Poly-PC: A Polyhedral Network for Multiple Point Cloud Tasks at Once

Tao Xie[1,3] Shiguang Wang[2,3] Ke Wang[1] Linqi Yang[1] Zhiqiang Jiang[1]
Xingcheng Zhang[3] Kun Dai[1] Ruifeng Li[1*] Jian Cheng[2]
[1]Harbin Institute of Technology [2]University of Electronic Science and Technology of China [3]SenseTime Research
{xietao1997, wangke, nana07, jzq, daikun, lrf100}@hit.edu.cn
{wangshiguang, chenjian}@uestc.edu.cn {zhangxingcheng}@sensetime.com

Abstract

In this work, we show that it is feasible to perform multiple tasks concurrently on point cloud with a straightforward yet effective multi-task network. Our framework, Poly-PC, tackles the inherent obstacles (e.g., different model architectures caused by task bias and conflicting gradients caused by multiple dataset domains, etc.) of multi-task learning on point cloud. Specifically, we propose a residual set abstraction (Res-SA) layer for efficient and effective scaling in both width and depth of the network, hence accommodating the needs of various tasks. We develop a weight-entanglement-based one-shot NAS technique to find optimal architectures for all tasks. Moreover, such technique entangles the weights of multiple tasks in each layer to offer task-shared parameters for efficient storage deployment while providing ancillary task-specific parameters for learning task-related features. Finally, to facilitate the training of Poly-PC, we introduce a task-prioritization-based gradient balance algorithm that leverages task prioritization to reconcile conflicting gradients, ensuring high performance for all tasks. Benefiting from the suggested techniques, models optimized by Poly-PC collectively for all tasks keep fewer total FLOPs and parameters and outperform previous methods. We also demonstrate that Poly-PC allows incremental learning and evades catastrophic forgetting when tuned to a new task.

1. Introduction

With the advances in deep learning, modern architectures offer tremendous improvements in 3D understanding [29, 36, 39, 54], e.g., point classification, segmentation, and detection, etc. Nevertheless, these networks are inefficient when handling numerous tasks since they are often intended to accomplish a single task. Even if parallel computing can address this issue, the memory footprints and storage costs grow linearly with the number of networks, rendering them unaffordable with constrained resources.

Multitask learning (MTL) [3, 12, 13] offers a solution to this difficulty. In vision tasks, MTL models have been predominantly proposed to simultaneously perform depth estimation, surface normal estimation, and semantic segmentation on an input image [14, 19, 52]. Besides, the joint part-of-speech tagging, chunking, and named-entity recognition for Natural Language Processing (NLP) have also been investigated [8, 9]. Since a substantial piece of the network (i.e., the backbone) is shared among tasks, an MTL model offers benefits in terms of complexity, inference time, and learning efficiency. However, training multiple tasks for point cloud poses two key challenges:

1) In contrast to common vision tasks, where a backbone that performs well on image classification can be directly ported to other tasks, the backbone for point cloud tasks must be carefully developed. Consequently, it is not feasible for all point cloud tasks to directly share a single backbone.

2) Instead of using a multi-task dataset as input, we seek to jointly perform multiple tasks on point cloud with multiple dataset domains as input. Thus, in such a circumstance, multi-task learning would result in considerable disparities in directions and magnitude of different task gradients, a phenomenon known as task interference or negative transfer [34]. Meanwhile, task difficulty induced by multiple dataset domains is also different, so task prioritization should be considered to prevent placing undue emphasis on easier tasks when optimizing the multi-task network.

To address the first challenge, we introduce residual set abstraction (Res-SA) layer, a scalable point feature learning module that can adapt to requirements for a variety of tasks in terms of width and depth of the network. Simultaneously, when multiple tasks are presented to us, to reduce the manpower loss caused by manually designing the network, we seek to find the optimal architecture for each task using neural network search (NAS). Thus, we construct different search spaces (neighbour points number, group radius, width, and depth, etc.) for multiple tasks on Res-SA. Then, inspired by AutoFormer [4], BigNAS [49] and slimmable

*Corresponding author.

networks [48], we propose weight-entanglement-based one-shot NAS technique that entangles the weights of different tasks in the same layer, enabling different tasks to share parameters in their common parts for efficient storage deployment and offering task-specific parameters for learning task-related features. Moreover, unlike previous works that share or monopolize all parameters in one layer for all tasks, such strategy allows different tasks to share a certain proportion of parameters in a single layer, achieving fine-grained parameter sharing.

For negative transfer, previous methods narrow down the problem to two types of differences (i.e., gradient magnitudes and directions) among task gradients, and propose several algorithms [5, 6, 12, 18, 35, 50] to homogenize the differences. However, these methods exclusively focus on one aspect of task gradient differences and also disregard the difficulty of various tasks. Intuitively, it is conceivable for easy tasks to dominate learning while harder ones stagnate during multi-task training. In response, we introduce a task-prioritization-based gradient balance algorithm that resolves negative transfer as a whole by homogenizing both gradient magnitudes and directions across tasks via task prioritization. Specifically, we evaluate task prioritization using the previous loss record. Then, we use task prioritization to homogenize task gradient magnitudes and directions in the current epoch, endowing difficult task with larger grad scale and enabling direction of final merged gradient vector closer to that of the difficult task. Since the task prioritization is dynamically adjusted, our algorithm avidly concentrates on the difficult task learning at each epoch and ensures that each task converges to the optimal solution.

Over the well-trained Poly-PC, we undertake an evolutionary search with model size constraint to identify promising architectures for different tasks. Experiments show that the searched models with weights inherited from the supernet outperform several baselines and are comparable with the state-of-the-arts trained individually for specific tasks. We also demonstrate that Poly-PC allows incremental learning and evades catastrophic forgetting when generalizing to a new task. Thus, Poly-PC is parameter-efficient and can scale up more gracefully as the number of tasks increases.

The key contributions can be summarized as follows: 1) We propose Poly-PC to perform multi-task learning on point cloud. To the best of our knowledge, Poly-PC is the first framework that takes multiple dataset domains as input for multi-task learning on point cloud. 2) We introduce Res-SA layer that meets the needs of different tasks in both the width and depth of the network. 3) We develop weight-entanglement-based one-shot NAS technique to find optimal architectures for different tasks as well as shared parameters inside each layer for efficient storage. 4) We propose task-prioritization-based gradient balance algorithm that resolves negative transfer as a whole to promote the training of Poly-PC. 5) We demonstrate that Poly-PC allows incremental learning with fewer parameters.

2. Related work

Point cloud analysis. Point cloud analysis, e.g., point classification, detection, and segmentation, is particularly vital for downstream tasks. The development of point-based methods can be traced back to PointNet [29] and PointNet++ [30]. PointNet focuses on the processing of raw irregular point clouds and the learning of global information representations, primarily by employing point-wise multilayer perceptrons (MLP) and a symmetric function (i.e., max-pooling). As an expansion of PointNet, PointNet++ introduces set abstraction (SA) as the local aggregator for encoding local structures and presents a universal hierarchical encoder-decoder pipeline for point cloud analysis. Following this encoder-decoder architecture, some plug-and-play methods [17, 20, 22, 33, 39, 51] have been proposed, mainly focusing on local feature extractors by convolutions, graphs, or self-attention mechanisms to thoroughly explore local geometric information. However, these works focus on optimizing task-specific components, which can not be directly ported to other tasks.

Gradient homogenization for MTL. During the training of an MTL model, gradient magnitudes and directions of various tasks interact intricately via backpropagation. For gradient magnitudes, recent dynamic weighting algorithms update weights periodically according to the uncertainty of losses [13] and the rates of change of losses [19]. GradNorm [5] seeks to learn all tasks at the same pace, but they propose to learn these weights as parameters, resulting in additional memory consumption. Instead, both RotoGrad [12] and IMTL-G [18] normalize and scale each gradient of the shared parameters by locating weights. For gradient direction, Maninis et al. [23] and Sinha et al. [37] propose to make task gradients statistically indistinguishable via adversarial training. More recently, GradDrop [6] randomly drops elements of the task gradients based on a sign-purity score, whereas PCGrad [50] proposes to drop the projection of one task gradient onto another if they are in conflict. In this work, we propose a task-prioritization-based gradient balance algorithm that resolves negative transfer as a whole to promote the training of the network.

3. Methodology

Poly-PC concurrently learns K distinct tasks on point cloud, that is, discovering K mappings from K different datasets $\{\chi_k\}$ to a task-specific collection of labels $\{y_k\}$, $k = 1, 2, ..., K$. Poly-PC aims at finding optimal architectures to ensure high performance for all tasks while using as few total parameters as possible, i.e., how to design and optimize Poly-PC.

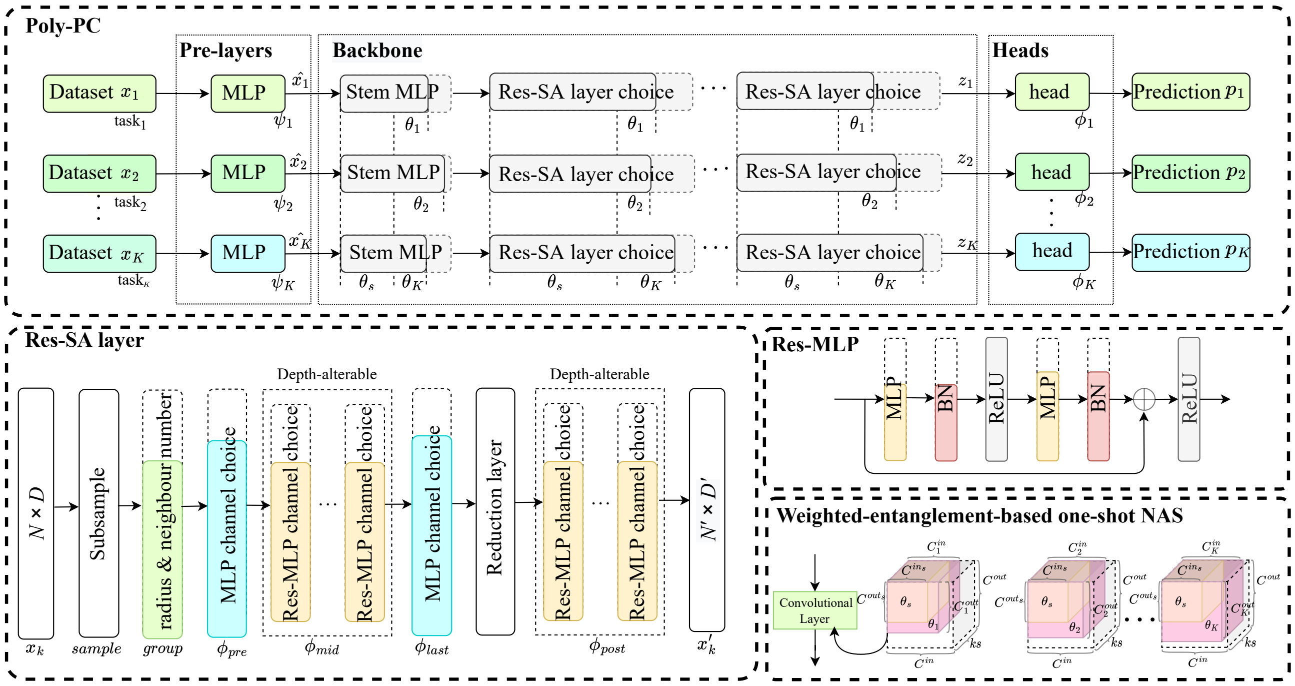


Figure 1. **Above:** The overall architecture of Poly-PC. **Bottom left:** The structure of Res-SA layer. **Bottom right:** One Res-MLP layer in Φ_{mid} or Φ_{post} and weight-entanglement-based one-shot NAS technique that entangles the weights of different tasks in one layer to consider their common parts as task-shared weights and the rests as task-specific weights. Note that the stem MLP layer and each Res-SA layer are dynamic. The parts with solid lines mean they are chosen whereas those with dashed lines are not. The channel choice means the output channel of each layer in the backbone. The partial color in Weight-entanglement-based one-shot NAS part means the task-shared weight and task-specific weight respectively, while colors in other parts have no symbolic meaning and are simply for aesthetic purposes.

3.1. Design Poly-PC

Poly-PC is characterized by three components: pre-layer, backbone, and head networks, as shown in Fig. 1.

Pre-layers. Since Poly-PC accepts multiple dataset domains as input while the dimensions of these domains are different, we employ a multilayer perceptron (MLP) to map $x_k \in \chi_k$ into the same dimension as $\hat{x}_k$ before backbone. Each task k has its exclusive parameters ψ_k for such MLP.

Backbone. The backbone consists of a stem MLP used to map the input into a higher dimension and several Res-SA layers used to extract point features. The backbone concludes a set of task-shared parameters θ_s for K tasks and task-specific parameters θ_k for task k to transform each input $\hat{x}_k$ into a intermediate representation $z_k = f(\hat{x}_k; \theta_s, \theta_k) \in \mathbb{R}^d$, where d denotes the dimension of z.

Heads. Additionally, each task k has a head network h_k, with exclusive parameters ϕ_k, which takes the intermediate feature z_k as input and outputs the prediction $p_k = h_k(z_k; \phi_k)$ for the corresponding task.

In this work, we focus on finding optimal backbone for each task rather than designing network heads. As illustrated in Sec. 1, we design a scalable point feature extraction module (Res-SA layer) to meet the needs of various tasks and find optimal architectures for all tasks while keeping as few total parameters as possible (weight-entanglement-based one-shot NAS).

Residual set abstraction layer. We choose classical set abstraction (SA) layer [30] as baseline due to its wide range of applications on point cloud. Intriguingly, we find that adding more SA blocks or channels naively does not result in a discernible improvement in accuracy, primarily due to gradient vanishment and overfitting. In contrast, we present residual set abstraction (Res-SA) layer to scale up the classic SA in an effective and efficient way. Res-SA inherits most of the principles of SA layer, as shown in the bottom left of Fig. 1, using a subsampling layer (e.g., FPS algorithm [30]) to downsample the input points, a grouping layer to search neighbors for each sampled point, and a reduction layer to consolidate features inside the neighbors. The only difference is the point feature extraction. Res-SA decouples the point feature extraction into following steps:

$$g_i = \Phi_{post}(R(\Phi_{last}(\Phi_{mid}(\Phi_{pre}([f_{i,j}; (p_{i,j} - p_i)/r]))))), \tag{1}$$

where r is the group radius; p_i is the i-th sampled point coordinate; $p_{i,j}$ and $f_{i,j}$ are the j-th neighbor point coordinate and feature of p_i respectively; Φ_{pre} and Φ_{last} are basic feature extractor composed of an MLP followed by a batch normalization and an activation layer; Φ_{mid} and Φ_{post} are depth-alterable (i.e., how many times Res-MLP repeats) residual MLP blocks, in which each block is combined by MLP, normalization and activation layers (repeated two times), as shown in the bottom right of Fig. 1; R is the reduction layer (e.g., max-pooling) that aggregates features for point i from its neighbors. By embedding residual connections into Φ_{mid} and Φ_{post}, Res-SA can be readily expanded to dozens of layers without vanishing gradient problem. Moreover, bottleneck design and inverted bottleneck design are leveraged in Φ_{mid} and Φ_{post}, respectively.

Weight-entanglement-based one-shot NAS. Previous one-shot NAS methods [10, 25] typically share weights across subnets yet isolate the weights of various operators inside the same layer. These methods perform admirably when employed to search architectures for a single task. Nonetheless, such technique cannot enable each task to share the weights of the different operators in the same layer, nor can it offer partially shared and unshared parameters inside a single layer. Inspired by Autoformer [4], BigNAS [49] and slimmable networks [48], we propose weight-entanglement-based one shot NAS technique to tackle this issue, which entangles the weights of all tasks in each Res-SA layer, allowing different tasks to share weights for their common parts.

More concretely, as shown in the bottom right of Fig. 1, for the i-th convolution of K tasks in the Res-SA, we predefine K search spaces for the input channel S_k^{in} and output channel S_k^{out}. Accordingly, the weight W^i of the i-th convolution is defined as $W^i \in [C^{out}, C^{in}, ks]$, where C^{out} and C^{in} are large enough to accommodate the requirements of different tasks and ks is the kernel size. Then, we find the task-shared weights P^i of W^i by slicing $P^i = W^i[:C^{out_s}, :C^{in_s}, :]$, where C^{out_s} and C^{in_s} are the minimum number of $\{S_1^{out}, S_2^{out}, \ldots, S_K^{out}\}$ and $\{S_1^{in}, S_2^{in}, \ldots, S_K^{in}\}$ respectively. For each batch in supernet training, we follow the central idea of SPOS [10], uniformly sampling a number C_k^{in} from S_k^{in} and C_k^{out} from S_k^{out} for the k-th task. We slice out the weights W_k^i for current batch by $W_k^i = W^i[:C_k^{out}, :C_k^{in}, :]$, which is used to produce the output. Notably, the common parts P^i of W_k^i $(k = 1, 2, ..., K)$ is viewed as task-shared weights optimized in the global group while the others viewed as task-specific weights optimized in the task-specific group. The implementation details are given in Appendix C.2.

Such weight sharing within a single layer provokes the weight updating of W_k^i to be entangled with each other, that is, the training of any task will impact the task-shared parameters. In this work, we apply weight-entanglement-based one-shot NAS to all convolution and normalization layers of the backbone, enabling different tasks to share parameters in their common parts and monopolize the rests. The designed search space is given in Sec. 5.2.

Task-specific principle features. We further embed a task-specific attention module into Res-SA, allowing Poly-PC to acquire additional task-specific principle features. The attention module learns a soft attention mask for the output features, which automatically determines the importance of features for the individual task along the channel dimension, resulting in self-supervised, end-to-end learning of more task-specific principal features. We adopt SENet [11] as attention module and integrate it before reduction layer into Res-SA. Each task k has its exclusive parameters Λ_k.

3.2. Optimize Poly-PC

We seek to optimize the architectural parameters $\{\psi_1, \psi_2, ..., \psi_K, \theta_s, \theta_1, \theta_2, ..., \theta_K, \phi_1, \phi_2, ..., \phi_K, \Lambda_1, \Lambda_2, .., \Lambda_K\}$ of Poly-PC by concurrently minimizing all task losses, that is, $L_k(p_k, y_k)$ for $k = 1, 2, ..., K$. It is a prior multiobjective optimization problem [35] in which an unique proxy loss comprised of a linear combination of the task losses, $L = \sum_k w_k L_k$, is optimized. Whereas this solution streamlines the optimization problem, it may also result in negative transfer, thus lowering overall performance due to unequal competition across tasks for the task-shared parameters. Moreover, Poly-PC is designed for jointly optimizing multiple point cloud tasks under different dataset domains, where the varied dataset domains will further exacerbate the negative transfer.

In this work, we propose task-prioritization-based gradient balance (TGB) algorithm to resolve negative transfer as a whole by homogenizing both magnitude and direction of task gradients.

Task prioritization. The central theme of proposed algorithm is to prioritize learning for difficult task, where difficulty is measured by the previous loss record. We define task difficulty as task prioritization, which is measured as the ratio of the change rate for the task-specific loss at time $t-1$ and $t-2$ to the change rate of the total loss at time $t-1$ and $t-2$, formulated as:

$$r_k = (\frac{L_k^{t-1}/L_k^{t-2}}{L^{t-1}/L^{t-2}})^{\alpha}, k = 1, 2..., K, \tag{2}$$

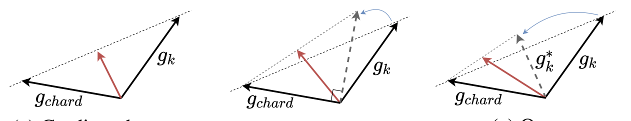
where t is an epoch index; L_k^{t-1} is the loss of the k-th task at $t-1$ epoch; L^{t-1} is the total loss of all tasks at $t-1$ epoch; α is a hyper-parameter (set to 1 in our implementation).

Following [19], we also normalize r_k as:

$$TH_k^t = \frac{K exp(r_k/T)}{\sum_{i=1}^{K} exp(r_i/T)}, k = 1, 2..., K, \tag{3}$$

with K being the number of tasks. The temperature T (set to 2) determines the softness of the task weighting in the softmax operator. TH_k^t is task prioritization measure of the k-th task. Note that for $t = 1, 2$, we initialize $r_k = 1$ since there is no prior information about losses.

Gradient-magnitude homogenization. At t epoch, we also endeavor to homogenize gradient magnitudes $w_k^t L_k^t$



(a) Gradient descent (b) PCGrad (c) Ours

Figure 2. The combined update vector of g_k and g_{chard} with naive gradient descent, PCGrad, and our algorithm. The naive gradient descent simply adds the two gradient vectors. PCGrad first projects g_k onto the normal plane of g_{chard} (vice versa), and then adds g_{chard} and the projected gradient. Our algorithm constructs consensus vectors g_k^* of g_k to enable the direction of the final merged gradient vector closer to that of g_{chard}, compelling the network to concentrate on the difficult task in current epoch.

across tasks, as large magnitude discrepancies may cause some tasks to dominate the learning process. Without the prior of task prioritization, we expect that the loss scale $w_k^t L_k^t$ of task k contributes equally to the overall loss L^t:

$$w_k^t L_k^t = \frac{1}{K} L^t \Rightarrow w_k^t = \frac{L^t}{K L_k^t}. \tag{4}$$

However, the total loss L^t and the task k loss L_k^t at t epoch are not available. Note that we use the sum losses of all iterations for task k at t epoch as L_k^t, so we cannot derive L_k^t at the beginning of t epoch. Similarly, we can not derive L^t at t epoch. To tackle this issue, we assume that L_k^t have the same proportion of L^t at t epoch as it does at $t-1$ epoch:

$$\frac{L_k^t}{L^t} = \frac{L_k^{t-1}}{L^{t-1}} \Rightarrow w_k^t = \frac{L^{t-1}}{K L_k^{t-1}}. \tag{5}$$

It is a plausible assumption if all tasks are learned at a comparable pace. By incorporating the task prioritization TH_k into w_k^t, we can get the loss weight of task k at t epoch as:

$$\hat{W}_k^t = TH_k^t w_k^t. \tag{6}$$

For $t = 1, 2$, we also initialize $w_k^t = 1$.

Therefore, for $t \geq 2$, the total loss can be formulated as:

$$L = \sum_k \hat{W}_K^t L_k^t (h_k(f(\hat{x}_k; \theta_s, \theta_k); \phi_k), y_k) \tag{7}$$

Gradient-direction homogenization. Gradient-direction homogenization is established by the sign of the relevant gradient vector components associated with each task. Specifically, given a gradient vector of the most difficult task (the largest TH_k^t), we build consensus vectors of other tasks by preserving those components that point toward the identical direction (i.e., those with the same sign) and altering the conflicting components. In this way, the direction in which the gradient vectors of all tasks add up will be closer to the direction of the difficult task, enabling the network to emphasis on learning of the difficult task.

We elaborate how to deal with the conflicting components. Given loss $W_k^t L_k^t$ of the task k, we perform a backward pass to obtain gradients of θ_s: $g_k = \bigtriangledown_{\theta_s} W_k^t L_k^t$. To measure the agreement between different task gradients, we first find the gradient of the most difficult task at t epoch according to Eq. (3) as g_{chard}. Then, we define the following function:

$$\Psi(g_k, g_{chard})^{(i)} = \begin{cases} 1, sgn(g_k^{(i)}) = sgn(g_{chard}^{(i)}) \\ 0, otherwise \end{cases} \tag{8}$$

where $g_k^{(i)}$ indicates the i-th gradient component of the k-th task and $sgn(\cdot)$ represents the sign function. Function $\Psi(\cdot)$ verifies whether the signs of the gradient components between task k and the most difficult task agree in an element-wise manner. When two components possess the identical sign for a given i, it yield 1; if difference, it yields 0. The overall size of the gradient vectors is denoted by the number of task-shared parameters, i.e., $n = |\theta_s|$.

After which, we determine the value of each component under the consensus gradient g_k^* of task k. The value of the i-th component g_k^* is formulated as follows:

$$(g_k^*)^{(i)} = \Psi(g_k, g_{chard})^{(i)} g_k^{(i)}, i = 1, 2..., n. \tag{9}$$

Specifically, $\Psi(g_k, g_{chard})^{(i)} = 1$ implies that gradient component i is consistent between the task k and the difficult task. If there is no consensus ($\Psi(g_k, g_{chard})^{(i)} = 0$), however, the $(g_k^*)^{(i)}$ is set as zero to resolve the conflict.

Finally, the gradient vector of the shared parameters θ_s is formulated as:

$$g_{\theta_s}^* = \frac{1}{K}(g_1^* + g_2^* + g_{chard} + ... + g_K^*), \tag{10}$$

where $g_{\theta_s}^*$ is utilized to update the shared parameters θ_s.

Compared with naive gradient descent and PCGrad [50], our proposed algorithm leverages task prioritization to homogenize task gradient directions in the current epoch, enabling the direction of the final merged gradient vector closer to that of difficult task, as shown in Fig. 2. Since the difficulty degree of each task is dynamically adjusted, our proposed algorithm focuses on the learning of difficult tasks at each epoch, and finally ensures that each task converges to the optimal solution.

4. Search pipline

Supernet training. At each iteration of supernet training, we randomly sample a subnet for each task from the search space (detailed in Sec. 5.2) and update its associated weights in Poly-PC while freezing the rest, where the task-shared parameters are optimized in global group and the task-specific parameters are optimized in task-specific group. More details are given in Appendix C.1.

Evolution search under resource constraints. Over the well-trained Poly-PC, we undertake an evolution search on it to identify the optimal subnets for each task. Subnets are assessed and selected based on the evolution algorithm manager. Our objective here is to maximize the proxy performance of each task while minimizing the model size (Parameters). Detailed process is also given in Appendix C.2.

5. Experiment

In this work, we first assess the performance of Poly-PC when simultaneously training three tasks, including 3D shape classification, semantic segmentation, and object detection. Then, we demonstrate that Poly-PC allows incremental learning when fitted to a new task or dataset.

5.1. Dataset and metric

ModelNet40 [44]. The ModelNet40 dataset contains 12,311 CAD models with 40 object categories. They are split into 9,843 models for training and 2,468 for testing. Following [22], we employ the overall accuracy (OA) across all classes to evaluate Poly-PC for ModelNet40.

S3DIS [1]. The S3DIS dataset is comprised of 271 rooms in six sections spanning three buildings. Following a standard process, we evaluate Poly-PC as: area 5 is withheld during training and utilized during testing. Mean classwise intersection over union (mIoU) and overall pointwise accuracy (OA) are used as evaluation metrics.

SUN RGBD [38]. The dataset consists of 5K RGB-D training photos annotated with amodal-oriented 3D bounding boxes for 37 object types. We follow the standard data splits for the dataset. We adhere to standard data processing procedures of VoteNet [27] and use mean average precision (mAP) at different IoU thresholds, i.e., 0.25 and 0.5 to report the results for SUN RGBD.

5.2. Implementation Details

Searching space. We design a large search space that includes several variable factors in Res-SA building layers: neighbour points number, group radius, Res-MLP layer numbers in Φ_{mid} and Φ_{post}, reduction rate ϵ_1, expansion rate ϵ_2, and output channels. We also search the channel of stem MLP. Note that each task has its own search space. The detailed description of the searching space for Poly-PC (base) and Poly-PC (large) is given in Appendix B.

SuperNet training. Poly-PC is end-to-end optimized by using the following settings: AdamW optimizer with weight decay 0.05, initial learning rate 0.008 with cosine annealing, a batch size of 16 for ModelNet40, 8 for S3DIS, and 8 for SUN RGBD. We train Poly-PC for 50k iterations total with 24 Tesla V100 GPUs, that is, classification, segmentation and detection take up 8 GPUs respectively. For classification and segmentation, we use the same head with PointNet++ [30]. For detection, we adopt the same head with VoteNet [27]. The backbone has four Res-SA layers. We use the same data augmentations provided in PointNet++ for 3D shape classification and semantic segmentation, and the same data augmentations provided in VoteNet for 3D object detection. Note that we place the upsampling (i.e., feature propagation [30]) layer in the head of each task and treat them as task-specific parameters.

Evolution search. The evolution search algorithm is implemented by using the same protocol as SPOS [10]. The population size is set to 50 and the number of generations is set at 20. In each generation, the top 10 architectures are selected as the parents for the generation of offspring networks through mutation and crossover. The detailed implementation is given in Appendix C.2.

5.3. Comparing with State-of-the-art Methods

We compare Poly-PC against baselines and the state-of-the-arts on ModelNet40, S3DIS, and SUN RGBD datasets. Results are summarized in Tab. 1, Tab. 2, and Tab. 3.

Method	Points	FLOPs (G)	Params (M)	OA
PointCNN [16]	1k	1.6	0.6	92.2
PointConv [43]	1k	-	18.6	92.5
Kpconv [39]	7k	1.7	15.2	92.9
DGCNN [42]	1k	4.8	1.8	92.9
DeepGCN [15]	1k	3.9	2.2	93.6
ASSANet [31]	1k	2.4	3.6	92.4
ASSANet-L [31]	1k	28.9	118.4	92.9
PointMLP [22]	1k	31.4	12.6	93.7
Point Trans. [54]	1k	9.4	13.9	93.7
PointNet++ [30]	1k	3.2	1.5	90.7
PointNet++ [30]	5k	3.2	1.5	91.9
Poly-PC (base)	1k	2.9	1.9	92.6 (+1.9)
Poly-PC (large)	1k	5.9	5.5	93.7 (+3.0)

Table 1. Shape classification results on the ModelNet40 dataset.

ModelNet40. The results are presented in Tab. 1. When using 1k points as input, Poly-PC (base) surpasses PointNet++ [30] by 1.9 units in overall accuracy with a decrease of 0.3G FLOPs and an increase of 0.4M number of parameters. Moreover, Poly-PC (large) delivers on par accuracy as the state-of-the-art methods DEEPGCN [15], PointMLP [22] and Point Transformer [54] with comparable FLOPs and parameters.

S3DIS. Tab. 2 compares Poly-PC (base/large) with PointNet++ [30] and the state-of-the-arts on S3DIS dataset. Poly-PC (base) outperforms PointNet++ by 6.1 units in mIoU and 2.4 units in overall accuracy with similar FLOPs and parameters. Poly-PC (large) also achieves competitive results compared with the state-of-the-art methods KPConv [39], ASSANet-L [31] and PointNext-B [32] under fewer FLOPs and parameters. In this work, we investigate the usage of a framework to execute multiple point cloud tasks concurrently, which has significant implications for final model deployment, despite the fact that Poly-PC yields lower segmentation accuracy than current state-of-the-arts designed deliberately for segmentation task.

SUN RGBD. We evaluate Poly-PC against several competing approaches on SUN RGB-D dataset for 3D object detection. Results are summarized in Tab. 3. Poly-PC (base)

Method	Flops (G)	Params (M)	OA	mIoU
PointCNN [16]	-	0.6	85.9	57.3
PCCN [2]	7.3	5.4	-	58.3
PAT [47]	-	6.1	-	60.1
Kpconv [39]	2.1	14.9	-	67.1
ASSANet [31]	2.5	2.4	-	63.0
ASSANet-L [31]	36.2	115	-	66.8
Point Trans. [54]	7.8	5.6	90.8	70.4
PointNeXT-S [32]	3.6	0.8	87.9	63.4
PointNeXT-B [32]	8.8	3.8	89.4	67.5
PointNet++* [30]	1.0	1.0	85.8	56.9
Poly-PC (base)	1.0	1.0	88.2 (+2.4)	63.0 (+6.1)
Poly-PC (large)	5.6	5.6	89.5 (+3.7)	66.0 (+9.1)

Table 2. Semantic segmentation results on the S3DIS dataset, evaluated on area 5.

achieves 62.3 mAP@0.25 and 40.2 mAP@0.5, exceeding the baseline VoteNet [27] by 3.2 units under mAP@0.25 and 4.4 units under mAP@0.5 respectively. Moreover, Poly-PC (large) achieves 63.5 on mAP@0.25 and 41.9 mAP@0.5, which outperforms most previous state-of-the-arts that only use the point cloud. We do not design sophisticated head for detection task compared with Group-free [21] and RBGNet [41], but achieve comparable performance with them, demonstrating that the optimal backbone can also enhance performance.

Overall performance. In this part, we compare the overall performance of Poly-PC under such three tasks with the baseline (i.e., PointNet++ and VoteNet), along with the total parameters for jointly optimizing these tasks. Notably, we integrate PointNet++ and VoteNet together as the whole baseline of Poly-PC since they both leverage the SA layer as backbone. We also denote Poly-PC* that accommodates classification and segmentation task to compare with current state-of-the-arts that only validate performance on such two task. We define S-Score as the proxy measure of overall performance derived by linear sum of all task performances. As shown in Tab. 4, Poly-PC (base) outperforms the baseline by 10 units in terms of S-Score, demonstrating the effectiveness of provided techniques. Poly-PC* (base/large) also achieves equivalent performance compared to the current state-of-the-art, which solely verifies performance on classification and segmentation tasks. Specifically, Poly-PC* (base) exceeds ASSANet [31] 0.2 unit in S-Score, while containing over 3M fewer parameters. Poly-PC* (large) also yields nearly the same performance as Kpconv [39], but with almost 20M fewer parameters.

5.4. Ablation Study

We first compare the results of jointly training these tasks with those of individually training these tasks to demonstrate the efficacy of Poly-PC. Then, We discuss the influence of different proposed modules (i.e., Res-SA layer, weight-entanglement-based one-shot NAS and TGB algo-

Method	Flops (G)	Params (M)	mAP@0.25	mAP@0.5
F-PointNet [28]†	-	11.8	54.0	-
ImVoteNet* [26]†	241	42.5	64.5	38.6
3Detr [24]	9.8	7.1	59.1	32.7
MLCVNe [46]	7.2	1.2	59.8	-
H3DNet [53]	14.5	6.3	60.1	39.0
BRNet [7]	8.0	3.2	61.1	43.7
VENet [45]	30.3	4.9	62.5	39.2
Group-free* [21]	11.2	19.8	63.0	45.2
RBGNet [41]	3.5	2.2	64.1	47.2
VoteNet* [27]	5.8	1	59.1	35.8
Poly-PC (base)	6.1	1	62.3 (+3.2)	40.2 (+4.4)
Poly-PC (large)	18.8	7.8	63.5 (+4.4)	41.9 (+6.1)

Table 3. 3D object detection results on the SUN-RGBD val set. Note that † means it uses RGB as addition inputs and our method is geometric only.

Method	OA	mIoU	mAP@0.25	S-Score	Params (t)
PointNet++ & VoteNet	91.9	56.9	59.1	207.9	3.5M
Poly-PC (base)	92.6	63.0	62.3	217.9	3.4M
Poly-PC (large)	93.7	66.0	63.5	223.2	13.7M
PointCNN [16]	92.2	57.3	-	149.5	1.2M
Kpconv [39]	92.9	67.1	-	160.0	30.1M
ASSANet [31]	92.4	63.0	-	155.4	6.0M
Poly-PC* (base)	92.6	63.0	-	155.6	2.7M
Poly-PC* (large)	93.7	66.0	-	159.7	8.5M

Table 4. The overall performance of Poly-PC with baseline and other state-of-the-arts. Params (t) denotes the total parameters.

rithm) on the results, followed by an in-depth examination of each module's internal design. Notably, as the general one-shot NAS methods can not allow parameter sharing among tasks, we do not compare proposed weight-entanglement-based one-shot NAS with them.

Joint training vs individual training. We train the optimal subnet searched by Poly-PC for each task from scratch and compare the results with Poly-PC for all tasks. As illustrated in Tab. 5, performance of subnets derived by Poly-PC can achieve competitive results compared with individually training for each task, slightly lower in classification and segmentation while higher in detection. Moreover, a large portion parameters of the subnets in Poly-PC are shared for all tasks, which is more storage-efficient. The implementation details of the baseline is given in Appendix F.

The effect of different proposed modules. We start with the baseline that uses the naive SA layer in Poly-PC to train multiple tasks. The results are summarized in Tab. 6. The second row shows that the proposed Res-SA layer improves the performance from 59.5 to 60.3 for segmentation and 60.4 to 60.8 for detection. Including the weight-entanglement-based one-shot NAS technique into Poly-PC further improves the results to 92.3, 61.1 and 61.6 for classification, segmentation and detection respectively. Then, integrating attention module into Poly-PC further improves the results to 92.4, 61.9 and 61.8, respectively. Finally, as expected, when equipped with all modules, the results are improved to 92.6, 63.0 and 62.3, respectively, demonstrating the validation of our proposed modules.

The naive SA vs residual SA. A straightforward baseline to Res-SA is that using naive set abstraction layer in the backbone. We conduct experiments by using the

*We report the results of MMDetection3D (https://github.com/openmmlab/mmdetection3d), which are better than the official paper.

	Training scopes	cls	seg	det	Params (b)
Poly-PC (base)	individual	**93.0**	**63.2**	61.9	1.323M
Poly-PC (base)	joint & nas	92.6	63.0	**62.3**	**0.877M**
Poly-PC (large)	individual	**93.8**	**67.1**	63.2	15.677M
Poly-PC (large)	joint & nas	93.7	66.0	**63.5**	**10.443M**

Table 5. The effect of joint training or individual training. Params (b) means that the total parameters of all tasks in the backbone network.

	Res-SA	NAS	Att	TGB	cls	seg	det
Baseline					91.9	59.5	60.4
EXP1	✓				91.9	60.3	60.8
EXP2	✓	✓			92.3	61.1	61.6
EXP3	✓	✓	✓		92.4	61.9	61.8
EXP4	✓	✓	✓	✓	**92.6**	**63.0**	**62.3**

Table 6. The effect of proposed modules in Poly-PC (base). Att means proposed attention module.

	Res-SA	SA	cls	seg	det
base		✓	92.3	61.8	59.2
base	✓		92.6	63.0	62.3
large		✓	92.9	63.4	61.7
large	✓		**93.7**	**66.0**	**63.5**

Table 7. Performance of Poly-PC with SA and Res-SA layer.

GMH	GDH	PCGrad	DWA	cls	seg	det
		✓		92.3	61.3	62.0
			✓	92.8	61.8	61.8
✓				92.4	62.2	61.7
✓	✓			**92.6**	**63.0**	**62.3**

Table 8. Performance of Poly-PC (base) with different algorithms to tackle negative transfer.

naive SA layer to replace the Res-SA layer in Poly-PC (base and large) while keeping other configurations (i.e., weight-entanglement-based one-shot NAS and TGB algorithm, etc.). As shown in Tab. 7, naive width scaling and depth scaling in SA layer can not result in a discernible improvement in accuracy, primarily due to gradient vanishment and overfitting. However, Poly-PC (large) achieves much higher performance than these naive scaling strategies, mainly due to the residual connection, bottleneck and inverted bottleneck designs in Res-SA layer.

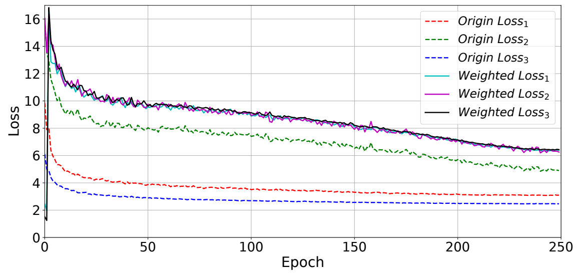


Figure 3. Illustration of task loss scales by applying gradient-magnitude homogenization of TGB or not.

The effect of TGB algorithm. In this part, we compare our TGB algorithm with other typical algorithms (e.g., PCGrad [50] and Dynamic Weight Average [19]) used to tackle negative transfer in vision tasks. Since our TGB algorithm solves negative transfer by homogenizing task

	cls	seg	det	obj. cls (OA)	Params (e)
PointNet++ [30]	90.7	56.9	-	77.9	100%
Poly-PC (base)	92.6	63.0	62.3	86.8	88%
Poly-PC (large)	93.7	66.0	63.5	87.9	50%

Table 9. Incremental learning of Poly-PC. Params(e) means that the percentage of additional parameters that each method needs when generalizing to a new task.

gradients in terms of both magnitudes and directions, we also decouple our algorithm into two components: gradient-magnitude homogenization (GMH) and gradient-direction homogenization (GDH). As depicted in Tab. 8, our TGB algorithm achieves higher performances compared with PCGrad and DWA algorithm, further demonstrating the validation of our algorithm. Fig. 3 shows an example of loss trend when the gradient-magnitude homogenization of TGB is applied or not. *Origin* $Loss_1$, $Loss_2$ and $Loss_3$ yield different loss scales, causing gradient-magnitude differences, while *weighted* $Loss_1$, $Loss_2$ and $Loss_3$ are approximately equalized by using the gradient-magnitude homogenization rule in Eq. (6).

5.5. Incremental learning of Poly-PC

When a new task is presented to Poly-PC, Poly-PC only needs to use the task-specific parameters to fit this task while freezing all task-shared parameters. Thus, Poly-PC is designed to enable incremental learning. When Poly-PC is trained on above three datasets, we add the real-world object classification dataset ScanObjectNN [40] into Poly-PC. The training recipe and details of ScanObjectNN is given in Appendix D. As shown in Tab. 9, Poly-PC (base/large) achieve 86.8/87.9 overall accuracy (OA) for ScanObjectNN with fewer parameter proportions, further demonstrating that Poly-PC can avoid catastrophic forgetting and achieve genuinely incremental learning.

6. Conclusion

In this work, we introduce Poly-PC, a unified framework for multi-task learning on point cloud with multiple dataset domains. Poly-PC achieves high performance for all tasks and keeps storage-efficient for model deployment through model design (i.e., Res-SA layer and weight-entanglement-based one shot NAS) and model optimization (i.e., task-prioritization-based gradient balance algorithm). We also demonstrate that Poly-PC can adapt to a new task by small task-specific parameters while keeping the same performance for other tasks. Comprehensive experiments show that Poly-PC delivers superior performance on multiple point cloud tasks.

Acknowledgement. This work was in part by National Natural Science Foundation of China under Grant 62176072 and 62073101, in part by by the National Natural Science Foundation of China (No. 62071104, No. U2233209 and No. U2133211).

References

[1] Iro Armeni, Ozan Sener, Amir R Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. 3d semantic parsing of large-scale indoor spaces. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1534–1543, 2016. 6

[2] Matan Atzmon, Haggai Maron, and Yaron Lipman. Point convolutional neural networks by extension operators. *ACM Transactions on Graphics*, 37(4), 2018. 7

[3] R Caruana. Multitask learning: A knowledge-based source of inductive bias1. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 41–48. Citeseer, 1993. 1

[4] Minghao Chen, Houwen Peng, Jianlong Fu, and Haibin Ling. Autoformer: Searching transformers for visual recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12270–12280, 2021. 1, 4

[5] Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *International conference on machine learning*, pages 794–803. PMLR, 2018. 2

[6] Zhao Chen, Jiquan Ngiam, Yanping Huang, Thang Luong, Henrik Kretzschmar, Yuning Chai, and Dragomir Anguelov. Just pick a sign: Optimizing deep multitask models with gradient sign dropout. *Advances in Neural Information Processing Systems*, 33:2039–2050, 2020. 2

[7] Bowen Cheng, Lu Sheng, Shaoshuai Shi, Ming Yang, and Dong Xu. Back-tracing representative points for voting-based 3d object detection in point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8963–8972, 2021. 7

[8] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167, 2008. 1

[9] José GC de Souza, Matteo Negri, Elisa Ricci, and Marco Turchi. Online multitask learning for machine translation quality estimation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 219–228, 2015. 1

[10] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. In *European conference on computer vision*, pages 544–560. Springer, 2020. 4, 6

[11] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018. 4

[12] Adrián Javaloy and Isabel Valera. Rotograd: Gradient homogenization in multitask learning. In *International Conference on Learning Representations*, 2022. 1, 2

[13] Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7482–7491, 2018. 1, 2

[14] Jae-Han Lee, Chul Lee, and Chang-Su Kim. Learning multiple pixelwise tasks based on loss scale balancing. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5107–5116, 2021. 1

[15] Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9267–9276, 2019. 6

[16] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. Pointcnn: Convolution on x-transformed points. *Advances in neural information processing systems*, 31, 2018. 6, 7

[17] Ying Li, Lingfei Ma, Zilong Zhong, Dongpu Cao, and Jonathan Li. Tgnet: Geometric graph cnn on 3-d point cloud segmentation. *IEEE Transactions on Geoscience and Remote Sensing*, 58(5):3588–3600, 2019. 2

[18] Liyang Liu, Yi Li, Zhanghui Kuang, J Xue, Yimin Chen, Wenming Yang, Qingmin Liao, and Wayne Zhang. Towards impartial multi-task learning. ICLR, 2021. 2

[19] Shikun Liu, Edward Johns, and Andrew J Davison. End-to-end multi-task learning with attention. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1871–1880, 2019. 1, 2, 4, 8

[20] Yongcheng Liu, Bin Fan, Shiming Xiang, and Chunhong Pan. Relation-shape convolutional neural network for point cloud analysis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8895–8904, 2019. 2

[21] Ze Liu, Zheng Zhang, Yue Cao, Han Hu, and Xin Tong. Group-free 3d object detection via transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2949–2958, 2021. 7

[22] Xu Ma, Can Qin, Haoxuan You, Haoxi Ran, and Yun Fu. Rethinking network design and local geometry in point cloud: A simple residual mlp framework. In *International Conference on Learning Representations*, 2021. 2, 6

[23] Kevis-Kokitsi Maninis, Ilija Radosavovic, and Iasonas Kokkinos. Attentive single-tasking of multiple tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1851–1860, 2019. 2

[24] Ishan Misra, Rohit Girdhar, and Armand Joulin. An end-to-end transformer model for 3d object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2906–2917, 2021. 7

[25] Houwen Peng, Hao Du, Hongyuan Yu, Qi Li, Jing Liao, and Jianlong Fu. Cream of the crop: Distilling prioritized paths for one-shot neural architecture search. *Advances in Neural Information Processing Systems*, 33:17955–17964, 2020. 4

[26] Charles R Qi, Xinlei Chen, Or Litany, and Leonidas J Guibas. Imvotenet: Boosting 3d object detection in point clouds with image votes. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4404–4413, 2020. 7

[27] Charles R Qi, Or Litany, Kaiming He, and Leonidas J Guibas. Deep hough voting for 3d object detection in point

clouds. In *proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9277–9286, 2019. 6, 7

[28] Charles R Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J Guibas. Frustum pointnets for 3d object detection from rgb-d data. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 918–927, 2018. 7

[29] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017. 1, 2

[30] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017. 2, 3, 6, 7, 8

[31] Guocheng Qian, Hasan Hammoud, Guohao Li, Ali Thabet, and Bernard Ghanem. Assanet: An anisotropic separable set abstraction for efficient point cloud representation learning. *Advances in Neural Information Processing Systems*, 34:28119–28130, 2021. 6, 7

[32] Guocheng Qian, Yuchen Li, Houwen Peng, Jinjie Mai, Hasan Hammoud, Mohamed Elhoseiny, and Bernard Ghanem. Pointnext: Revisiting pointnet++ with improved training and scaling strategies. 6, 7

[33] Shi Qiu, Saeed Anwar, and Nick Barnes. Geometric back-projection network for point cloud classification. *IEEE Transactions on Multimedia*, 24:1943–1955, 2021. 2

[34] Sebastian Ruder. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*, 2017. 1

[35] Ozan Sener and Vladlen Koltun. Multi-task learning as multi-objective optimization. *Advances in neural information processing systems*, 31, 2018. 2, 4

[36] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. Pointrcnn: 3d object proposal generation and detection from point cloud. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 770–779, 2019. 1

[37] Ayan Sinha, Zhao Chen, Vijay Badrinarayanan, and Andrew Rabinovich. Gradient adversarial training of neural networks. *arXiv preprint arXiv:1806.08028*, 2018. 2

[38] Shuran Song, Samuel P Lichtenberg, and Jianxiong Xiao. Sun rgb-d: A rgb-d scene understanding benchmark suite. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 567–576, 2015. 6

[39] Hugues Thomas, Charles R Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J Guibas. Kpconv: Flexible and deformable convolution for point clouds. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6411–6420, 2019. 1, 6, 7

[40] Mikaela Angelina Uy, Quang-Hieu Pham, Binh-Son Hua, Thanh Nguyen, and Sai-Kit Yeung. Revisiting point cloud classification: A new benchmark dataset and classification model on real-world data. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1588–1597, 2019. 8

[41] Haiyang Wang, Shaoshuai Shi, Ze Yang, Rongyao Fang, Qi Qian, Hongsheng Li, Bernt Schiele, and Liwei Wang. Rbgnet: Ray-based grouping for 3d object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1110–1119, 2022. 7

[42] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)*, 38(5):1–12, 2019. 6

[43] Wenxuan Wu, Zhongang Qi, and Li Fuxin. Pointconv: Deep convolutional networks on 3d point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9621–9630, 2019. 6

[44] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015. 6

[45] Qian Xie, Yu-Kun Lai, Jing Wu, Zhoutao Wang, Dening Lu, Mingqiang Wei, and Jun Wang. Venet: Voting enhancement network for 3d object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3712–3721, 2021. 7

[46] Qian Xie, Yu-Kun Lai, Jing Wu, Zhoutao Wang, Yiming Zhang, Kai Xu, and Jun Wang. Mlcvnet: Multi-level context votenet for 3d object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10447–10456, 2020. 7

[47] Jiancheng Yang, Qiang Zhang, Bingbing Ni, Linguo Li, Jinxian Liu, Mengdie Zhou, and Qi Tian. Modeling point clouds with self-attention and gumbel subset sampling. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3323–3332, 2019. 7

[48] Jiahui Yu and Thomas S Huang. Universally slimmable networks and improved training techniques. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1803–1811, 2019. 2, 4

[49] Jiahui Yu, Pengchong Jin, Hanxiao Liu, Gabriel Bender, Pieter-Jan Kindermans, Mingxing Tan, Thomas Huang, Xiaodan Song, Ruoming Pang, and Quoc Le. Bignas: Scaling up neural architecture search with big single-stage models. In *European Conference on Computer Vision*, pages 702–717. Springer, 2020. 1, 4

[50] Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. *Advances in Neural Information Processing Systems*, 33:5824–5836, 2020. 2, 5, 8

[51] Min Zhang, Haoxuan You, Pranav Kadam, Shan Liu, and C-C Jay Kuo. Pointhop: An explainable machine learning method for point cloud classification. *IEEE Transactions on Multimedia*, 22(7):1744–1755, 2020. 2

[52] Zhenyu Zhang, Zhen Cui, Chunyan Xu, Yan Yan, Nicu Sebe, and Jian Yang. Pattern-affinitive propagation across depth, surface normal and semantic segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4106–4115, 2019. 1

[53] Zaiwei Zhang, Bo Sun, Haitao Yang, and Qixing Huang. H3dnet: 3d object detection using hybrid geometric primitives. In *European Conference on Computer Vision*, pages 311–329. Springer, 2020. 7

[54] Hengshuang Zhao, Li Jiang, Jiaya Jia, Philip HS Torr, and Vladlen Koltun. Point transformer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 16259–16268, 2021. 1, 6, 7