	4.5	57.4	29.0	60.0	24.3	53.7	17.5	24.5	29.5	**65**
SSG‡ [56]	68.3	18.1	5.1	4.1	4.8	16.5	17.3	1.2	84.9	28.4	54.7	4.6	61.5	29.2	59.6	25.5	54.7	11.2	36.3	30.8	53
SSGV2 [58]	81.8	18.5	17.9	13.4	14.0	20.1	25.1	3.9	88.6	45.8	67.6	17.7	73.7	41.1	71.8	35.8	60.2	20.2	36.3	39.7	50
SSGV2‡ [58]	82.7	21.0	22.6	14.5	15.9	20.2	24.3	2.9	88.5	42.4	65.5	18.7	73.8	41.0	68.5	36.9	58.9	12.9	41.0	39.6	39
RGN21 [30]	85.4	26.2	26.5	18.6	15.6	31.8	33.6	4.0	91.4	57.0	74.0	26.4	81.9	52.3	77.6	48.4	63.6	36.0	50.0	47.4	20
RGN53 [30]	86.4	24.5	32.7	25.5	22.6	36.2	33.6	4.7	**91.8**	64.8	74.6	**27.9**	84.1	55.0	78.3	50.1	64.0	38.9	52.2	49.9	12
RGN53* [30]	91.4	25.7	34.4	25.7	23.0	38.3	38.8	4.8	**91.8**	**65.0**	**75.2**	27.8	87.4	58.6	80.5	55.1	64.6	47.9	55.9	52.2	11
SSGV3-21	84.6	31.5	32.4	11.3	20.9	39.4	36.1	**21.3**	90.8	54.1	72.9	23.9	81.1	50.3	77.6	47.7	63.9	36.1	51.7	48.8	16
SSGV3-53	87.4	35.2	33.7	29.0	31.9	41.8	39.1	20.1	**91.8**	63.5	74.4	27.2	85.3	55.8	79.4	52.1	64.7	38.6	53.4	52.9	7
SSGV3-21*	89.4	33.7	34.9	11.3	21.5	42.6	44.9	21.2	90.8	54.1	73.3	23.2	84.8	53.6	80.2	53.3	64.5	46.4	57.6	51.6	15
SSGV3-53*	92.5	**38.7**	**36.5**	29.6	33.0	45.6	46.2	20.1	91.7	63.4	74.8	26.4	**89.0**	**59.4**	**82.0**	58.7	**65.4**	**49.6**	**58.9**	**55.9**	6

6.3 Comparing with Prior Methods

We compare two proposed models, SSGV3-21 and SSGV3-53, with previous published work [35,36,22,43,46,56,58,30]. From Table 2, we can see that the proposed SqueezeSegV3 models outperforms all the baselines. Compared with the previous state-of-the-art RangeNet53 [30], SSGV3-53 improves the accuracy by 3.0 mIoU. Moreover, when we apply post-processing KNN refinement following [30] (indicated as *), the proposed SSGV3-53* outperforms RangeNet53* by 3.7 mIoU and achieves the best accuracy in 14 out of 19 categories. Meanwhile, the proposed SSGV3-21 also surpasses RangeNet21 by 1.4 mIoU and the performance is close to RangeNet53* with post-processing. The advantages are more significant for smaller objects, as SSGV3-53* significantly outperforms RangeNet53* by 13.0 IoU, 10.0 IoU, 7.4 IoU and 15.3 IoU in categories of bicycle, other-vehicle, bicyclist and Motorcyclist respectively.

In terms of speed, SSGV3-21 (16 FPS) is closet RangeNet21 (20 FPS). Even though SSGV3-53 (7 FPS) is slower than RangeNet53 (12 FPS), note that our implementation of SAC is primitive and it can be optimized to achieve further speedup. In comparison, PointNet-based methods [35,36,22,43,46] do not perform well in either accuracy and speed except RandLA-Net [15] which is a new efficient and effective work.

Table 3: mIoU [%] and Accuracy [%] for variants of spatially-adaptive convolution

Method	Baseline	SAC-S	SAC-IS	SAC-SK	SAC-ISK	PAC [42]	SE [14]	CBAM [51]	CAM [58]
mIoU	44.0	44.9	44.0	45.4	46.3	45.2	44.2	44.8	42.1
Accuracy	86.8	87.6	86.9	88.2	88.6	88.2	87.0	87.5	85.8

6.4 Ablation Study

We conduct ablation studies to analyze the performance of SAC with different configurations. Also, we compare it with other related operators to show its effectiveness. To facilitate fast training and experiments, we shrink the LiDAR images to 64×512, and use the shallower model of SSGV3-21 as the starting point. We evaluate the accuracy directly on the projected 2D image, instead of the original 3D points, to make the evaluation faster. We train the models in this section on the training set of SemanticKITTI and report the accuracy on the validation set. We study different variations of SAC, input kernel sizes, and other techniques used in SqueezeSegV3.

Variants of spatially-adaptive convolution: As shown in Figure 4 & 5, spatially-adaptive convolution can have many variation. Some variants are equivalent to or similar with methods proposed by previous papers, including squeeze-and-excitation (SE) [14], convolutional block attention maps (CBAM) [51], pixel-adaptive convolution (PAC) [42], and context-aggregation module (CAM) [58]. To understand the effectiveness of SAC variants and previous methods, we swap them into SqueezeSegV3-21. The results are reported in Table. 3.

It can be seen that SAC-ISK significantly outperforms all the other settings in term of mIoU. CAM and SAC-IS have the worst performance, which demonstrates the importance of the attention on the kernel dimension. Squeeze-and-excitation (SE) also does not perform well, since SE is not spatially-adaptive, and the global average pooling used in SE ignores the feature distribution shift across the LiDAR image. In comparison, CBAM [51] improves the baseline by 0.8 mIoU. Unlike SE, it also adapts the input feature spatially. This comparison shows that being spatially-adaptive is crucial for processing LiDAR images. Pixel-adaptive convolution (PAC) is similar to the SAC variant of SAC-SK, except that PAC uses a Gaussian function to compute the kernel-wise attention. Experiments show that the proposed SAC-SK slightly outperforms SAC-SK, possibly because SAC-SK adopts a more general and learnable convolution to compute the attention map. Comparing SAC-S and SAC-IS, adding the input channel dimension does not improve the performance.

Kernel Sizes of SAC: We use a one-layer convolution to compute the attention map for SAC. However, what should be the kernel size for this convolution? A larger kernel size makes sure that it can capture spatial information around, but it also costs more parameters and MACs. To examine the influence of kernel size, we use different kernel sizes in the SAC convolution. As we can see in Table 4, a 1x1 convolution provides a very strong result that is better than its 3x3 and 5x5 counterparts. 7x7 convolution performs the best.

Table 4: mIoU [%] and Accuracy [%] for different convolution kernel sizes for coordinate map

Kernel size	baseline	1×1	3×3	5×5	7×7
mIoU	44.0	45.5	44.5	45.4	46.3
Accuracy	86.8	88.4	87.6	88.2	88.6

Table 5: mIoU [%] and Accuracy [%] with downsampling removal, multi-layer loss, and spatially-adaptive convolution

method	Baseline	+DS removal	+Multi-layer loss	+SAC-ISK
mIoU	38.6	42.5 (+3.9)	44.0 (+1.5)	46.3 (+2.3)
Accuracy	84.7	86.2 (+1.5)	86.8 (+1.4)	88.6 (+1.8)

The effectiveness of other techniques: In addition to SAC, we also introduce several new techniques to SqueezeSegV3, including removing the last two downsample layers and multi-layer loss. We start from the baseline of RangeNet21. First, we remove downsampling layers and reduce the channel sizes of the last two stages to 256 to keep the MACS the same. The performance improves by 3.9 mIoU. After adding the multi-layer loss, the mIoU increases by another 1.5%. Based on the above techniques, adding SAC-ISK further boost mIoU by 2.3%.

References

1. Behley, J., Garbade, M., Milioto, A., Quenzel, J., Behnke, S., Stachniss, C., Gall, J.: SemanticKITTI: A Dataset for Semantic Scene Understanding of LiDAR Sequences. In: Proc. of the IEEE/CVF International Conf. on Computer Vision (ICCV) (2019)
2. Behley, J., Stachniss, C.: Efficient surfel-based slam using 3d laser range data in urban environments. In: Robotics: Science and Systems (2018)
3. Chen, X., Ma, H., Wan, J., Li, B., Xia, T.: Multi-view 3d object detection network for autonomous driving. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 1907–1915 (2017)
4. Chen, X., Milioto, A., Palazzolo, E., Giguère, P., Behley, J., Stachniss, C.: Suma++: Efficient lidar-based semantic slam. In: 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 4530–4537. IEEE (2019)
5. Chen, Y., Mensink, T., Gavves, E.: 3d neighborhood convolution: Learning depth-aware features for rgb-d and rgb semantic segmentation. In: 2019 International Conference on 3D Vision (3DV). pp. 173–182. IEEE (2019)
6. Dai, J., Qi, H., Xiong, Y., Li, Y., Zhang, G., Hu, H., Wei, Y.: Deformable convolutional networks. In: Proceedings of the IEEE international conference on computer vision. pp. 764–773 (2017)
7. Dai, X., Zhang, P., Wu, B., Yin, H., Sun, F., Wang, Y., Dukhan, M., Hu, Y., Wu, Y., Jia, Y., et al.: Chamnet: Towards efficient network design through platform-aware model adaptation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 11398–11407 (2019)
8. Dovrat, O., Lang, I., Avidan, S.: Learning to sample. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2760–2769 (2019)
9. Geiger, A., Lenz, P., Stiller, C., Urtasun, R.: Vision meets robotics: The kitti dataset. The International Journal of Robotics Research **32**(11), 1231–1237 (2013)
10. Gholami, A., Kwon, K., Wu, B., Tai, Z., Yue, X., Jin, P., Zhao, S., Keutzer, K.: Squeezenext: Hardware-aware neural network design. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops. pp. 1638–1647 (2018)
11. Howard, A., Sandler, M., Chu, G., Chen, L.C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., et al.: Searching for mobilenetv3. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 1314–1324 (2019)
12. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861 (2017)
13. Hu, J., Shen, L., Albanie, S., Sun, G., Vedaldi, A.: Gather-excite: Exploiting feature context in convolutional neural networks. In: Advances in Neural Information Processing Systems. pp. 9401–9411 (2018)
14. Hu, J., Shen, L., Sun, G.: Squeeze-and-excitation networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 7132–7141 (2018)
15. Hu, Q., Yang, B., Xie, L., Rosa, S., Guo, Y., Wang, Z., Trigoni, N., Markham, A.: Randla-net: Efficient semantic segmentation of large-scale point clouds. arXiv preprint arXiv:1911.11236 (2019)
16. Iandola, F.N., Han, S., Moskewicz, M.W., Ashraf, K., Dally, W.J., Keutzer, K.: Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. arXiv preprint arXiv:1602.07360 (2016)

17. Jaritz, M., Vu, T.H., de Charette, R., milie Wirbel, Prez, P.: xmuda: Cross-modal unsupervised domain adaptation for 3d semantic segmentation (2019)
18. Johnson, J., Alahi, A., Fei-Fei, L.: Perceptual losses for real-time style transfer and super-resolution. In: European conference on computer vision. pp. 694–711. Springer (2016)
19. Kim, B., Ponce, J., Ham, B.: Deformable kernel networks for joint image filtering. arXiv preprint arXiv:1910.08373 (2019)
20. Klokov, R., Lempitsky, V.: Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 863–872 (2017)
21. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images (2009)
22. Landrieu, L., Simonovsky, M.: Large-scale point cloud semantic segmentation with superpoint graphs. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4558–4567 (2018)
23. Li, J., Chen, B.M., Hee Lee, G.: So-net: Self-organizing network for point cloud analysis. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 9397–9406 (2018)
24. Li, Y., Bu, R., Sun, M., Wu, W., Di, X., Chen, B.: Pointcnn: Convolution on x-transformed points. In: Advances in neural information processing systems. pp. 820–830 (2018)
25. Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft coco: Common objects in context. In: European conference on computer vision. pp. 740–755. Springer (2014)
26. Liu, F., Li, S., Zhang, L., Zhou, C., Ye, R., Wang, Y., Lu, J.: 3dcnn-dqn-rnn: A deep reinforcement learning framework for semantic parsing of large-scale 3d point clouds. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 5678–5687 (2017)
27. Liu, Z., Tang, H., Lin, Y., Han, S.: Point-voxel cnn for efficient 3d deep learning. In: Advances in Neural Information Processing Systems. pp. 963–973 (2019)
28. Ma, N., Zhang, X., Zheng, H.T., Sun, J.: Shufflenet v2: Practical guidelines for efficient cnn architecture design. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 116–131 (2018)
29. Meng, H.Y., Gao, L., Lai, Y.K., Manocha, D.: Vv-net: Voxel vae net with group convolutions for point cloud segmentation. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 8500–8508 (2019)
30. Milioto, A., Vizzo, I., Behley, J., Stachniss, C.: Rangenet++: Fast and accurate lidar semantic segmentation. In: Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS) (2019)
31. Mo, K., Zhu, S., Chang, A.X., Yi, L., Tripathi, S., Guibas, L.J., Su, H.: Partnet: A large-scale benchmark for fine-grained and hierarchical part-level 3d object understanding. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 909–918 (2019)
32. Newell, A., Yang, K., Deng, J.: Stacked hourglass networks for human pose estimation. In: European conference on computer vision. pp. 483–499. Springer (2016)
33. Pedram, A., Richardson, S., Horowitz, M., Galal, S., Kvatinsky, S.: Dark memory and accelerator-rich system optimization in the dark silicon era. IEEE Design & Test **34**(2), 39–50 (2016)
34. Qi, C.R., Liu, W., Wu, C., Su, H., Guibas, L.J.: Frustum pointnets for 3d object detection from rgb-d data. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 918–927 (2018)

35. Qi, C.R., Su, H., Mo, K., Guibas, L.J.: Pointnet: Deep learning on point sets for 3d classification and segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 652–660 (2017)
36. Qi, C.R., Yi, L., Su, H., Guibas, L.J.: Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In: Advances in neural information processing systems. pp. 5099–5108 (2017)
37. Rethage, D., Wald, J., Sturm, J., Navab, N., Tombari, F.: Fully-convolutional point networks for large-scale point clouds. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 596–611 (2018)
38. Riegler, G., Osman Ulusoy, A., Geiger, A.: Octnet: Learning deep 3d representations at high resolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 3577–3586 (2017)
39. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.C.: Mobilenetv2: Inverted residuals and linear bottlenecks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 4510–4520 (2018)
40. Shen, W., Wang, B., Jiang, Y., Wang, Y., Yuille, A.: Multi-stage multi-recursive-input fully convolutional networks for neuronal boundary detection. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 2391–2400 (2017)
41. Song, S., Xiao, J.: Deep sliding shapes for amodal 3d object detection in rgb-d images. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 808–816 (2016)
42. Su, H., Jampani, V., Sun, D., Gallo, O., Learned-Miller, E., Kautz, J.: Pixel-adaptive convolutional neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 11166–11175 (2019)
43. Su, H., Jampani, V., Sun, D., Maji, S., Kalogerakis, E., Yang, M.H., Kautz, J.: Splatnet: Sparse lattice networks for point cloud processing. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2530–2539 (2018)
44. Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., Le, Q.V.: Mnasnet: Platform-aware neural architecture search for mobile. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2820–2828 (2019)
45. Tan, M., Le, Q.V.: Efficientnet: Rethinking model scaling for convolutional neural networks. arXiv preprint arXiv:1905.11946 (2019)
46. Tatarchenko, M., Park, J., Koltun, V., Zhou, Q.Y.: Tangent convolutions for dense prediction in 3d. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 3887–3896 (2018)
47. Tchapmi, L., Choy, C., Armeni, I., Gwak, J., Savarese, S.: Segcloud: Semantic segmentation of 3d point clouds. In: 2017 international conference on 3D vision (3DV). pp. 537–547. IEEE (2017)
48. Wang, B., Wu, V., Wu, B., Keutzer, K.: Latte: accelerating lidar point cloud annotation via sensor fusion, one-click annotation, and tracking. In: 2019 IEEE Intelligent Transportation Systems Conference (ITSC). pp. 265–272. IEEE (2019)
49. Wang, W., Neumann, U.: Depth-aware cnn for rgb-d segmentation. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 135–150 (2018)
50. Wang, Y., Sun, Y., Liu, Z., Sarma, S.E., Bronstein, M.M., Solomon, J.M.: Dynamic graph cnn for learning on point clouds. ACM Transactions on Graphics (TOG) **38**(5), 1–12 (2019)

51. Woo, S., Park, J., Lee, J.Y., So Kweon, I.: Cbam: Convolutional block attention module. In: Proceedings of the European Conference on Computer Vision (ECCV). pp. 3–19 (2018)
52. Wu, B.: Efficient deep neural networks. arXiv preprint arXiv:1908.08926 (2019)
53. Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., Tian, Y., Vajda, P., Jia, Y., Keutzer, K.: Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 10734–10742 (2019)
54. Wu, B., Iandola, F., Jin, P.H., Keutzer, K.: Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops. pp. 129–137 (2017)
55. Wu, B., Wan, A., Yue, X., Jin, P., Zhao, S., Golmant, N., Gholaminejad, A., Gonzalez, J., Keutzer, K.: Shift: A zero flop, zero parameter alternative to spatial convolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 9127–9135 (2018)
56. Wu, B., Wan, A., Yue, X., Keutzer, K.: Squeezeseg: Convolutional neural nets with recurrent crf for real-time road-object segmentation from 3d lidar point cloud. ICRA (2018)
57. Wu, B., Wang, Y., Zhang, P., Tian, Y., Vajda, P., Keutzer, K.: Mixed precision quantization of convnets via differentiable neural architecture search. arXiv preprint arXiv:1812.00090 (2018)
58. Wu, B., Zhou, X., Zhao, S., Yue, X., Keutzer, K.: Squeezesegv2: Improved model structure and unsupervised domain adaptation for road-object segmentation from a lidar point cloud. In: ICRA (2019)
59. Xie, Y., Tian, J., Zhu, X.X.: A review of point cloud semantic segmentation. arXiv preprint arXiv:1908.08854 (2019)
60. Xu, C., Qiu, K., Fu, J., Bai, S., Xu, Y., Bai, X.: Learn to scale: Generating multipolar normalized density maps for crowd counting. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 8382–8390 (2019)
61. Yang, Y., Huang, Q., Wu, B., Zhang, T., Ma, L., Gambardella, G., Blott, M., Lavagno, L., Vissers, K., Wawrzynek, J., et al.: Synetgy: Algorithm-hardware co-design for convnet accelerators on embedded fpgas. In: Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. pp. 23–32 (2019)
62. Yue, X., Wu, B., Seshia, S.A., Keutzer, K., Sangiovanni-Vincentelli, A.L.: A lidar point cloud generator: from a virtual world to autonomous driving. In: Proceedings of the 2018 ACM on International Conference on Multimedia Retrieval. pp. 458–464 (2018)
63. Zhang, X., Zhou, X., Lin, M., Sun, J.: Shufflenet: An extremely efficient convolutional neural network for mobile devices. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 6848–6856 (2018)
64. Zhou, Y., Sun, P., Zhang, Y., Anguelov, D., Gao, J., Ouyang, T., Guo, J., Ngiam, J., Vasudevan, V.: End-to-end multi-view fusion for 3d object detection in lidar point clouds. arXiv preprint arXiv:1910.06528 (2019)
65. Zhou, Y., Tuzel, O.: Voxelnet: End-to-end learning for point cloud based 3d object detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4490–4499 (2018)
66. Zhu, X., Hu, H., Lin, S., Dai, J.: Deformable convnets v2: More deformable, better results. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 9308–9316 (2019)

7 Appendix

In this appendix, we provide the pseudo code implementation for SAC variants discussed in Section 4.2, including SAC-S, SAC-IS, SAC-SK and SAC-ISK.

```
""" The input of each function is input_feature and coordinate map.
input_feature (N, C, H, W), coordinate_map (N, 3, H, W)
output_feature (N, C, H, W) """


def SAC_S(input_feature, coordi_map):
# Note: Pseudo code for SAC-S.
  attention_map = Conv_attention7x7(coordin_map) # (N, 1, H, W)
  input_feature = input_feature * attention_map # (N, C, H, W)
  feature = Conv_feature3x3(input_feature) # (N, C, H, W)
  output_feature = feature+input_feature # (N, C, H, W)
  return output_feature # (N, C, H, W)

def SAC_IS(input_feature, coordi_map):
# Note: Pseudo code for SAC-IS.
  attention_map = Conv_attention7x7(coordin_map) # (N, C, H, W)
  input_feature = input_feature * attention_map # (N, C, H, W)
  feature = Conv_feature3x3(input_feature) # (N, C, H, W)
  output_feature = feature+input_feature # (N, C, H, W)
  return output_feature # (N, C, H, W)

def SAC_SK(input_feature, coordi_map):
# Note: Pseudo code for SAC-SK.
  unfold_feature = unfold(input_feature, kernel_size=K,
  padding=K//2) # (N, C*K*K, H, W)
  attention_map = Conv_attention7x7(coordin_map) # (N, K*K, H, W)
  attention_map = attention_map.repeat(1, C, 1, 1) # (N, C*K*K, H, W)
  input_feature = input_feature * attention_map # (N, C*K*K, H, W)
  feature = Conv_feature1x1(input_feature) # (N, C, H, W)
  feature = Conv_feature3x3(feature) # (N, C, H, W)
  output_feature = feature+input_feature # (N, C, H, W)
  return output_feature # (N, C, H, W)


def SAC_ISK(input_feature, coordi_map):
# Note: Pseudo code for SAC-ISK.
  unfold_feature = unfold(input_feature, kernel_size=K,
  padding=K//2) # (N, C*K*K, H, W )
  attention_map = Conv_attention7x7(coordin_map) # (N, C*K*K, H, W)
  input_feature = input_feature * attention_map # (N, C*K*K, H, W)
  feature = Conv_feature1x1(input_feature) # (N, C, H, W)
  feature = Conv_feature3x3(feature) # (N, C, H, W)
```

```
output_feature = feature+input_feature # (N, C, H, W)
return output_feature # (N, C, H, W)
```