Searching Efficient 3D Architectures with Sparse Point-Voxel Convolution

Haotian Tang[1,*], Zhijian Liu[1,*],
Shengyu Zhao[1,2], Yujun Lin[1], Ji Lin[1], Hanrui Wang[1], and Song Han[1]

[1] Massachusetts Institute of Technology
[2] IIIS, Tsinghua University

Abstract. Self-driving cars need to understand 3D scenes efficiently and accurately in order to drive safely. Given the limited hardware resources, existing 3D perception models are not able to recognize small instances (*e.g.*, pedestrians, cyclists) very well due to the low-resolution voxelization and aggressive downsampling. To this end, we propose *Sparse Point-Voxel Convolution (SPVConv)*, a lightweight 3D module that equips the vanilla Sparse Convolution with the high-resolution point-based branch. With negligible overhead, this point-based branch is able to preserve the fine details even from large outdoor scenes. To explore the spectrum of efficient 3D models, we first define a flexible architecture design space based on SPVConv, and we then present *3D Neural Architecture Search (3D-NAS)* to search the optimal network architecture over this diverse design space efficiently and effectively. Experimental results validate that the resulting SPVNAS model is fast and accurate: it outperforms the state-of-the-art MinkowskiNet by 3.3%, ranking **1st** on the competitive SemanticKITTI leaderboard⋆. It also achieves **8-23**× computation reduction and **3**× measured speedup over MinkowskiNet and KPConv with higher accuracy. Finally, we transfer our method to 3D object detection, and it achieves consistent improvements over the one-stage detection baseline on KITTI.

1 Introduction

3D perception models have received increased attention as they serve as the eyes of autonomous driving systems: *i.e.*, they are used to understand the semantics of the scenes to parse the drivable area (*e.g.*, roads, parking areas). As the safety of the passenger is the top priority of the self-driving cars, 3D perception models are required to achieve *high accuracy and low latency*. Also, the hardware resources on the self-driving cars are tightly constrained by the form factor (since we do not want a whole trunk of workstations) and heat dissipation; therefore, it is crucial to design efficient 3D models with *low computational resource*, *e.g.*, memory.

Researchers have mainly exploited two 3D data representations: point cloud and rasterized voxel grids. As analyzed in Liu *et al.* [22], point-based methods [29, 32, 18] waste up to 90% of their time on structuring the irregular data,

∗ indicates equal contributions; order determined by a coin toss.

⋆ `https://competitions.codalab.org/competitions/20331#results`

(a) Large 3D Scene **(b)** Low Resolution (0.8m)

Fig. 1. Small instances (e.g., pedestrians and cyclists) are hard to be recognized at a low resolution (due to the coarse voxelization or the aggressive downsampling).

not on the actual feature extraction. On the other hand, voxel-based methods usually suffer from the low resolution: the resolution of dense voxels [25, 22] is strictly constrained by the memory; the sparse voxels [9, 6] require aggressive downsampling to achieve larger receptive field, leading to low resolution at deeper layers. With low resolution (see Figure 1), multiple points or even multiple small objects may be merged into one grid and become indistinguishable. In this case, small instances (*e.g.*, pedestrians and cyclists) are at a disadvantage compared to large objects (*e.g.*, cars). Therefore, the effectiveness of previous 3D modules is discounted when the hardware resource is limited and resolution is low.

To tackle these problems, we propose a new 3D module, *Sparse Point-Voxel Convolution (SPVConv)* that introduces a low-cost high-resolution point-based stream to the vanilla Sparse Convolution, which helps to capture the fine details. On top of the new SPVConv module, we further present *3D Neural Architecture Search (3D-NAS)* to search an efficient network architecture. We refer our whole framework as *Sparse Point-Voxel Neural Architecture Search (SPVNAS)*. Fine-grained channel numbers in the search space allow us to explore efficient models; progressive depth shrinking is introduced for training SPVNAS with elastic depth stably. Experimental results validate that our model is fast and accurate: compared to MinkowskiNet, it improves the accuracy by 3.3% with lower latency. It also achieves **8-23**× computation reduction and **3**× measured speedup over MinkowskiNet and KPConv, while offering higher accuracy. We also transfer our method to KITTI for 3D object detection and achieve consistent improvements over previous one-stage detection baseline.

The contribution of this paper has three aspects:

1. We design a lightweight 3D module, SPVConv, that pays attention to both local fine details and neighborhood relationship. It boosts the accuracy on small objects, which used to be challenging under limited hardware resource.
2. We boost the efficiency of the module by 3D-NAS: a fine-grained search space offers model efficiency balanced against accuracy; the progressive shrinking gets rid of re-training from scratch and reduces deployment complexity over various hardware platform and conditions.

3. Our method outperforms all previous methods with a large margin and ranks 1^{st} on the competitive SemanticKITTI leaderboard. It can also be transferred to the object detection task and achieve consistent improvements.

2 Related Work

2.1 3D Perception Models

Increased attention has been paid to 3D deep learning, which is important for LiDAR perception in autonomous driving. Previous research [5, 25, 31, 53, 61] relied on the volumetric representation and vanilla 3D convolution to process the 3D data. Due to the *sparse* nature of 3D representation, the *dense* volumetric representation is inherently inefficient and it also inevitably introduces information loss. Therefore, later research [29] proposes to directly learn on 3D point cloud representation using a symmetric function. To improve the neighborhood modeling capability, researchers define point-based convolutions on the geometric [18, 24, 32, 41, 44, 45, 55, 56] or semantic [52] neighborhood. There are also 3D models tailored for specific tasks such as detection [27, 28, 30, 36–38, 57, 59] and instance segmentation [11, 14, 15, 58] built upon these primitives.

Recently, some research started to pay attention to efficient 3D deep learning primitives. Riegler *et al*. [34], Wang *et al*. [49, 50] and Lei *et al*. [16] proposed to reduce the memory footprint of volumetric representation using octrees. Liu *et al*. [22] analyzed the efficiency bottleneck of point-based deep learning methods and proposed Point-Voxel Convolution. Graham *et al*. [9] and Choy *et al*. [6] proposed Sparse Convolution which accelerates the volumetric convolution by skipping non-activated regions.

2.2 Neural Architecture Search

Designing neural networks is highly challenging and time-consuming. To alleviate the burden of manually designing neural networks [13, 35, 23, 60], researchers have introduced neural architecture search (NAS) to automatically design the neural network with high accuracy using reinforcement learning [63, 64] and evolutionary search [19]. A new wave of research starts to design efficient models with neural architecture search [42, 54, 43], which is very important for the mobile deployment. However, conventional frameworks require high computation cost (typically 10^4 GPU hours) and considerable CO_2 emission [40]. To this end, researchers have proposed different techniques to reduce the computation cost (to 10^2 GPU hours), such as differentiable architecture search [20], path-level binarization [4], single-path one-shot sampling [10], and weight sharing [39, 2, 17, 46]. Besides, neural architecture search has also been used in compressing and accelerating neural networks, such as pruning [12, 21, 3] and quantization [47, 10, 48, 51]. Most of these methods are tailored for 2D visual recognition, which has many well-defined search spaces [33]. To the best of our knowledge, neural architecture search for 3D deep learning is under-studied. Previous research on VNAS [62] only focus on 3D medical image segmentation, which is not suitable for general-purpose 3D deep learning.

	Input	Voxel Size (m)	Latency (ms)	Mean IoU
PVConv [22]	Sliding Window	0.05	35640	–
	Entire Scene	0.78	146	39.0
SPVConv (Ours)	Entire Scene	0.05	**85**	**58.8**

Table 1. Point-Voxel Convolution [22] is not suitable for large 3D scenes. If processing with sliding windows, the large latency is not affordable for real-time applications. If taking the whole scene, the resolution is too coarse to capture useful information.

3 SPVConv: Designing Effective 3D Modules

We revisit two recent 3D modules: Point-Voxel Convolution [22] and Sparse Convolution [6] and analyze their bottlenecks. We observe that both of them suffer from information loss (caused by coarse voxelization or aggressive downsampling) when the memory is constrained. To this end, we introduce *Sparse Point-Voxel Convolution (SPVConv)*, to effectively process the large 3D scene (as in Figure 2).

3.1 Point-Voxel Convolution: Coarse Voxelization

Liu *et al*. [22] proposed Point-Voxel Convolution where the 3D input are represented in high-resolution points and convolution is applied over low-resolution voxel grids. Specifically, the point-based branch transforms each point individually, and the voxel-based branch convolves over the voxelized input from the point-based branch.

PVCNN (which is built upon Point-Voxel Convolution) can afford the resolution of at most 128 in its voxel-based branch on a single GPU (with 12 GB of memory). For a large outdoor scene (with size of 100m×100m×10m), each voxel grid will correspond to a fairly large area (with size of 0.8m×0.8m×0.1m). In this case, the small instances (*e.g.*, pedestrians) will only occupy a few voxel grids (see Figure 1). From such few points, PVCNN can hardly learn any useful information from the voxel-based branch, leading to a relatively low performance (see Table 1). Alternatively, we can process the large 3D scenes piece by piece so that each sliding window is of smaller scale. In order to preserve the fine-grained information, we found empirically that the voxel size needs to be at least lower than 0.05m. In this case, we have to run PVCNN once for each of the 244 sliding windows, which will take 35 seconds to process a single scene. Such a large latency is not affordable for most real-time applications (*e.g.*, autonomous driving).

3.2 Sparse Convolution: Aggressive Downsampling

Volumetric convolution has always been considered inefficient and prohibitive to be scaled up. Lately, researchers proposed Sparse Convolution [9, 6] that skips the non-activated regions to significantly reduce the memory consumption. More specifically, it first finds all active synapses (denoted as *kernel map*) between the input and output points; it then performs the convolution based on this kernel map. In order to keep the activation sparse, it only considers these output points that also belong to the input point cloud.

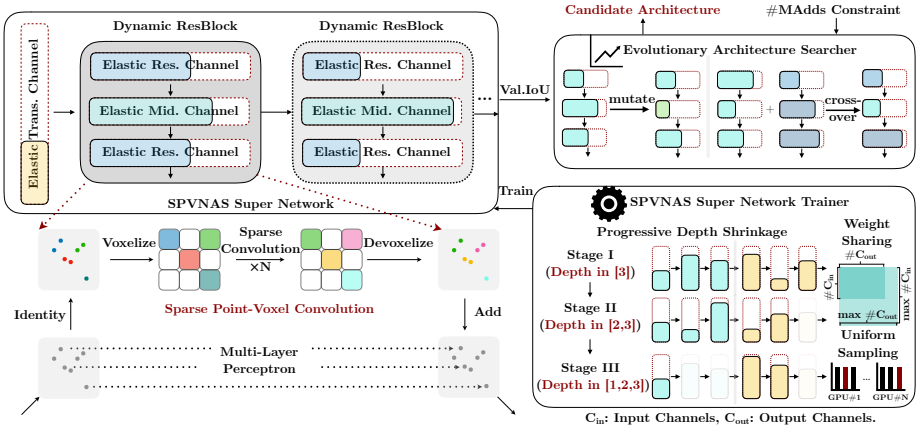


Fig. 2. Overview of SPVNAS: we first train a super network composed of SPVConv layers and supports elastic depth and width. Then, we perform computation-constrained 3D-NAS to obtain best candidate model.

As such, Sparse Convolution can afford a much higher resolution than the vanilla volumetric convolution. However, the network cannot be very deep due to the limited computation resource. As a result, the network has to downsample very aggressively in order to achieve a sufficiently large receptive field, which is very lossy. For example, the state-of-the-art MinkowskiNet [6] gradually applies four downsampling layers to the input point cloud, after which, the voxel size will be $0.05 \times 2^4 = 0.8$m. Similar to Point-Voxel Convolution, this resolution is too coarse to capture the small instances (see Figure 3).

3.3 Solution: Sparse Point-Voxel Convolution

In order to solve the problem of both modules, we present Sparse Point-Voxel Convolution as shown in Figure 2. The point-based feature transformation branch **always** keeps high-resolution representation. The voxel-based branch applies Sparse Convolution to efficiently model over different receptive field size. Two branches communicate at negligible cost through sparse voxelization and devoxelization.

Our Sparse Point-Voxel Convolution operates on:

- sparse voxelized tensor representation $\boldsymbol{S} = \{(\boldsymbol{p}^s_m, \boldsymbol{f}^s_m), v\}$, where $\boldsymbol{p}^s_m = (\boldsymbol{x}^s_m, \boldsymbol{y}^s_m, \boldsymbol{z}^s_m)$ is the grid coordinates and $\boldsymbol{f}^s_m$ is the grid feature vector of m-th nonzero grid, v is the voxel size for one grid in the current layer;
- point cloud tensor representation $\boldsymbol{T} = \{(\boldsymbol{p}^t_k, \boldsymbol{f}^t_k)\}$, where $\boldsymbol{p}_k = (\boldsymbol{x}_k, \boldsymbol{y}_k, \boldsymbol{z}_k)$ is the point coordinates and $\boldsymbol{f}_k$ is point feature vector of k-th point.

Sparse Voxelization. We start from introducing the voxel-based neighborhood aggregation branch in Figure 2. We first transform the high-resolution point

cloud representation $\boldsymbol{T}$ to a sparse tensor $\boldsymbol{S}$ by sparse voxelization:

$$\hat{\boldsymbol{p}}_k^t = (\hat{\boldsymbol{x}}_k^t, \hat{\boldsymbol{y}}_k^t, \hat{\boldsymbol{z}}_k^t) = (\text{floor}(\boldsymbol{x}_k^t/v), \text{floor}(\boldsymbol{y}_k^t/v), \text{floor}(\boldsymbol{z}_k^t/v)), \tag{1}$$

$$\boldsymbol{f}_m^s = \frac{1}{N_m} \sum_{k=1}^{n} \mathbb{I}[\hat{\boldsymbol{x}}_k^t = \boldsymbol{x}_m^s, \hat{\boldsymbol{y}}_k^t = \boldsymbol{y}_m^s, \hat{\boldsymbol{z}}_k^t = \boldsymbol{z}_m^s] \cdot \boldsymbol{f}_k^t, \tag{2}$$

where $\mathbb{I}[\cdot]$ is the binary indicator of whether $\hat{\boldsymbol{p}}_k^t$ belongs to the voxel grid $\boldsymbol{p}_m^s$, and N_m is the normalization factor (*i.e.*, the number of points that fall in the m-th nonzero voxel grid). Such formulation, however, requires $\mathcal{O}(mn)$ complexity where $|\boldsymbol{S}| = m, |\boldsymbol{T}| = n$. With typical values of m, n at the order of 10^5, the naive implementation is impractical for real time applications.

To this end, we propose to use the GPU hash table to accelerate the sparse voxelization and devoxelization. Specifically, we first build a hash table for all activated points in the sparse voxelized representation (where the key is the 3D coordinates, and the value is the index in the sparse voxelized tensor), which can be finished in $\mathcal{O}(n)$ time. After that, we iterate over all points, and for each point, we use its coordinate as the key to query the corresponding index in the sparse voxelized representation. As the lookup over the hash table requires $\mathcal{O}(1)$ time in the worst case [26], this query step will in total take $\mathcal{O}(m)$ time. Therefore, the total time of coordinate indexing will be reduced from $\mathcal{O}(mn)$ to $\mathcal{O}(m+n)$.

Feature Aggregation. We then perform neighborhood feature aggregation on the sparse voxelized tensor using a sequence of Sparse Convolution residual blocks [6]. We parallelize the kernel map operation in Sparse Convolution (see Section 3.2) on GPU with the same hash table implementation in sparse voxelization, which offers 1.3× speedup over Choy *et al.*'s latest implementation. Both our method and the baseline have been upgraded to this accelerated implementation.

Sparse Devoxelization. With transformed neighborhood features and a sparse tensor representation, we hope to transform it back to the point-based representation so that information from both branches can be fused later. Similar to [22], we choose to interpolate a point's feature with its 8 neighbor voxel grids using trilinear interpolation instead of naive nearest interpolation.

Point Transformation and Feature Fusion. We directly apply MLP on each point to extract individual point features, and then fuse the outputs of two branches with an addition to combine the complementary information provided. Compared against Sparse Convolution, MLP layers only cost little computation overhead (4% in terms of #MACs) but introduce important fine details into the information flow.

4 3D-NAS: Searching Efficient 3D Architectures

Even with our module, designing an efficient neural network is still challenging. We need to carefully adjust the network architecture (*e.g.*, channel numbers and kernel sizes of all layers) to meet the constraints for real-world applications (*e.g.*, latency, energy, and accuracy). To this end, we introduce *3D Neural Architecture Search (3D-NAS)*, to automatically design efficient 3D models (as in Figure 2).

4.1 Design Space

The performance of neural architecture search is greatly impacted by the design space quality. In our search space, we incorporate fine-grained channel numbers and elastic network depths; however, we do not support different kernel sizes.

Fine-grained Channel Numbers. The computation cost increases quadratically with the number of channels; therefore, the channel number selection has a large influence on the network efficiency. Most existing neural architecture frameworks [4] only support the coarse-grained channel number selection: *e.g.*, searching the expansion ratio of the ResNet/MobileNet blocks over a few (2-3) choices. In this case, only intermediate channel numbers of the blocks can be changed; while the input and output channel numbers will still remain the same. Empirically, we observe that this limits the variety of the search space. To this end, we enlarge the search space by allowing all channel numbers to be selected from a large collection of choices (with size of $O(n)$). This fine-grained channel number selection largely increase the number of candidates for each block: *e.g.*, from constant (2-3) to $\mathcal{O}(n^2)$ for a block with two consecutive convolutions.

Elastic Network Depths. We support different network depth in our design space. For 3D CNNs, reducing the channel numbers alone cannot achieve significant measured speedup, which is very different from the 2D CNNs. For example, by shrinking all channel numbers in MinkowskiNet [6] by $4\times$ and $8\times$, the number of MACs will be reduced to 7.5 G and 1.9 G, respectively. However, although #MACs is drastically reduced, their measured latency on the GPU is very similar: 105 ms and 96 ms (measured on a single GTX 1080Ti GPU). This suggests that scaling down the number of channels cannot offer us with very efficient models, even though the number of MACs is very small. This might be because 3D modules are usually more memory-bounded than 2D modules; #MACs decreases quadratically with channel number, while memory decreases linearly. Motivated by this, we choose to incorporate the elastic network depth into our design space so that these layers with very small computation (and large memory cost) can be removed and merged into their neighboring layers.

Small Kernel Matters. Kernel sizes are usually included into the search space of 2D CNNs. This is because a single convolution with larger kernel size can be more efficient than multiple convolutions with smaller kernel sizes on GPUs. However, it is not the case for the 3D CNNs. From the computation perspective, a single 2D convolution with kernel size of 5 requires only $1.4\times$ more MACs than two 2D convolutions with kernel sizes of 3; while a single 3D convolution with kernel size of 5 requires $2.3\times$ more MACs than two 3D convolutions with kernel sizes of 3 (if applied to dense voxel grids). This larger computation cost makes it less suitable to use large kernel sizes in 3D CNNs. Furthermore, the computation overhead of 3D modules is also related to the kernel sizes. For example, Sparse Convolution [9, 6] requires $\mathcal{O}(k^3 n)$ time to build the kernel map, where k is the kernel size and n is the number of points, which indicates that its cost grows cubically with respect to the kernel size. Based on these reasons, we decide to keep the kernel size of all convolutions to be 3 and do not allow the kernel size to

change in our search space. Even with the small kernel size, we can still achieve a large receptive field by changing the network depth, which can achieve the same effect as changing the kernel size.

4.2 Training Paradigm

Searching over a fine-grained design space is very challenging as it is impossible to train every sampled candidate network from scratch [42]. Motivated by Guo *et al*. [10], we incorporate all candidate networks into a single super network so that the total training cost can be reduced from $\mathcal{O}(n)$ to $\mathcal{O}(1)$: we train the super network once, and after that, each candidate network can be directly extracted from this super network with inherited weights.

Uniform Sampling. At each training iteration, we randomly sample a candidate network from the super network: randomly select the channel number for each layer, and then randomly select the network depth (*i.e*. the number of blocks to be used) for each stage. The total number of candidate networks to be sampled during training is very limited; therefore, we choose to sample different candidate networks on different GPUs and average their gradients at each step so that more candidate networks can be sampled. For 3D, this is more critical because the 3D datasets usually contain fewer training samples than the 2D datasets: *e.g*. 20K on SemanticKITTI [1] *vs*. 1M on ImageNet [7].

Weight Sharing. As the total number of candidate networks is enormous, every candidate network will only be optimized for a small fraction of the total schedule. Therefore, uniform sampling alone is not enough to train all candidate networks sufficiently (*i.e*., achieving the same level of accuracy as being trained from scratch). To this end, we adopt the weight sharing technique so that every candidate network can be optimized at each iteration even if it is not sampled. Specifically, given the input channel number C_{in} and output channel number C_{out} of each convolution layer, we simply index the first C_{in} and C_{out} channels from the weight tensor accordingly to perform the convolution [10]. For each batch normalization layer, we similarly crop the first c channels from the weight tensor based on the sampled channel number c. Finally, with the sampled depth d for each stage, we choose to keep the first d layers, instead of randomly sampling d of them. This ensures that each layer will always correspond to the same depth index within the stage.

Progressive Depth Shrinking. Suppose we have n stages, each of which has m different depth choices from 1 to m. If we sample the depth d_k for each stage k randomly, the expected total depth of the network will be

$$\mathbb{E}[d] = \sum_{k=1}^{n} \mathbb{E}[d_k] = n \times \frac{m+1}{2}, \tag{3}$$

which is much smaller than the maximum depth nm. Furthermore, the probability of the largest candidate network (with the maximum depth) being sampled is extremely small: m^{-n}. Therefore, the largest candidate networks are poorly trained due to the small possibility of being sampled. To this end, we introduce

progressively shrinking the depth to alleviate this issue. We divide the training epochs into m segments for m different depth choices. During the k^{th} training segment, we only allow the depth of each stage to be selected from $m-k+1$ to m. This is essentially designed to enlarge the search space gradually so that these large candidate networks can be sampled more frequently.

4.3 Search Algorithm

After the super network is fully trained, we use evolutionary architecture search to find the best architectures under a certain resource constraint.

Resource Constraints. We use the number of MACs as the resource constraint. For the 3D CNNs, the number of MACs cannot be simply determined by the input size and network architecture: *e.g.*, Sparse Convolution only performs the computation over the active synapses; therefore, its computation is also related to the kernel map size, which is determined by the input sparsity pattern. To address this, we first estimate the average kernel map size over the entire dataset for each convolution layer, and we can then measure the number of MACs based on these statistics.

Evolutionary Search. We automate the architecture search with evolutionary algorithm [10]. We first initialize the starting population with n randomly sampled candidate networks. At each iteration, we evaluate all candidate networks in the population and select the k models with the highest accuracies (*i.e.*, the fittest individuals). The population for the next iteration is then generated with $(n/2)$ mutations and $(n/2)$ crossovers. For each mutation, we randomly pick one among the top-k candidates and alter each of its architectural parameters (*e.g.*, channel numbers, network depths) with a pre-defined probability; for each crossover, we select two from the top-k candidates and generate a new model by fusing them together randomly. Finally, the best model is obtained from the population of the last iteration. During the evolutionary search, we ensure that all the candidate networks in the population always meet the given resource constraint (otherwise, we will resample another candidate network until the resource constraint is satisfied).

5 Experiments

We conduct experiments on 3D semantic segmentation and 3D object detection for outdoor scenes. Benefit from our designed module (SPVConv) and neural architecture search framework (3D-NAS), our model (denoted as SPVNAS) consistently outperforms previous state-of-the-art methods with lower computation cost and measured latency (on an NVIDIA GTX1080Ti).

5.1 3D Scene Segmentation

We first evaluate our method on 3D semantic segmentation and conduct experiments on the large-scale outdoor scene dataset, SemanticKITTI [1]. This dataset contains 23,201 LiDAR point clouds for training and 20,351 for testing, and it is annotated from all 22 sequences in the KITTI [8] Odometry benchmark. We train all models on the entire training set and report the mean intersection-over-union

	#Params (M)	#MACs (G)	Latency (ms)	Mean IoU
PointNet [29]	3.0	–	500	14.6
PointNet++ [32]	6.0	–	5900	20.1
PVCNN [22]	2.5	42.4	146	39.0
KPConv [45]	18.3	207.3	279	58.8
MinkowskiNet [6]	21.7	114.0	294	63.1
SPVNAS (Ours)	2.6	15.0	110	63.7
	12.5	73.8	259	**66.4**

Table 2. Results of outdoor scene segmentation on SemanticKITTI: our SPVNAS outperforms the state-of-the-art MinkowskiNet with **2.7**× measured speedup.

	#Params (M)	#MACs (G)	Latency (ms)	Mean IoU
DarkNet21Seg [1]	24.7	212.6	73	47.4
DarkNet53Seg [1]	50.4	376.3	102	49.9
SPVNAS (Ours)	1.1	8.9	89	**60.3**

Table 3. Results of outdoor scene segmentation on SemanticKITTI: our SPVNAS outperforms the 2D projection-based DarkNets by more than **10%** in mIoU.

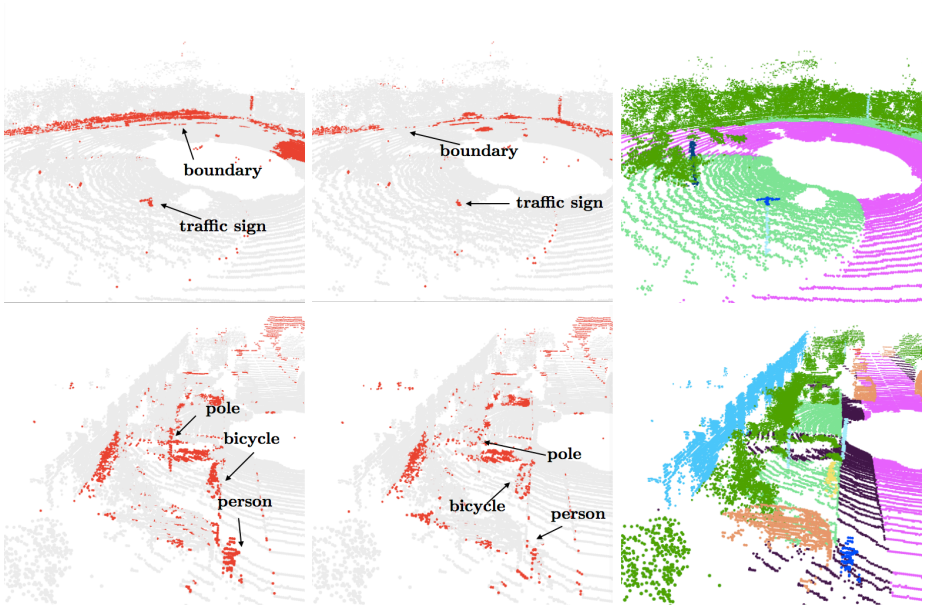
(mIoU) on the official test set under the single scan setting. We provide more implementation details and experimental results in the appendix.

Results. As in Table 2, our SPVNAS outperforms the previous state-of-the-art MinkowskiNet [6] by **3.3%** in mIoU with 1.7× model size reduction, 1.5× computation reduction and 1.1× measured speedup. We further downscale our SPVNAS by setting the resource constraint to 15G MACs. This offers us with a much smaller model that outperforms MinkowskiNet by 0.6% in mIoU with **8.3**× model size reduction, **7.6**× computation reduction, and **2.7**× measured speedup. In Figure 3, we also provide some qualitative comparisons between SPVNAS and MinkowskiNet: our SPVNAS has lower errors especially for small instances.

We further compare our SPVNAS with 2D projection-based models in Table 3. With the smaller backbone (by removing the decoder layers), SPVNAS outperforms DarkNets [1] by more than **10%** in mIoU with 1.2× measured speedup even though 2D convolutions are much better optimized by modern deep learning libraries. Furthermore, our SPVNAS achieves higher mIoU than KPConv [45], which is the previous state-of-the-art point-based model, with **17**× model size reduction, **23**× computation reduction and **3**× measured speedup.

5.2 3D Object Detection

We also evaluate our method on 3D object detection and conduct experiments on the popular outdoor scene dataset, KITTI [8]. We follow the generally adopted training-validation split, where 3,712 samples are used for training and 3,769 samples are left for validation. We report the mean average precision on the test



(a) Error by MinkowskiNet **(b)** Less error by SPVNAS **(c)** Ground Truth

Fig. 3. MinkowskiNet has a higher error recognizing small objects and region boundaries, while SPVNAS recognizes small objects better thanks to the high-resolution point-based branch.

split using the official evaluation code (with 40 recall positions) under 3D IoU theresholds of 0.7 for car, 0.5 for cyclist and pedestrian. We refer the readers to the appendix for additional results on the validation set.

Results. We compare our method against SECOND [57], the state-of-the-art single-stage model for 3D object detection. SECOND consists of a sparse encoder using 3D Sparse Convolutions and a region proposal network that performs 2D convolutions after projecting the encoded features to the bird's-eye view (BEV). We reimplement and retrain SECOND: our implementation already outperforms the results in the original paper [57]. As for our model, we only replace the 3D Sparse Convolutions in SECOND with our SPVConv while keeping all the other settings the same for fair comparison. As summarized in Table 4, our SPVCNN achieves significant improvement in cyclist detection, for which we argue that the high-resolution point-based branch carries more information for small instances.

6 Analysis

Our SPVNAS significantly outperforms the previous state of the art, MinkowskiNets with better efficiency. After carefully examining the per-class performance of both methods on the test split (Table 5), we find that SPVNAS has very large advantage (up to **25%**) on relatively small objects such as pedestrians and

	Car			Cyclist			Pedestrian		
	Easy	Mod.	Hard	Easy	Mod.	Hard	Easy	Mod.	Hard
SECOND [57]	84.7	76.0	68.7	75.8	60.8	53.7	45.3	35.5	33.1
SECOND (Repro.)	87.5	77.9	74.4	76.0	59.7	52.9	49.1	**41.7**	**39.1**
SPVCNN (Ours)	**87.8**	**78.4**	**74.8**	**80.1**	**63.7**	**56.2**	**49.2**	41.4	38.4

Table 4. Results of outdoor object detection on KITTI: our SPVCNN outperforms SECOND in most categories especially for the cyclist.

	Person	Bicycle	Bicyclist	Motorcycle	Motorcyclist
MinkowskiNet [6]	60.9	40.4	61.9	47.4	18.7
SPVNAS (Ours)	65.7 (+**4.8**)	51.6 (+**11.2**)	65.2 (+**3.3**)	50.8 (+**3.4**)	43.7 (+**25.0**)

Table 5. Results of per-class performance on SemanticKITTI: SPVNAS has a large advantage on small objects, such as bicyclist and motorcyclist.

cyclists, which justifies our design of a high resolution point-based branch in SPVConv. In this section, we provide more detailed analysis on the effectiveness of SPVConv and also perform ablation experiments on our 3D-NAS pipeline to further explain the benefit of SPVNAS.

6.1 Sparse Point-Voxel Convolution

We analyze the effectiveness of SPVConv by comparing the *point* and *sparse-voxel* activations from the last SPVConv layer in SPVCNN. The model is trained on part of SemanticKITTI [1] training set with the ninth sequence left out for visualization. Specifically, we first calculate the norm of point/sparse voxel features from each point. Then, we rank the feature norms from both branches separately and define points with top 10% largest feature norm from each branch respectively as **activated** points of that branch. In Figure 5 we show the top 50% activated points of the point-based branch with red color and all other points with gray color. Clearly, the point branch of our SPVCNN learns to attend to small objects such as pedestrians, cyclists, trunks and traffic signs. As a result, our method does achieve compelling performance on these small classes.

We also collect the statistics of class-wise averaged percentage for activated points from both point-based and sparse voxel-based branch in Figure 4. On small objects, the percentage of activated points from the point-based branch is significantly higher than the sparse voxel-based branch. For some classes like the bicyclist, more than **80%** of its points are activated on the point branch, which validates that our observation in Figure 5 is general.

6.2 Architecture Search

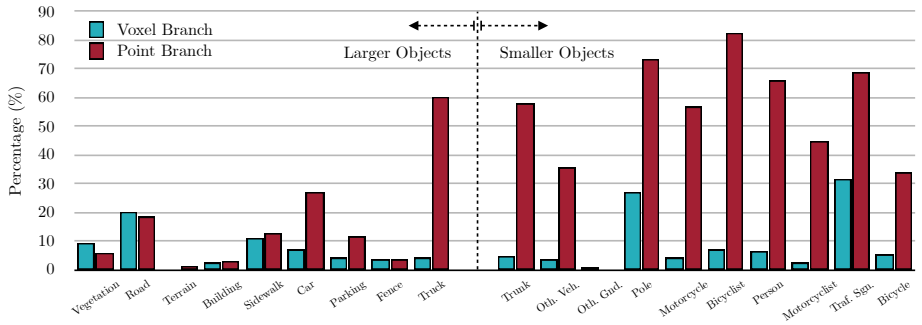


Fig. 4. Average percent of activated points on voxel/point branches from all 19 classes of SemanticKITTI [1] dataset: the point-branch attends to smaller objects as the red bar is much higher.

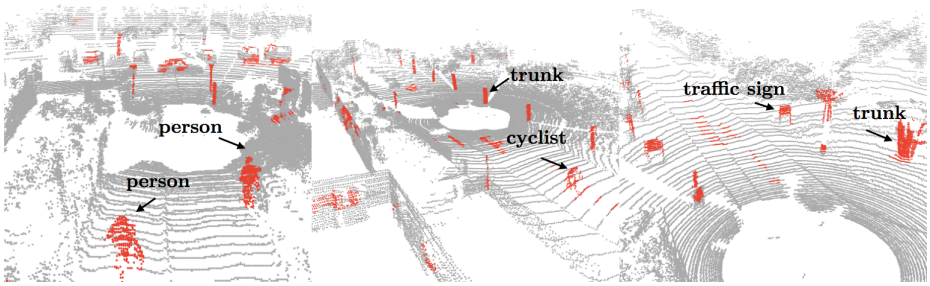


Fig. 5. The point-based branch learns to put its attention on small instances (*i.e.*, pedestrians, cyclists, traffic signs). Here, the points in red are the ones with the top 5% largest feature norm in the point-based branch.

In Figure 6 we show the MACs (Figure 6a) and latency (Figure 6b) tradeoff curves on SemanticKITTI [1]. Manually designed SPVCNN and MinkowskiNets with uniform channel shrinking are the baselines. Clearly, a better 3D module (SPVConv) and a well-designed network architecture contribute equally to the performance boost. Remarkably, the improvement over MinkowskiNets exceeds **6%** mIoU at 110 ms latency. We believe the improvement comes from the non-uniform channel scaling and depth selection in our 3D-NAS. In the original MinkowskiNets [6]

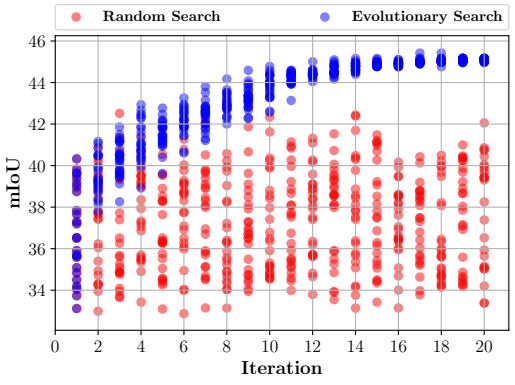
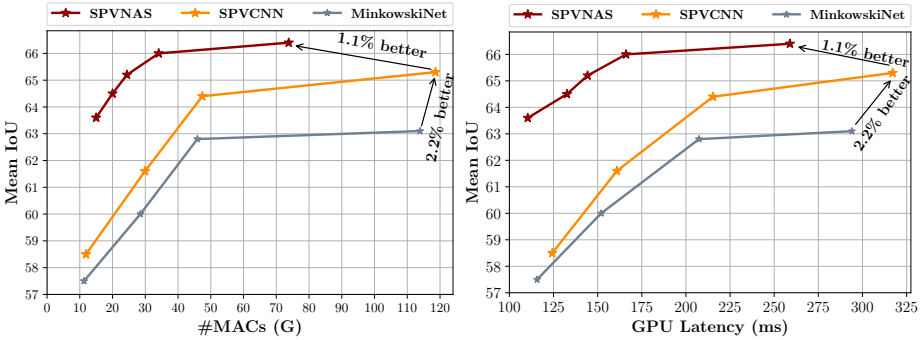


Fig. 7. Evolutionary Search has better sample efficiency comparing with Random Search.



(a) #MACs *v.s.* IoU Tradeoff **(b)** Latency *v.s.* IoU Tradeoff

Fig. 6. An efficient 3D module (SPVConv) and a well-designed network architecture (3D-NAS) are equally important to the final performance of our SPVNAS.

or SPVCNN, 77% of the total MACs is concentrated on the upsampling stages. However, this ratio is reduced to 47% to 63% in 3D-NAS, making computation more balanced and downsampling stages more emphasized.

We also compare our evolutionary search method with random architecture search to prove that the succeed of 3D-NAS doesn't entirely come from the search space. As is shown in Figure 7, random architecture search has poor sample efficiency in our search space: the best model at the 20^{th} generation performs even worse than the best model in the 4^{th} generation. In contrast, our evolutionary search is capable of progressively finding better architecture as iteration increases, and the final best architecture performs around 4% better than the best one in the first generation.

7 Conclusion

We present Sparse Point-Voxel Convolution (SPVConv), a novel module for efficient 3D deep learning, especially for small object recognition. With SPVCNN built upon the SPVConv module, we solve the problem that Sparse Convolution cannot always keep high resolution representation and that Point-Voxel Convolution doesn't scale up to large outdoor scenes. We then propose 3D-NAS, the first AutoML method for 3D scene understanding, to greatly improve the efficiency and performance of SPVCNN. Extensive experiments on outdoor 3D scene benchmarks demonstrates that SPVNAS models are lightweight, fast and powerful. We hope that this work will inspire more future research on efficient 3D deep learning model design.

Acknowledgements. We thank MIT Quest for Intelligence, MIT-IBM Watson AI Lab, Xilinx, Samsung for supporting this research. We also thank AWS Machine Learning Research Awards for providing the computational resource.

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: ICCV (2019)
2. Cai, H., Gan, C., Wang, T., Zhang, Z., Han, S.: Once for All: Train One Network and Specialize it for Efficient Deployment. In: ICLR (2020)
3. Cai, H., Lin, J., Lin, Y., Liu, Z., Wang, K., Wang, T., Zhu, L., Han, S.: AutoML for Architecting Efficient and Specialized Neural Networks. IEEE Micro (2019)
4. Cai, H., Zhu, L., Han, S.: ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. In: ICLR (2019)
5. Chang, A.X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., Xiao, J., Yi, L., Yu, F.: ShapeNet: An Information-Rich 3D Model Repository. arXiv (2015)
6. Choy, C., Gwak, J., Savarese, S.: 4D Spatio-Temporal ConvNets: Minkowski Convolutional Neural Networks. In: CVPR (2019)
7. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: ImageNet: A Large-Scale Hierarchical Image Database. In: CVPR (2009)
8. Geiger, A., Lenz, P., Stiller, C., Urtasun, R.: Vision meets Robotics: The KITTI Dataset. IJRR (2013)
9. Graham, B., Engelcke, M., van der Maaten, L.: 3D Semantic Segmentation With Submanifold Sparse Convolutional Networks. In: CVPR (2018)
10. Guo, Z., Zhang, X., Mu, H., Heng, W., Liu, Z., Wei, Y., Sun, J.: Single Path One-Shot Neural Architecture Search with Uniform Sampling. In: ECCV (2020)
11. Han, L., Zheng, T., Xu, L., Fang, L.: OccuSeg: Occupancy-aware 3D Instance Segmentation. In: CVPR (2020)
12. He, Y., Lin, J., Liu, Z., Wang, H., Li, L.J., Han, S.: AMC: AutoML for Model Compression and Acceleration on Mobile Devices. In: ECCV (2018)
13. 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 (2017)
14. Jiang, L., Zhao, H., Shi, S., Liu, S., Fu, C.W., Jia, J.: PointGroup: Dual-Set Point Grouping for 3D Instance Segmentation. In: CVPR (2020)
15. Lahoud, J., Ghanem, B., Pollefeys, M., Oswald, M.R.: 3D Instance Segmentation via Multi-Task Metric Learning. In: ICCV (2019)
16. Lei, H., Akhtar, N., Mian, A.: Octree Guided CNN With Spherical Kernels for 3D Point Clouds. In: CVPR (2019)
17. Li, M., Lin, J., Ding, Y., Liu, Z., Zhu, J.Y., Han, S.: GAN Compression: Efficient Architectures for Interactive Conditional GANs. In: CVPR (2020)
18. Li, Y., Bu, R., Sun, M., Wu, W., Di, X., Chen, B.: PointCNN: Convolution on $\mathcal{X}$-Transformed Points. In: NeurIPS (2018)
19. Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.J., Fei-Fei, L., Yuille, A., Huang, J., Murphy, K.: Progressive Neural Architecture Search. In: ECCV (2018)
20. Liu, H., Simonyan, K., Yang, Y.: DARTS: Differentiable Architecture Search. In: ICLR (2019)
21. Liu, Z., Mu, H., Zhang, X., Guo, Z., Yang, X., Cheng, K.T., Sun, J.: MetaPruning: Meta Learning for Automatic Neural Network Channel Pruning. In: ICCV (2019)
22. Liu, Z., Tang, H., Lin, Y., Han, S.: Point-Voxel CNN for Efficient 3D Deep Learning. In: NeurIPS (2019)
23. Ma, N., Zhang, X., Zheng, H.T., Sun, J.: ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design. In: ECCV (2018)

24. Mao, J., Wang, X., Li, H.: Interpolated Convolutional Networks for 3D Point Cloud Understanding. In: ICCV (2019)
25. Maturana, D., Scherer, S.: VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition. In: IROS (2015)
26. Pagh, R., Rodler, F.F.: Cuckoo Hashing. Journal of Algorithms (2001)
27. Qi, C.R., Chen, X., Litany, O., Guibas, L.J.: ImVoteNet: Boosting 3D Object Detection in Point Clouds with Image Votes. In: CVPR (2020)
28. Qi, C.R., Litany, O., He, K., Guibas, L.J.: Deep Hough Voting for 3D Object Detection in Point Clouds. In: ICCV (2019)
29. Qi, C.R., Su, H., Mo, K., Guibas, L.J.: PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In: CVPR (2017)
30. Qi, C.R., Liu, W., Wu, C., Su, H., Guibas, L.J.: Frustum PointNets for 3D Object Detection from RGB-D Data. In: CVPR (2018)
31. Qi, C.R., Su, H., Niessner, M., Dai, A., Yan, M., Guibas, L.J.: Volumetric and Multi-View CNNs for Object Classification on 3D Data. In: CVPR (2016)
32. Qi, C.R., Yi, L., Su, H., Guibas, L.J.: PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. In: NeurIPS (2017)
33. Radosavovic, I., Johnson, J., Xie, S., Lo, W.Y., Dollar, P.: On Network Design Spaces for Visual Recognition. In: ICCV (2019)
34. Riegler, G., Ulusoy, A.O., Geiger, A.: OctNet: Learning Deep 3D Representations at High Resolutions. In: CVPR (2017)
35. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.C.: MobileNetV2: Inverted Residuals and Linear Bottlenecks. In: CVPR (2018)
36. Shi, S., Guo, C., Jiang, L., Wang, Z., Shi, J., Wang, X., Li, H.: PV-RCNN: Point-Voxel Feature Set Abstraction for 3D Object Detection. In: CVPR (2020)
37. Shi, S., Wang, X., Li, H.: PointRCNN: 3D Object Proposal Generation and Detection from Point Cloud. In: CVPR (2019)
38. Shi, S., Wang, Z., Shi, J., Wang, X., Li, H.: PV-RCNN: Point-Voxel Feature Set Abstraction for 3D Object Detection. TPAMI (2020)
39. Stamoulis, D., Ding, R., Wang, D., Lymberopoulos, D., Priyantha, B., Liu, J., Marculescu, D.: Single-Path NAS: Designing Hardware-Efficient ConvNets in less than 4 Hours. arXiv (2019)
40. Strubell, E., Ganesh, A., McCallum, A.: Energy and Policy Considerations for Deep Learning in NLP. In: ACL (2019)
41. Su, H., Jampani, V., Sun, D., Maji, S., Kalogerakis, E., Yang, M.H., Kautz, J.: SPLATNet: Sparse Lattice Networks for Point Cloud Processing. In: CVPR (2018)
42. Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., Le, Q.V.: MnasNet: Platform-Aware Neural Architecture Search for Mobile. In: CVPR (2019)
43. Tan, M., Le, Q.V.: EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In: ICML (2019)
44. Tatarchenko, M., Park, J., Koltun, V., Zhou, Q.Y.: Tangent Convolutions for Dense Prediction in 3D. In: CVPR (2018)
45. Thomas, H., Qi, C.R., Deschaud, J.E., Marcotegui, B., Goulette, F., Guibas, L.J.: KPConv: Flexible and Deformable Convolution for Point Clouds. In: ICCV (2019)
46. Wang, H., Wu, Z., Liu, Z., Cai, H., Zhu, L., Gan, C., Han, S.: HAT: Hardware-Aware Transformers for Efficient Natural Language Processing. In: ACL (2020)
47. Wang, K., Liu, Z., Lin, Y., Lin, J., Han, S.: HAQ: Hardware-Aware Automated Quantization with Mixed Precision. In: CVPR (2019)
48. Wang, K., Liu, Z., Lin, Y., Lin, J., Han, S.: Hardware-Centric AutoML for Mixed-Precision Quantization. IJCV (2020)

49. Wang, P.S., Liu, Y., Guo, Y.X., Sun, C.Y., Tong, X.: O-CNN: Octree-based Convolutional Neural Networks for 3D Shape Analysis. In: SIGGRAPH (2017)
50. Wang, P.S., Liu, Y., Guo, Y.X., Sun, C.Y., Tong, X.: Adaptive O-CNN: A Patch-based Deep Representation of 3D Shapes. In: SIGGRAPH Asia (2018)
51. Wang, T., Wang, K., Cai, H., Lin, J., Liu, Z., Wang, H., Lin, Y., Han, S.: APQ: Joint Search for Network Architecture, Pruning and Quantization Policy. In: CVPR (2020)
52. Wang, Y., Sun, Y., Liu, Z., Sarma, S.E., Bronstein, M.M., Solomon, J.M.: Dynamic Graph CNN for Learning on Point Clouds. In: SIGGRAPH (2019)
53. Wang, Z., Lu, F.: VoxSegNet: Volumetric CNNs for Semantic Part Segmentation of 3D Shapes. TVCG (2019)
54. 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: CVPR (2019)
55. Wu, W., Qi, Z., Fuxin, L.: PointConv: Deep Convolutional Networks on 3D Point Clouds. In: CVPR (2019)
56. Xu, Y., Fan, T., Xu, M., Zeng, L., Qiao, Y.: SpiderCNN: Deep Learning on Point Sets with Parameterized Convolutional Filters. In: ECCV (2018)
57. Yan, Y., Mao, Y., Li, B.: SECOND: Sparsely Embedded Convolutional Detection. Sensors (2018)
58. Yang, B., Wang, J., Clark, R., Hu, Q., Wang, S., Markham, A., Trigoni, N.: Learning Object Bounding Boxes for 3D Instance Segmentation on Point Clouds. In: NeurIPS (2019)
59. Yang, Z., Sun, Y., Liu, S., Shen, X., Jia, J.: STD: Sparse-to-Dense 3D Object Detector for Point Cloud. In: ICCV (2019)
60. Zhang, X., Zhou, X., Lin, M., Sun, J.: ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. In: CVPR (2018)
61. Zhou, Y., Tuzel, O.: VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection. In: CVPR (2018)
62. Zhu, Z., Liu, C., Yang, D., Yuille, A., Xu, D.: V-NAS: Neural Architecture Search for Volumetric Medical Image Segmentation. In: 3DV (2019)
63. Zoph, B., Le, Q.V.: Neural Architecture Search with Reinforcement Learning. In: ICLR (2017)
64. Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V.: Learning Transferable Architectures for Scalable Image Recognition. In: CVPR (2018)