# Searching Efficient Dynamic Graph CNN for Point Cloud Processing

**Panyue Chen**[1] **Rui Wang**[1] **Ping Zhao**[1] **Guanming Liu**[1] **Zhihua Wei**[1]✉

[1]Department of Computer Science and Technology, Tongji University

**Abstract** Despite superior performance on various point cloud processing tasks, convolutional neural networks (CNN) are challenged by deploying on resource-constraint devices such as cars and cellphones. Most existing convolution variants, such as dynamic graph CNN (DGCNN), require elaborately manual design and scaling-up across various constraints to accommodate multiple hardware deployments. It results in a massive amount of computation and limits the further application of these models. To this end, we propose a one-shot neural architecture search method for point cloud processing to achieve efficient inference and storage across various constraints. We conduct our method with DGCNN to create a compressed model. Extensive experiments on the point cloud classification and part segmentation tasks strongly evidence the benefits of the proposed method. Compared with the original network, we achieve 17.5× computation saving on the classification task with the comparable performance and obtain a 2.7× model compression ratio on the part segmentation task with slight IoU loss. Our code is available at `https://github.com/razIove/NAS-DGCNN`

## 1 Introduction

Point clouds provide rich geometric, shape, and scale information for a better understanding of the surrounding environment for machines. This promotes numerous applications in different areas, including autonomous driving, robotics, and remote sensing. In recent years, various convolutional neural networks (CNNs) have been proposed to process point clouds by either projecting three-dimensional scenes and objects onto two-dimensional planes (Su et al., 2015; Lawin et al., 2017; Yu et al., 2018), employing voxel convolution (Maturana and Scherer, 2015; Wu et al., 2015), or building feature extractor to directly process the raw point cloud format(Qi et al., 2017a,b). Most of the existing point cloud models (Qi et al., 2017b; Wang et al., 2019; Wu et al., 2019; Thomas et al., 2019; Zhou et al., 2021) are designed with complicated feature extractors and network architectures which causes high costs on computing and storage resources. The computational bottleneck prevents these models from being deployed on various resource-limited devices. There are several common techniques for compressing a model, e.g., neural architecture search (NAS), which enables scaling the network during training using an automatic architecture design method (Tang et al., 2020). However, applying NAS to point cloud processing suffers from two problems: (1) most of the NAS methods (Liu et al., 2018; Cai et al., 2019, 2020; Chen et al., 2021) are designed for 2D visual recognition and ignore point cloud operations such as point grouping, which limits their application in point cloud processing; (2) some NAS methods for point cloud processing (Nie et al., 2021; Lin et al., 2021) aim to achieve higher accuracy but neglect the various requirements of computational resources. How to discover a compressed network for point cloud processing that can take advantage of both point grouping and model size to satisfy various resource constraints is not well explored.

In this work, we propose a novel one-shot NAS method for efficient point cloud processing. Specifically, we first propose a point cloud supernet that introduces dynamic neighborhood points sampling to enable scaling in both point grouping and architectural settings, which helps search for important points and reduce the amount of computation. Then a progressive distillation strategy

 © 2022 the authors, released under CC BY 4.0

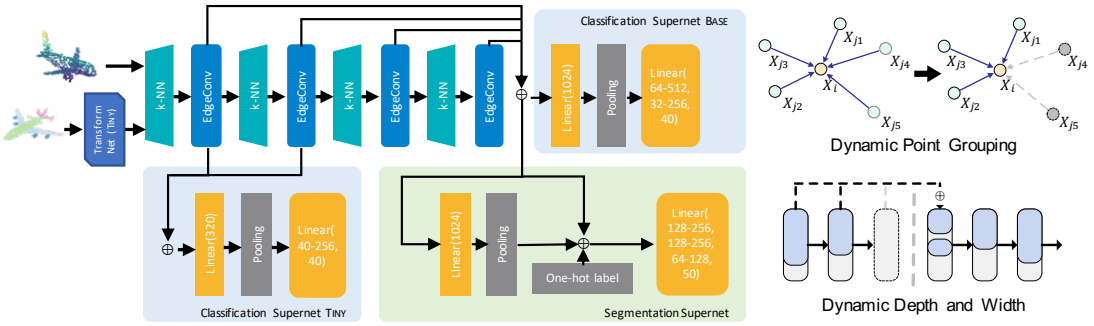


Figure 1: Overall search space. There are three supernets for different model sizes and point cloud processing tasks, e.g., classification supernet TINY, classification supernet BASE, and segmentation supernet. There are three searchable spaces, e.g., dynamic point grouping as well as dynamic depth and width. The gray blocks and Transformer Net (TINY) are not dynamic, and the others are of scaling size in either input or output.

is proposed to optimize subnets that share weights, which can reduce the interference between them. To discover a subnet with a specific resource constraint, we utilize a simple random search to obtain a satisfactory architecture without re-training. We conduct our method with dynamic graph CNN to create a compressed model. Experiments on the ModelNet40 dataset (Wu et al., 2015) and ShapeNetPart dataset (Yi et al., 2016) show that the proposed method obtains a favorable trade-off between accuracy and computation. Our method achieve 17.5× computation saving on the classification task while maintaining comparable performance. Besides, we achieve a 2.7× model compression ratio on the part segmentation with slight IoU loss. We further provide an empirical comparison of the training stages and modules and conduct an ablation study to demonstrate the effectiveness of the proposed progressive training strategy and knowledge distillation. The contributions of this paper are summarized as follows.

- To the best of our knowledge, this is the first work to investigate a point clouds neural network search framework that jointly adjusts the point grouping and network architecture for compressing raw point clouds models.
- We propose a multiple-dimensional weight-sharing search space that enables architecture search with different model sizes regarding point grouping, network depths, and channel widths, which is essential to quickly discover the specific network with the given resource requirements.
- Our method achieves higher accuracy and smaller size than the original network. It achieves up to 17.5× computation saving on the classification task without compromising test accuracy and obtains a 2.7× model compression ratio on the part segmentation task with 1% IoU loss.

## 2 3D Neural Architecture Search

To apply network architecture search to point cloud CNNs, we modify the existing 2D image-based NAS method and propose a novel design space and training strategy. We establish a point cloud network based on DGCNN (Wang et al., 2019). DGCNN uses EdgeConv blocks to encode point cloud features. Each EdgeConv block contains a $k$-NN layer, a linear layer, and a pooling layer. Instead of modifying the EdgeConv block, we dynamically scale the size of the neural architecture. The overview of our approach is shown in Figure 1. We design three supernets with different sizes for classification and segmentation tasks: classification supernet TINY, classification supernet BASE, and segmentation supernet. The size of classification supernet BASE and segmentation supernet is the same as DGCNN. These supernets contain three kinds of search space: point grouping, network depths, and channel numbers.

### 2.1 Design Space

The proper design space quality can significantly reduce redundancy in the network. Our search space provides dynamic scaling of point aggregation and architectural settings, including depth and width. Specifically, dynamic point grouping is proposed jointly with network depths and channel numbers to adjust sampled points and scale model size. The details of our search spaces can be found in Appendix.

**Dynamic Point Grouping**. The size of the nearest neighbors in the $k$-nearest neighbor ($k$-NN) algorithm is essential to represent the point cloud. Previous point cloud networks are proposed to dynamically calculate the nearest neighbors to obtain dynamic graphs with the fixed number $k$ at each layer. However, determining $k$ at each layer is not well explored. We propose dynamic point grouping to support dynamic adjustment of $k$ to help network learning point representation with variable-size dynamic graphs. Dynamically adjusting $k$ can reduce the amount of computation and allow the network to learn the essential information. Specifically, we randomly sample $k$ in the search space for each block containing $k$-NN in each forward propagation during training, making the value of $k$ different between blocks.

**Dynamic Network Depths and Channel Numbers**. Making network depths scalable and reducing the number of channels can produce a compact network. In our design space, the weights of the selected blocks or channels are shared between architectures of different sizes. Specifically, we keep the number of sampled blocks from the front of the encoding part instead of skipping blocks randomly when we adjust the network depth. For channel numbers, the sub-network uses the weights of the sampled channel numbers from the front of the weight matrix, which is extracted from the supernet. The manner of scaling depth and with can make the model training more stable and facilitates various-resolution feature encoding and aggregation. On the other hand, decreasing network depths may result in performance loss in the point cloud processing tasks, especially for a shallow network. Thus, we only adjust the number of encoding blocks for the classification network.

### 2.2 Training and Search

We train the network using progressive staged training and knowledge distillation. Then, we employ a random search to sample subnets and evaluate them without fine-tuning or re-training.

**Progressive Training**. It is impractical to train millions of networks from scratch. We implement a one-shot NAS method that allows all networks to share parameter weights, e.g., the small subnets are nested in large ones. It causes interference between subnets with different sizes because shared weights between subnets have to play different roles. To alleviate interference between different subnets, we train the networks progressively, in sequence, from large to small subnets. Specifically, we first train an entire network from scratch. Subsequently, we randomly sample a subnet based on the specific search space at different stages in each training iteration. For the classification network: (1) randomly select the network depth (i.e., the number of encoding blocks); (2) randomly select the network depth, the encoding channels, and $k$; (3) randomly select the network depth, the encoding channels, $k$, and the decoding channels. For the segmentation network, only the (2) and (3) stages are sequentially used in training. The empirical results suggest that the proposed progressive training strategy makes all subnets well trained.

**Knowledge Distillation**. To obtain higher accuracy, we repeat the training procedure twice. The best network and weights obtained in the first training are used as a teacher model to improve subnets by knowledge distillation. The output of the teacher network is used as a soft label to guide the model training through cross entropy loss.

Table 1: Results comparison with state-of-the-art architectures. NAS-DGCNN Cls. 1, 2, and 3 are found in the classification supernet BASE, and NAS-DGCNN Cls. 4 is found in the TINY.

(a) Classification on ModelNet40.

| Architecture | OA (%) | Params (M) | MACs (M) | Latency (ms) |
|---|---|---|---|---|
| PointNet (Qi et al., 2017a) | 90.4 | 3.5 | 440 | - |
| PointNet++ (Qi et al., 2017b) | 90.7 | 1.48 | - | - |
| KPConv (Thomas et al., 2019) | 92.9 | 14.99 | 11,650 | - |
| PointASNL (Yan et al., 2020) | 93.2 | 3.36 | 1,705 | - |
| AdaptConv (Zhou et al., 2021) | 93.4 | 1.86 | 3,494 | - |
| SGAS (Li et al., 2020) | 93.2 | 8.49 | - | - |
| PolyConv (Lin et al., 2021) | 93.5 | - | - | - |
| DGCNN (Wang et al., 2019) | 92.9 | 1.810 | 2,426 | 470 |
| NAS-DGCNN Cls. 1 | **93.4** | 0.711 | 1,213 | 368 |
| NAS-DGCNN Cls. 2 | 93.1 | 0.745 | 458 | 192 |
| NAS-DGCNN Cls. 3 | 92.9 | 0.492 | 148 | 80 |
| NAS-DGCNN Cls. 4 | 92.9 | **0.150** | **139** | **78** |

(b) Part segmentation on ShapeNetPart.

| Architecture | mIoU (%) | Params (M) | MACs (G) |
|---|---|---|---|
| PointNet (Qi et al., 2017a) | 83.7 | - | - |
| PointNet++ (Qi et al., 2017b) | 85.1 | - | - |
| DGCNN (Wang et al., 2019) | 85.2 | 1.46 | 4.30 |
| NAS-DGCNN Seg. 1 | **85.4** | 1.35 | 4.01 |
| NAS-DGCNN Seg. 2 | 84.5 | 0.622 | 3.07 |
| NAS-DGCNN Seg. 3 | 84.4 | 0.608 | 2.90 |
| NAS-DGCNN Seg. 4 | 84.2 | **0.537** | **2.35** |

**Searching Method**. Our search method is dedicated to finding architectures geared towards compression and acceleration. We apply a simple random to discover subnets given the parameter constraints guided by the validation accuracy. Specifically, we randomly sample 648 networks from the search space, evaluate the accuracy, and measure computational complexity. Once the architecture is discovered, we adjust the weights of the batch normalization by one epoch to calibrate the change in channel numbers.

## 3 Experiments

**Dataset**. We evaluate our NAS-DGCNN on the Modelnet40 (Wu et al., 2015) for classification and ShapeNetPart (Yi et al., 2016) for part segmentation. Modelnet40 contains a total of 12,311 3D models in 40 categories, of which 9,843 are used for training and 2,468 for testing. ShapeNetPart contains 16,881 shapes with 16 categories, and each category is labeled in 2-6 parts with a total number of 50. We build our code based on Tao et al. (2021). We sample 1,024 and 2,048 points for classification and segmentation, respectively.

**Implementation Detail**. Details of the three supernets are shown in the supplementary material. We use the same settings in each training stage. In each stage, we train the classification and segmentation models for 250 and 200 epochs with a batch size of 32, respectively. The hyperparameters are the same as DGCNN. We do not perform voting tests with random scaling in testing since it is time-consuming and unstable. The whole training and searching process costs less than one GPU day on a single RTX 3090 GPU. Latency is measured on CPU (E5-2630v4×2) with a batch size of 1. We first warm up for 30 iterations and then measure the average latency of 100 steps.

**Classification**. We show the classification results in Table 1a. Our method achieves a maximum average accuracy of 93.4%. We also achieve comparable or better performance in smaller sizes than most hand-designed state-of-the-art networks. Compared to the baseline DGCNN, we can obtain higher accuracy with a smaller size. We achieve an equal accuracy with an approximate 3.7× model compression ratio in the classification supernet BASE. We further achieve a computation compression ratio of up to 17.5× in the classification supernet TINY compared to the original network while maintaining the same level of accuracy. More details about the searched networks can be found in the supplementary material.

**Part segmentation**. We show the segmentation results in Table 1b. Our method achieves a maximum mIoU of 85.4%, outperforming the original DGCNN segmentation network, PointNet, and PointNet++. The mIoU decreases as the networks are compressed, which suggests the complexity of the segmentation task and the requirements of a large model. Within a 1% drop, we compressed the network by more than 2.7× the number of parameters and 1.8× the amount of computation. More details about the searched networks can be found in the supplementary material.

Table 2: Ablation studies on ModelNet40 for classification. Model A, B, C, D, and E are the best architecture found in the corresponding search space, where models A and B are of optimal accuracy, and models C, D, and E are the networks with the minimal computation. Model F is a random sampled architecture with similar computational complexity as E. Model G has the same architecture as E but is trained from scratch. H is the model obtained by fine-tuning E.

| Model | Depth | $k$ & Encoding Channels | Decoding Channels | Retrain | Fine-tune | Params (%) | MACs (%) | OA (%) |
|---|---|---|---|---|---|---|---|---|
| DGCNN | | | | | | 100.0 | 100.0 | 92.9 |
| A | | | ✓ | | | 46.4 | 99.9 | 93.2 |
| B | | ✓ | ✓ | | | 39.2 | 46.8 | 93.3 |
| C | ✓ | | | | | 81.8 | 33.0 | 92.8 |
| D | ✓ | ✓ | | | | 73.8 | 12.7 | 92.9 |
| E | ✓ | ✓ | ✓ | | | 27.2 | 6.1 | 92.9 |
| F | | | | ✓ | | 12.7 | 6.9 | 92.5 |
| G | ✓ | ✓ | ✓ | ✓ | | 27.2 | 6.1 | 92.7 |
| H | ✓ | ✓ | ✓ | | ✓ | 27.2 | 6.1 | 92.8 |

**Ablation Study**. We investigate the effects of our progressive stage-based training strategy through experiments as shown in Table 2. Comparing models A and B with DGCNN (Wang et al., 2019), the results show that dynamic $k$ and scalable model widths can boost the performance of the models. Combining all these components enables the model to be compressed while maintaining validity. The performance of the model E-H suggests that fine-tuning and retraining the found network do not further improve the performance of the models. More ablation studies and details about these models can be found in the supplementary material.

## 4 Conclusion

In this work, we propose NAS-DGCNN, which achieves a significant network compression without compromising performance. In contrast to previous 3D NAS methods, we focus on adjusting the architectural parameters of the network to improve efficiency. The proposed method consumes very few additional computational resources and produces multiple networks of different sizes. Experimental results show that the progressive stage-based training strategy helps obtain architectures and weights with competitive performance comparable with fine-tuned and retrained ones. Besides, compared with the original DGCNN, we achieve 17.5× computation saving on the classification task with the comparable performance and obtain a 2.7× model compression ratio on the part segmentation task with a slight IoU loss. It is safe to conclude that NAS-DGCNN gives a good solution for automatically compressing point cloud processing networks.

## 5 Limitations and Broader Impact Statement

We focus on the impact of using our methods to provide better structures and parameter weights for such networks. As a one-shot weight NAS framework, our method does not require a lot of computational costs and can theoretically be applied to most point cloud networks. There are lots of benefits to using such a method, such as reducing the cost of training and deployment. Implications of using a NAS method include: (1) Lower computational requirements allow models to be deployed on a broader range of devices, which could increase the risk of mispredictions. (2) For complex tasks, model compression and acceleration inevitably bring performance degradation, requiring a careful balance between accuracy and efficiency. We encourage research dedicated to the interpretability and the adversarial robustness of the method. Why this approach enables all networks to be comparable in expressiveness to individually trained networks? Is it possible to quantitatively analyze the redundancy of the networks of different sizes and structures?

**Acknowledgements**. The work is partially supported by the National Nature Science Foundation of China (No. 61976160, 61906137, 61976158, 62076184, 62076182) and Shanghai Science and Technology Plan Project (No.21DZ1204800) and Technology research plan project of Ministry of Public and Security (Grant No.2020JSYJD01).

## References


Cai, H., Gan, C., Wang, T., Zhang, Z., and Han, S. (2020). Once-for-all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*.

Cai, H., Zhu, L., and Han, S. (2019). Proxylessnas: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*.

Chen, M., Peng, H., Fu, J., and Ling, H. (2021). Autoformer: Searching transformers for visual recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12270–12280.

Lawin, F. J., Danelljan, M., Tosteberg, P., Bhat, G., Khan, F. S., and Felsberg, M. (2017). Deep projective 3d semantic segmentation. In *International Conference on Computer Analysis of Images and Patterns*, pages 95–107.

Li, G., Qian, G., Delgadillo, I. C., Muller, M., Thabet, A., and Ghanem, B. (2020). Sgas: Sequential greedy architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1620–1630.

Lin, X., Chen, K., and Jia, K. (2021). Object point cloud classification via poly-convolutional architecture search. In *Proceedings of the ACM International Conference on Multimedia*, pages 807–815.

Liu, H., Simonyan, K., and Yang, Y. (2018). Darts: Differentiable architecture search. In *International Conference on Learning Representations*.

Maturana, D. and Scherer, S. (2015). Voxnet: A 3d convolutional neural network for real-time object recognition. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 922–928.

Nie, X., Liu, Y., Chen, S., Chang, J., Huo, C., Meng, G., Tian, Q., Hu, W., and Pan, C. (2021). Differentiable convolution search for point cloud processing. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7437–7446.

Qi, C. R., Su, H., Mo, K., and Guibas, L. J. (2017a). Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 652–660.

Qi, C. R., Yi, L., Su, H., and Guibas, L. J. (2017b). Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in Neural Information Processing Systems*, 30.

Su, H., Maji, S., Kalogerakis, E., and Learned-Miller, E. (2015). Multi-view convolutional neural networks for 3d shape recognition. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 945–953.

Tang, H., Liu, Z., Zhao, S., Lin, Y., Lin, J., Wang, H., and Han, S. (2020). Searching efficient 3d architectures with sparse point-voxel convolution. In *Proceedings of European Conference on Computer Vision*, pages 685–702.

Tao, A., Pengliang, J., and et.al. (2021). dgcnn.pytorch. `https://github.com/AnTao97/dgcnn.pytorch`.

Thomas, H., Qi, C. R., Deschaud, J.-E., Marcotegui, B., Goulette, F., and Guibas, L. J. (2019). Kpconv: Flexible and deformable convolution for point clouds. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*.

Wang, Y., Sun, Y., Liu, Z., Sarma, S. E., Bronstein, M. M., and Solomon, J. M. (2019). Dynamic graph cnn for learning on point clouds. *ACM Transactions On Graphics*, 38(5):1–12.

Wu, W., Qi, Z., and Fuxin, L. (2019). Pointconv: Deep convolutional networks on 3d point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9621–9630.

Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., and Xiao, J. (2015). 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1912–1920.

Yan, X., Zheng, C., Li, Z., Wang, S., and Cui, S. (2020). Pointasnl: Robust point clouds processing using nonlocal neural networks with adaptive sampling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5589–5598.

Yi, L., Kim, V. G., Ceylan, D., Shen, I.-C., Yan, M., Su, H., Lu, C., Huang, Q., Sheffer, A., and Guibas, L. (2016). A scalable active framework for region annotation in 3d shape collections. *ACM Transactions on Graphics (ToG)*, 35(6):1–12.

Yu, T., Meng, J., and Yuan, J. (2018). Multi-view harmonized bilinear network for 3d object recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 186–194.

Zhou, H., Feng, Y., Fang, M., Wei, M., Qin, J., and Lu, T. (2021). Adaptive graph convolution for point cloud analysis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4965–4974.

Table 3: Search space of classification (left) and segmentation (right) network. Layer($a - b : c$) means a channel number between $[a, b]$ and is divisible by $c$. Layer* indicates a layer that can be skipped.

(a) Search space of classification.

| NAS-DGCNN_cls(Base) | NAS-DGCNN_cls(Tiny) |
|---|---|
| kNN(10-30) | kNN(10-25) |
| EdgeConv(32-64:8) | EdgeConv(24-48:8) |
| kNN(10-30) | kNN(10-25) |
| EdgeConv(32-64:8) | EdgeConv(48-96:8) |
| kNN(10-30)* | kNN(10-25)* |
| EdgeConv(64-128:8)* | EdgeConv(48-96:8)* |
| kNN(10-30)* | |
| EdgeConv(64-256:8)* | |
| mlp(1024) | mlp(320) |
| Max & Avg Pooling | Max & Avg Pooling |
| linear(64-512:8) | linear(40-256:8) |
| linear(32-256:8) | |
| linear(40) | linear(40) |
| # of Subnets: $1.49 \times 10^{12}$ | # of Subnets: $1.87 \times 10^{7}$ |

(b) Search space of segmentation.

| NAS-DGCNN_partseg |
|---|
| transform_net(Tiny) |
| kNN(10-80) |
| EdgeConv(48-64:8,48-64:8) |
| kNN(10-80) |
| EdgeConv(48-64:8,48-64:8) |
| kNN(10-80) |
| EdgeConv(48-64:8) |
| Pooling & one-hot label |
| linear(128-256:8) |
| linear(128-256:8) |
| linear(64-128:8) |
| linear(50) |
| # of subnets: $2.17 \times 10^{11}$ |

## A Search Space

We design three supernets with different sizes for two tasks, e.g., classification and segmentation. Specifically, the classification supernet Tiny aims to find an architecture with a tiny model size. The classification supernet Base discovers a network in a larger search space. The segmentation supernet enables searching for different dense prediction models.

We summarize three search spaces as shown in Table 3, where the classification supernet Base, the classification supernet Tiny,and the segmentation supernet contain $1.49 \times 10^{12}$, $1.87 \times 10^{7}$, $2.17 \times 10^{11}$ subnets, respectively.

## B Ablation Study

**Knowledge Distillation**. Quantitative results of knowledge distillation is shown in Figure 2 and Table 4. We sample the same 648 architectures from both NAS-DGCNN and NAS-DGCNN w/o KD. In Figure 2, the performance of subnets sampled from NAS-DGCNN obtains significant overall accuracy. Besides, distilled NAS-DGCNN achieves 92.28% accuracy and outperforms the one without knowledge distillation, which suggests knowledge distillation improves subnets. The average MACs of subnets superior to DGCNN are lower than that from NAS-DGCNN without knowledge distillation, which indicates distillation benefits smaller architectures.

**Random Search**. We investigate the effects of different search numbers. As shown in Table 5, we summarize the architectures found under different search numbers on the cloud point classification task. The highest overall accuracy of 93.4% is obtained among the first 648 sampled networks and surpasses the one from the first 328 networks. The architectures found from 1,000 or 2,000 networks have 5.12% MACs or 21.3% parameters, which are less than 1% smaller than the architectures with the smallest parameters and the least computational cost found from 648 networks. It is safe to

Table 4: NAS-DGCNN w/ knowledge distillation (KD) vs. NAS-DGCNN w/o KD. The mOA and mMAC are the average overall accuracy and average MACs of the found architectures, respectively. † indicates these results from models superior to DGCNN.

| **Method** | **mOA(%)** | **# of Arch†** | **mMACs†(M)** |
|---|---|---|---|
| NAS-DGCNN w/ KD | 92.28 | 98/648 | 798.2 |
| NAS-DGCNN w/o KD | 92.18 | 45/648 | 931.3 |

Table 5: Accuracy and model size of the found architectures under different search numbers on the point cloud classification task.

| Search Number | Maximum OA (%) | Minimum MACs (%) | Minimum Params (%) |
|---|---|---|---|
| 98 | 93.2 | 5.68 | 25.1 |
| 328 | 93.2 | 5.68 | 25.1 |
| 648 | 93.4 | 5.68 | 22.1 |
| 1000 | 93.4 | 5.12 | 21.3 |
| 2000 | 93.4 | 5.12 | 21.3 |

conclude that searching for 648 networks is enough to find small architectures with the highest accuracy, while sampling more than 3× networks cannot significantly improve.

# C Network Architectures

## C.1 Networks for classification

Searched classification network architectures are shown in table 6.

## C.2 Networks for segmentation

Searched part segmentation network architectures are shown in table 7.

## C.3 Networks for ablation study

Network A-F in ablation studies are shown in table 8.

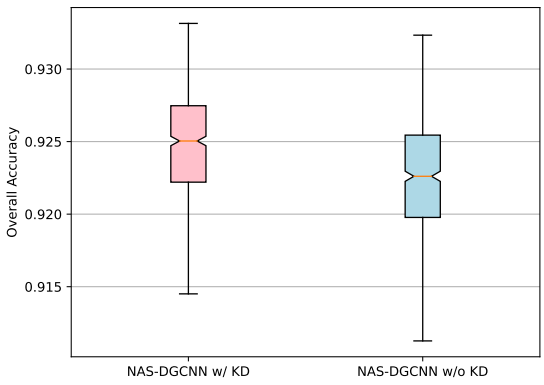


Figure 2: Boxplot of the overall accuracy of sampled found architectures between NAS-DGCNN and NAS-DGCNN w/o KD.

Table 6: Classification network architectures. NAS-DGCNN Cls. 1, 2, and 3 are found in the classification supernet Base, and NAS-DGCNN Cls. 4 is found in the classification supernet Tiny.

| NAS-DGCNN Cls. 1 | NAS-DGCNN Cls. 2 | NAS-DGCNN Cls. 3 | NAS-DGCNN Cls. 4 |
|---|---|---|---|
| kNN(26) | kNN(25) | kNN(17) | kNN(21) |
| EdgeConv(64) | EdgeConv(56) | EdgeConv(32) | EdgeConv(24) |
| kNN(22) | kNN(21) | kNN(17) | kNN(21) |
| EdgeConv(56) | EdgeConv(56) | EdgeConv(48) | EdgeConv(88) |
| kNN(13) | kNN(13) | - | - |
| EdgeConv(88) | EdgeConv(72) | - | - |
| kNN(25) | - | - | - |
| EdgeConv(120) | - | - | - |
| mlp(1024) | mlp(1024) | mlp(1024) | mlp(320) |
| Max & Avg Pooling | Max & Avg Pooling | Max & Avg Pooling | Max & Avg Pooling |
| linear(144) | linear(240) | linear(184) | linear(160) |
| linear(208) | linear(168) | linear(120) | - |
| linear(40) | linear(40) | linear(40) | linear(40) |

Table 7: Part segmentation network architectures.

| NAS-DGCNN Seg. 1 | NAS-DGCNN Seg. 2 | NAS-DGCNN Seg. 3 | NAS-DGCNN Seg. 4 |
|---|---|---|---|
| transform_net | transform_net(Tiny) | transform_net(Tiny) | transform_net(Tiny) |
| kNN(45) | kNN(37) | kNN(24) | kNN(29) |
| EdgeConv(64,64) | EdgeConv(56,56) | EdgeConv(56,56) | EdgeConv(56,56) |
| kNN(28) | kNN(44) | kNN(49) | kNN(36) |
| EdgeConv(64,64) | EdgeConv(56,56) | EdgeConv(56,56) | EdgeConv(56,56) |
| kNN(65) | kNN(22) | kNN(38) | kNN(27) |
| EdgeConv(64) | EdgeConv(56) | EdgeConv(48) | EdgeConv(48) |
| Pooling & one-hot label | Pooling & one-hot label | Pooling & one-hot label | Pooling & one-hot label |
| linear(152) | linear(160) | linear(240) | linear(176) |
| linear(240) | linear(216) | linear(184) | linear(208) |
| linear(128) | linear(120) | linear(112) | linear(88) |
| linear(50) | linear(50) | linear(50) | linear(50) |

## D Reproducibility Checklist

1. For all authors...

   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]

   (b) Did you describe the limitations of your work? [Yes] See Section 5.

   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Section 5.

   (d) Have you read the ethics author's and review guidelines and ensured that your paper conforms to them? `https://automl.cc/ethics-accessibility/` [Yes]

Table 8: Classification network architectures for ablation study.

| A | B | C | D | E | F |
|---|---|---|---|---|---|
| kNN(20) | kNN(24) | kNN(20) | kNN(28) | kNN(17) | kNN(20) |
| EdgeConv(64) | EdgeConv(48) | EdgeConv(64) | EdgeConv(56) | EdgeConv(32) | EdgeConv(32) |
| kNN(20) | kNN(24) | kNN(20) | kNN(14) | kNN(17) | kNN(20) |
| EdgeConv(64) | EdgeConv(40) | EdgeConv(64) | EdgeConv(32) | EdgeConv(48) | EdgeConv(64) |
| kNN(20) | kNN(24) | kNN(20) | kNN(14) | - | kNN(20) |
| EdgeConv(128) | EdgeConv(72) | EdgeConv(128) | EdgeConv(72) | - | EdgeConv(64) |
| kNN(20) | kNN(24) | - | - | - | - |
| EdgeConv(256) | EdgeConv(152) | - | - | - | - |
| mlp(1024) | mlp(1024) | mlp(1024) | mlp(1024) | mlp(1024) | mlp(256) |
| Max & Avg Pooling | Max & Avg Pooling | Max & Avg Pooling | Max & Avg Pooling | Max & Avg Pooling | Max & Avg Pooling |
| linear(94) | linear(168) | linear(512) | linear(360) | linear(184) | linear(128) |
| linear(176) | linear(120) | linear(256) | linear(64) | linear(120) | - |
| linear(40) | linear(40) | linear(40) | linear(40) | linear(40) | linear(40) |

2. If you are including theoretical results...

   (a) Did you state the full set of assumptions of all theoretical results? [N/A] The article does not contain the theoretical results.

   (b) Did you include complete proofs of all theoretical results? [N/A] The article does not contain the theoretical results.

3. If you ran experiments...

   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results, including all requirements (e.g., `requirements.txt` with explicit version), an instructive `README` with installation, and execution commands (either in the supplemental material or as a URL)? [Yes] Code can be available in supplementary materials and online.

   (b) Did you include the raw results of running the given instructions on the given code and data? [Yes] The pre-trained model is included in the published code.

   (c) Did you include scripts and commands that can be used to generate the figures and tables in your paper based on the raw results of the code, data, and instructions given? [Yes]

   (d) Did you ensure sufficient code quality such that your code can be safely executed and the code is properly documented? [Yes]

   (e) Did you specify all the training details (e.g., data splits, pre-processing, search spaces, fixed hyperparameter settings, and how they were chosen)? [Yes]

   (f) Did you ensure that you compared different methods (including your own) exactly on the same benchmarks, including the same datasets, search space, code for training and hyperparameters for that code? [No]Some methods do not have available code, so the results from the original paper are used directly.

   (g) Did you run ablation studies to assess the impact of different components of your approach? [Yes] See section 3.

   (h) Did you use the same evaluation protocol for the methods being compared? [No] Some methods do not have available code, so the results from the original paper are used directly.

   (i) Did you compare performance over time? [No] Due to lack of computing resources, we did not run the experiment multiple times.

(j) Did you perform multiple runs of your experiments and report random seeds? [No] Due to lack of computing resources, we did not run the experiment multiple times. However, we controlled the seeds of the experiments to provide reproducibility.

(k) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] Due to lack of computing resources, we did not run the experiment multiple times.

(l) Did you use tabular or surrogate benchmarks for in-depth evaluations? [No]

(m) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See section 3.

(n) Did you report how you tuned hyperparameters, and what time and resources this required (if they were not automatically tuned by your AutoML method, e.g. in a NAS approach; and also hyperparameters of your own method)? [N/A] We used the same hyperparameters as DGCNN(Wang et al., 2019). We did not adjust hyperparameters.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes]

(b) Did you mention the license of the assets? [Yes]

(c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We have released our code.

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] We used the existing datasets and did not collect new data.

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] We used the existing datasets, and the datasets do not contain personally identifiable information or offensive content.

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]