AutoGPart: Intermediate Supervision Search for Generalizable 3D Part Segmentation
Supplementary Materials

1. Overview

The appendix aims at providing more details on the method design (Sec. 2), experimental settings (Sec. 3), and more experimental results (Sec. 4).

2. Method Details

In this section, we give some explanations for some details of the proposed method that are not stated in the main paper.

2.1. Supervision Feature Structure

This section gives more details about designed structural model for generating ground-truth part-aware geometric features w.r.t. further explanations for each component, operation cells and the rationality of the design.

Input features. Input features for each point are ordered part-aware feature sets formed by different kinds of geometric features from points having the same part labels with the target point. A maximum radius is set between the sampled points and the target point considering the limited receptive field of some point-cloud processing backbones such as DGCNN. This value is set to 0.5 in our implementation.

The full set of ordered input feature sets are formed by first sampling such part-aware feature sets from input geometric features such as coordinate vectors, then expanding the resulting sets by adding feature sets calculated by simple point-level operations among them to it. For example, suppose each point i has two different input geometric features: the coordinate vector $\vec{p}_i$ and the normal vector $\vec{n}_i$. Their ordered respective part-aware feature sets for point i, constructed by sampling same kind of features from points having the same part labels with it, are denoted as $\mathcal{P}_i$ and $\mathcal{N}_i$ respectively. In the first step, we can get $\{\mathcal{P}_i, \mathcal{N}_i\}$ for point i. Then after expanding the set, we get $\{\mathcal{P}_i, \mathcal{N}_i, \mathcal{P}_i - \mathcal{N}_i ...\}$, where $\mathcal{P}_i - \mathcal{N}_i$ refers to a feature set formed by element-wise minus between $\mathcal{P}_i$ and $\mathcal{N}_i$. An ordered part-aware geometric input feature set such as $\mathcal{P}_i$ is referred as an "operant" in the operation tree. In practice, an ordered feature set is formed to a matrix with each of its line a feature vector of a point, thus P_i for $\mathcal{P}_i$ and N_i for $\mathcal{N}_i$. If each point has two different geometric features, the full set of input part-aware geometric matrices contains 7 elements: $\{P_i, N_i, P_i \cdot N_i, P_i + N_i, P_i - N_i, N_i - P_i, \text{cross\_product}(N_i, P_i)\}$. If each point has one geometric feature, the full set of input part-aware geometric matrices contains 4 elements: $\{P_i, 2P_i, P_i^2, -P_i\}$.

Operators. Operators are introduced to transform the input features step by step for the output intermediate supervision feature. Such operators include grouping operators, point-level unary operators, and point-level binary operators, as summarized in Table 1. Then the grouping operator set, the unary operator set, and the binary operator set are further referred as $\mathcal{G}, \mathcal{U}$, and $\mathcal{B}$ respectively.

Some operators may seem confusing, for which we give them some brief explanations as follows:

- Orthogonalize: Given a matrix M, first calculate its signular vectors and singular values by $U, S, V^T = \text{SVD}(M)$, then take the matrix-vector multiplication between two singular vector matrices for the result: $\text{Orthogonalize}(M) = UV^T$.
- Cartesian Product: Given two feature sets from one point, $\mathcal{S}_1$ and $\mathcal{S}_2$, calculate the cartesian product between them by $\mathcal{S}_1 \times \mathcal{S}_2 = \{(s_i, s_j) | s_i \in \mathcal{S}_1, s_j \in \mathcal{S}_2\}$.

Operation cells. We introduce several different kinds of operation cells with fixed operator combinations to encourage some fixed operation sequences. Details of such operator combinations for operation cells in different levels are listed as follows:

- Level-3 cells (Top level cells): a grouping operator followed by a unary operator.
- Level-2 cells: a grouping operator followed by a uanry operator.
- Level-1 cells: an operant followed by a uanry operator.

Justification for the designed operation cells. In the design of operation cells, we set some fixed operator sequences for them such as a grouping operator followed by a unary operator. The design of the structure of operation cells is heuristic but also reasonable. For example, a unary

Table 1. Candidates of each kind of feature transformation operator. "SVD" denotes "Singular Value Decompse", "Identity" refers to no operations, where the input ordered feature set (matrix) is not transformed by any unary operator.

Operators	Choices
Grouping operators	Sum, Average, Maximum, SVD
Binary operators	Add, Minus, Multiply, Cross Product, Cartesian Product, Matrix-vector Product
Unary operators	Identity, Square, Double, Negative, Orthogonalize, Inverse

Table 2. Examples of hand-crafted ground-truth features used in previous works. Matrices in the table are part-level matrices formed by corresponding features from points in a part. For notations used, F is the flow matrix, P is the coordinate matrix, N is the normal matrix. $\overline{M}$ denotes the centralized matrix of M. Note that the geometric feature listed in the left column of the table may not be the exact feature output by the calculation process listed in its corresponding right cell, but a part of the output feature such as a row vector of the matrix.

Feature	Calculation process
Rotation matrix	$\text{orth}(\text{sum}(\text{cartesian}(P+F, P)))$
Cone apex	$\text{sum}(\text{cartesian}(N^{+T}, \text{rowsum}(N \cdot P)))$
Cylinder axis	$\text{svd}(\overline{N})$
Sphere center	$\text{sum}(\text{cartesian}((-2\overline{P})^{+T}, \text{rowsum}(P^2)))$
Sphere radius	$\text{sqrt}(\text{mean}(\text{P} - \vec{c})^2)$
Plane normal	$\text{svd}(\overline{P})$

operator following a grouping operator can further transform the grouped feature, extracting part-level information from it. Such calculation routines are common in the calculation process of some features such as the rotation matrix R. To be more specific, several steps in the calculation process of R contain a combination of "Centralize" and "Sum", where the former is a unary operator and the later is a grouping operator. This is only one possibility we explored in our practice, we believe there could be many other strategies to design operation cells (e.g. other computing routines) or just constructing the operation tree without introducing operation cells. How to design suitable operation cells, or how to add proper constraints for operation combinations in the operation space is interesting and worth exploring.

Connections with hand-crafted geometric features. The constructed supervision feature space contains many hand-crafted geometric features used in previous works. Some examples are summarized in Table 2. The calculation process uses matrix forms of the ordered geometric feature sets.

2.2. Supervision Feature Distribution Space

This section aims for some further explanations of the constructed supervision feature distribution space w.r.t. how we decompose it into a tree-structured distribution space and the conditional sampling process performed on the distribution tree.

Decomposed tree-structured distribution set. Instead of using one single distribution, we decompose the total/joint distribution into a tree-structured distributions set to better model the generation process of the operation tree. Specifically, we use a distribution cell to depict the space of an operation cell, where the operator sequence in the operation cell is generated by a set of conditional operator distributions. For instance, a distribution cell for an operation cell with the ordered operator sequence [a grouping operator, a unary operator] is composed of a grouping operator distribution and $|\mathcal{G}|$ unary operator distributions. Each possible grouping operator has its own unary operator distribution. Thus, the distribution cell for this specific operation cell can be organized into a distribution tree with the grouping operator distribution as the top distribution and a unary operator distribution attached to each element in the grouping operator.

To model possible connections between cells from adjacent two levels, we introduce a connection distribution for each possible operator sequence of the upper-level cell to depict each possible connection and the operator that should be used for connection. For example, a Level-3 cell can be connected with two Level-1 cells with each kind of binary operator, or a Level-2 cell and a Level-1 cell with each kind of binary operator, or a Level-1 cell and a Level-2 cell with each kind of binary operator, or a single Level-1 cell with no binary operator. Thus, the total number of possible connections is $3|\mathcal{B}| + 1$. A distribution containing elements of this number is introduced for each possible operator sequence of the upper-level cell.

Conditional operation tree sampling. After the distribution space has been constructed, a conditional sampling process is adopted to sample an operation tree from the space. Specifically, to sample an operation cell from its respective distribution cell, its top operator is sampled at first. Then, the distribution for the following level operator of this top operator is chosen to continue the sampling process. Specifically, the sampling process for sampling an operation sequence $[\text{op}_1, \text{op}_2, ..., \text{op}_k]$ is conducted by: Sample op_k, conditioned on op_k and sample op_{k-1}, ..., conditioned on $\text{op}_k, \text{op}_{k-1}, ..., \text{op}_2$ and sample op_1. The probability density value of the sequence is then calculated by:

$$p([\text{op}_1, \text{op}_2, ..., \text{op}_k]) = \quad (1)$$
$$p(\text{op}_k)p(\text{op}_{k-1}|\text{op}_k)...p(\text{op}_1|[\text{op}_k, ..., \text{op}_2]). \quad (2)$$

Similarly, to sample an operation tree, the top operation cell is first sampled. Then, a connection is sampled conditioned on the sampled operation cell structure. The sampled connection determines what distribution cells to continue the sampling process and what binary operator to use for connection if needed. Thus, the operation tree can be sampled by such a top-down manner.

2.3. Supervision Feature Distribution Learning

In this section, we talk about how to learn parameters for distributions in the supervision feature space, including the supervision feature sampling process, its cross-domain generalization ability evaluation and the distribution updating process.

Supervision feature sampling. A supervision feature is generated by first sampling an operation tree from the constructed distribution space and then pass input part-aware geometric features of each point through the operation tree to get its calculated part-aware geometric feature. The operation tree is sampled by the conditional sampling process from the distribution space as state above.

Cross-validation setting. Each selected supervision is put under a cross-validation process to estimate its cross-domain generalization ability. To cross-validate the generalization ability of the selected supervision, we first split the train dataset into K subsets with a relatively large distribution shift across them. Then, the selected supervision is tested on each $K-1:1$ train-validation fold. The average value between the metric on the validation set and the train set is taken as the estimated score for its generalization ability. In the supervision search process, the generalization score is further taken as the reward for the selected supervision and used for the following supervision distribution space update process. The cross-validation procedure for generalization gap evaluation is summarized in Algorithm 1.

Supervision distribution space optimization. We optimize distributions in each layer related with the sampling process of the selected supervision via the REINFORCE algorithm. Optimizing those conditional distributions via REINFORCE results optimizing the overall/joint distribution via REINFORCE. The optimization strategy is derived from the REINFORCE algorithm, where each parameter w_i that counts in the sampling process is updated by minusing its corresponding $\Delta w_i = \alpha_i(r-b_i)e_i$, where $e_i = \partial \ln g / \partial w_i$, g is the probability density function. Thus, e_i for parameter w_i of a distribution can be derived as:

$$e_i = \frac{\partial g}{\partial w_i} \tag{3}$$

$$= \frac{\partial \ln(g(\mathcal{H})g(v|\mathcal{H})g(\mathcal{C}\setminus\{v\}|\mathcal{H})g(\mathcal{L}|\mathcal{H},\mathcal{C}))}{\partial w_i} \tag{4}$$

$$= \frac{\partial \ln(g(\mathcal{H}) + \partial \ln(g(v|\mathcal{H})) + \partial \ln(g(\mathcal{L}|\mathcal{H},v)))}{\partial w_i} \tag{5}$$

$$= \frac{\partial \ln(g(v|\mathcal{H})}{\partial w_i}, \tag{6}$$

where $g(\mathcal{H})$ is the probability density of values sampled in upper layers, $g(\mathcal{L}|\mathcal{H},\mathcal{C})$ is the probability density of values sampled in lower layers conditioned on values sampled in current layers $\mathcal{C}$ and those sampled in upper layers $\mathcal{H}$, which are not relevant with the sampling distribution in the current layer resulting value v. $\mathcal{H}$ denotes the set of operators in upper layers of the operation tree, $\mathcal{C}$ means the set of operators in the current layer of the operation tree, while $\mathcal{L}$ refers to the set of operators in lower layers. The conditional distribution for each sample $g(v|\mathcal{H})$ is a multinomial distribution, in our implementation, which can be simply calculated by the class function "log_prob" of the corresponding PyTorch Distribution module.

2.4. Greedy Supervision Feature Selection

This section aims for some further explanations of the unstated details of the greedy feature selection process.

After we have got the optimized supervision distribution set, we greedily select suitable supervisions from the optimized space. At most three supervisions are selected from the optimized space. To complete the supervision selection process, we first select K supervisions from the optimized space by randomly sampling from the optimized supervision space. Then the performance of such K supervisions are evaluated under a simulated domain-shift setting. From the ranked set, top 2 supervisions are selected to combine with top $K/2$ supervisions and form the second supervision set. Supervision combinations in the second supervision set are further evaluated and ranked. Then, top 3 supervision combinations containing 2 supervisions are selected to combine with top $K/3$ single supervisions from the first supervision set. The resulting combination set is further evaluated and ranked. The supervision combination that achieves the best estimated performance is selected for further evaluation. The performance of the selected supervision combination is regarded as the performance of the optimized supervision distribution space. The training stage of the supervision evaluation process is referred as the "regular training stage". This greedy selection process is also summarized in Algorithm 3.

Algorithm 2 SortByCrossVal.

Input: The model $\mathcal{M}$; Split train set $\mathcal{S}_{sp} = \{\mathcal{S}_1, \mathcal{S}_2, ..., \mathcal{S}_k\}$; A set of intermediate supervision features $\mathcal{T} = \{t_1, t_2, ..., t_K\}$.
Output: Sorted intermediate supervision features $\mathcal{T}'$.

for $i = 1$ to $|\mathcal{T}|$ **do**
  $s \leftarrow$ Cross_Val($\mathcal{M}, \mathcal{T}_1[i], \mathcal{S}_{sp}$)
  $\mathcal{T}_1[i] \leftarrow (\mathcal{T}_1[i], s)$
$\mathcal{T}' \leftarrow$ sorted($\mathcal{T}$)
return $\mathcal{T}'$

Algorithm 1 Cross_Val. "TrainValidation($\cdot, \cdot, \cdot, \cdot$)" takes a model, the train dataset, the validation dataset and the number of training epochs n as input, trains the model for n epochs and returns the gap between the performance of the model achieved on the validation set and the training set at the best validation epoch.

Input: The model $\mathcal{M}$; Split train set $\mathcal{S}_{sp} = \{\mathcal{S}_1, \mathcal{S}_2, ..., \mathcal{S}_k\}$; Intermediate supervision feature t; Epochs for training n.
Output: Estimated generalization score which is actually the average generalization gap across all train-validation splits.

scores $\leftarrow [\,]$
for $i = 1$ to $|\mathcal{S}_{sp}|$ **do**
  $\mathcal{S}_{tr} \leftarrow \mathcal{S}_{sp} \setminus \{\mathcal{S}_i\}$
  $\mathcal{S}_{val} \leftarrow \{\mathcal{S}_i\}$
  $s_i \leftarrow$ TrainValidation$(\mathcal{M}, \mathcal{S}_{tr}, \mathcal{S}_{val}, n)$
  scores.append(s_i)
$\overline{s} \leftarrow$ mean(scores)
return $\overline{s}$

Algorithm 3 GreedySupervisionSelection.

Input: The model $\mathcal{M}$; Split train set $\mathcal{S}_{sp} = \{\mathcal{S}_1, \mathcal{S}_2, ..., \mathcal{S}_k\}$; A set of sampled intermediate supervision features $\mathcal{T} = \{t_1, t_2, ..., t_K\}$.
Output: Selected intermediate supervision feature set $\mathcal{T}_s = \{t_{s1}, ..., t_{sk}\}, 1 \leq k \leq 3$.

$\mathcal{T}_1 \leftarrow \mathcal{T}$
$\mathcal{T}_1 \leftarrow$ SortByCrossVal$(\mathcal{M}, \mathcal{T}_1, \mathcal{S}_{sp})$
$\mathcal{T}_2 \leftarrow \mathcal{T}_1[1:2] \times \mathcal{T}_1[1 : |\mathcal{T}_1|/2]$
$\mathcal{T}_2 \leftarrow$ SortByCrossVal$(\mathcal{M}, \mathcal{T}_2, \mathcal{S}_{sp})$
$\mathcal{T}_3 \leftarrow \mathcal{T}_2[1:3] \times \mathcal{T}_1[1 : |\mathcal{T}_1|/3]$
$\mathcal{T}_3 \leftarrow$ SortByCrossVal$(\mathcal{M}, \mathcal{T}_3, \mathcal{S}_{sp})$
$\mathcal{T}_s \leftarrow$ sorted$(\{\mathcal{T}_1[1], \mathcal{T}_2[1], \mathcal{T}_3[1]\})[1]$
return $\mathcal{T}_s$

3. Experimental Details

In this section, we provide some detailed information about experimental settings that are not stated in the main paper, including datasets used in each segmentation task, detailed settings for each stage, implementation for baseline models, etc.

3.1. Dataset

Mobility-based part segmentation. For the training dataset and auxiliary training dataset, we first infer part mobility information for parts in each shape. Then, during the training process, we generate shape pairs for training based on such part mobility information. For the training dataset, which is created from [15], containing 15,776 shapes from 16 categories, we first infer mobility information for parts in a shape heuristically based on their semantic labels such that the generated mobility information for the part can align better with real scenarios. Specifically, some heuristic constraints are added to infer mobility information for each part such as the chair back can rotate around the chair seat but not chair legs. For the auxiliary dataset created from PartNet [9], the mobility information for parts in a shape is also inferred heuristically where only some general motion rules are added, which means that parts of different semantic labels share the same motion rules. The test dataset used in our work is the same as the one used in [14], which is created from [3].

Primitive fitting. We use the same dataset as the one provided by [6], but adopt a different data splitting strategy such that it is more suitable to test a model's cross-domain generalization ability. Specifically, we split shapes via their primitive-type distributions. Primitive-type distribution reveals the ratio of each primitive-type in the shape calculated based on number of points belonged to each type of primitive. We first cluster all shapes into 7 clusters by K-Means++ [1], then merge them into 4 subsets with a relatively large distribution gap across them. Details of each cluster w.r.t. the average primitive-type distribution over all shapes, their respective train-test split, and total number of shapes contained in the cluster are summarized in Table 3. "Train_1", "Train_2", and "Train_3" are all used in the supervision search process and regular training process. In the supervision search process, such three train splits serve for distribution-shift simulation to cross-validate the cross-domain generalization ability of the selected supervision. In the regular training process, shapes in those three splits are merged together and further split into train-validation datasets via a ratio 9:1. The validation set is used for model selection.

Table 3. Details w.r.t. each cluster of the re-split dataset for the primitive fitting task. For abbreviations used, "Cluster ID." refers to "Cluster Index", "Dist." denotes "Distribution". Primitives in the distribution vector has the order (Plane, Sphere, Cylinder, Cone).

Split	Cluster ID.	#Shapes	Primitive-type Dist.
Train_1	1	1759	(0.227, 0.043, 0.453, 0.277)
	2	2005	(0.540, 0.003, 0.429, 0.028)
Train_2	3	5600	(0.098, 0.056, 0.771, 0.075)
Train_3	4	3944	(0.026, 0.012, 0.944, 0.018)
	5	220	(0.067, 0.011, 0.033, 0.890)
Test	6	1474	(0.908, 0.00, 0.067, 0.025)
	7	2195	(0.218, 0.166, 0.590, 0.026)

Semantic-based part segmentation. The dataset we use is the same as the one used in [7]. We only use the finest segmentation level for training and evaluation other than using all available levels as does in [7].

3.2. Experimental Settings

Supervision search. We set the learning rate to 0.001 without change during the whole stage. For mobility-based part segmentation, 100 automatic search epochs are conducted. For primitive fitting and semantic-based part segmentation, 30 automatic search epochs are conducted. For all three tasks, 4 supervisions sampled and evaluated in each epoch.

The supervision distribution space is optimized in each epoch. AdaM optimizer is used for segmentation networks for all three tasks, with $\beta = (0.9, 0.999), \epsilon = 10^{-8}$, and weight decay ratio set to 10^{-4}. Batch size is set to 36 for mobility-based part segmentation task, 2 for primitive fitting and semantic-based part segmentation using DGCNN, 8 those two tasks using PointNet++.

Regular training. We set the initial learning rate to 0.001, decayed by 0.7 when the model's performance on the validation set is improved for more than 20 epochs. AdaM optimizer is used for segmentation networks for all three tasks, with $\beta = (0.9, 0.999), \epsilon = 10^{-8}$, and weight decay ratio set to 10^{-4}. For mobility-based part segmentation networks, 400 epochs are performed with the epoch that the model achieves the best validation performance is take for inference. For primitive fitting and semantic-based part segmentation tasks, 200 epochs are performed using the same model selection strategy as the one for the mobility-based part segmentation networks. When using clustering-based segmentation module, 100 training epochs are conducted with the model achieves the lowest validation contrastive-style loss further used for inference. As for the cross-validation strategy used for estimating the effectiveness of the selected supervision in improving the network's domain generalization ability, two training datasets are used for cross-validation for the mobility-based part segmentation task, namely the training dataset and the auxiliary training dataset. While three training datasets are used for both primitive fitting and the semantic-based part segmentation. Three clusters out of four clusters are used for cross-validating in primitive fitting task. Three categories, including "Chair", "Lamp", and "StorageFurniture", are used for cross-validating semantic-based part segmentation.

As for the whole cross-validating process, we train the network on each train-validation split fold for one single epoch with intermediate supervisions added based on the selected supervision feature and the segmentation task related supervision optimized simultaneously. After that, the average metric gap across all train-validation splits is taken as the estimated generalization score for the selected supervision feature, which is also used as the reward score for further supervision feature distribution space update.

Ablation Study. For details of different ablated versions w.r.t. the supervision space design. "Less operants" denotes using a smaller input feature candidate set that is not expanded. For a set of input part-level matrix cadidates we used in the full regular searching process where the full version of the input feature set that is expanded from input features, only the set containing part and geometry-aware matrices formed from input features directly is used in this version. More specifically, in "Less operants" setting, we use $\{ P_i, N_i\}$ as the input feature set, while the full version contains 7 matrices. "Less unary operators" denotes the version where only a subset of unary operators is used as the unary operator set. Compared with the full version listed in Table 1, the one used in the ablated version is {Identity, Square, Double, Negative}. "Less binary operators" refers to the version where only a subset of binary operators is used as the binary operator set. Compared with the full version listed in Table 1, the one used in the ablated version is {Add, Minus, Multiply}. "Less tree height" means the maximum height of the operation tree, which is measured by the number of the connected operation cells. Compared with the one used in the full version that is set to 3, the maximum tree height in the ablated version is set to 2.

Models for comparison with HPNet. In the primitive fitting task, we develop two models to compare with HPNet fairly, considering the clustering-based network architecture used in HPNet that is different from the classification-based segmentation module used in our default setting to evaluate AutoGPart . Two models are designed by 1) replacing the clustering-based segmentation module used in HPNet with a classification-based segmentation module, denoted as "HPNet*"; 2) plug AutoGPart in the first learning stage of HPNet to search for useful intermediate supervisions that can help the network learn representations using more invariant features and avoid using shortcut features [2], denoted as "AutoGPart$_{\text{HPNet}}$". Following are some detailed settings for such two models. For HPNet*, the network is formed by replacing the clustering-based segmentation module with a classification-based segmentation module with supervisions added on the first learning stage kept. The model is trained and evaluated using the same setting as for our own model. That is, train the model for 200 epochs and select the best validation epoch for further evaluation. For AutoGPart$_{\text{HPNet}}$, we also adopt a two-stage training procedure. In the first stage, AutoGPart is applied to search for useful intermediate supervisions using the gap of the contrastive-style loss between the training dataset and the validation dataset across all train-validation splits as the reward. After the supervision distributions have been optimized, we greedily select what supervisions to use and plug them in the first learning stage of HPNet by optimizing such losses and other losses introduced in HPNet simultaneously. The resulting model optimized in the first learning stage is then taken for further evaluation using the clustering-based segmentation module.

Point-cloud processing backbones. PointNet++ used in our default setting is the same one as that used in SPFN [6]. DGCNN used in the default setting is the same one as that used in HPNet [13] for representation learning.

Input features. For semantic-based part segmentation task where per-point normal vector is not contained in input features, we estimate a normal vector for each point using

open3d [19]. For primitive fitting, input geometric features for each point i contain a coordinate vector $\vec{p}_i$ and a ground-truth normal vector $\vec{n}_i$. For mobility-based part segmentation task, input features for each point i are composed of a coordinate vector $\vec{p}_i$ and a flow vector $\vec{f}_i$. The flow vector is estimated according to two input shapes.

Baselines. For domain-agnostic baselines for general domain generalization problems, like MixStyle [18], Meta-learning [5], Gradient Surgery [8], we implement them for segmentation tasks carefully with reference to their released code. For mobility-based part segmentation task, Deep Part Induction [14] is a learning based segmentation network. Although the test dataset used in our model is the same as the one used in their work, we download the code, re-implement it using PyTorch, and further train it using our training dataset where each pair is generated on-the-fly from inferred meta-data for part mobility information. Note that the performance reported in the original paper (77.3% MIoU on the test set) is achieved using several iterations between flow estimation and part segmentation. The performance of the model using a single iteration is 63.1% as reported in their work. It is also different from the one we report, probably due to the different training dataset. The performance of other baselines such as JLinkage clustering (JLC) [17] and (Spectral Clustering) SC [11] are taken from [14] due to the same test datset.

For primitive fitting, SPFN [6] and HPNet [13] are two task-specific methods to solve the problem. We download the code of SPFN released by the author, carefully re-implement it using PyTorch and test the model on the same train-validation-test split as the one used for our model. For HPNet, we download the official implementation and test the model's performance on our train-validation-test split.

For the semantic-based part segmentation, Learning to Group [7] is a two-stage learning-based segmentation network with a representation learning stage and a reinforcement learning based strategy for part segmentation; SGPN [12] and GSPN [16] are also two task-specific segmentation strategies. WCSeg [4] is a traditional sgementation method. The performance of those methods are directly taken from [7] due to the same training and test dataset.

Time consumption. A training epoch for primitive fitting and semantic-based part segmentation takes the time varying roughly from 4 minutes to 10 minutes, while roughly from 1 minute to 3 minutes for the mobility-based part segmentation task, according to what intermediate supervisions used. An inference epoch takes about 2 minutes for primitive fitting and semantic-based part segmentation task, while about 40 seconds for the mobility-based segmentation task. On average, AutoGPart triples the train time of a part segmentation backbone (from 4.4hrs to 13.2hrs).

Software configurations. We use Python 3.8.8 and PyTorch 1.9.1 to write the main code framework. Other main packages used include torch_cluster 1.5.9, torch_scatter 2.0.7, horovod 0.23.0 for PyTorch, etc.

Hardware configuration. All training experiments, including supervision search stage and regular training stage, are conducted on 8 NVIDIA Geforce RTX 3090 GPUs in parallel. Experiments for inference stage is conducted on one single NVIDIA Geforce RTX 3090 GPU.

4. Additional Experimental Results

4.1. Supervision Distribution Space Evolution

In this section, we explore the evolution of the performance of the supervision distribution space during the supervision search process on the mobility-based part segmentation task. We take the optimized supervision distribution space at epoch 20,40,60,80,100 and evaluate their performance. Results are presented in Table 4. As shown in the table, the performance of the supervision distribution space has an increasing tendency during the optimization process, though not exactly monotonous. Such an observation indicates that: 1) During the optimization process, the distribution space is gradually optimized to prefer high-quality supervision features. 2) Both the supervision distribution space optimization and the supervision selection strategy contribute to good optimization combination that is further used in the following evaluation process. The effectiveness of the greedy supervision selection strategy as well as comparison with other selection process are discussed in the main paper.

Further, another direction worth exploring is the optimization strategy. We simply adopt a basic reinforcement learning based strategy to optimize the supervision distribution space, which may not be the optimal choice. Though our experimental results prove the effectiveness of such strategy, we did not design and compare other strategies in this work, more possible optimization methods are worth further exploring.

Table 4. Comparison of the optimized supervision distribution space across the supervision search process. The experiment is conducted on the mobility-based part segmentation task with DGCNN as its backbone.

Epoch	20	40	60	80	100
MIoU	0.696	0.724	0.687	0.733	0.738

4.2. Contrastive Learning based Part Segmentation

Our main experimental results have proved the effectiveness of HPNet [13] on the primitive fitting task. It is a carefully designed two-stage framework with a representation learning network that learns feature representations, parameters and other geometric features such as normal vectors and further hybrid such learned features for the following Mean-Shift clustering module. Such design achieves

impressive performance on the primitive fitting task, with MIoU performance 79.5%. It is not clear whether such network architecture is suitable for other two tasks. To apply them on the mobility-based part segmentation and semantic-based part segmentation, we abbreviate its first representation learning stage to only learn per-point representations optimized by a contrastive style loss and further fit the optimized representation to the clustering stage.

We conduct experiments by applying such contrastive learning based part segmentation strategy to the mobility-based part segmentation task and the semantic-based part segmentation task using PointNet++ [10] as the backbone. Results are summarized in Table 5. It can be inferred that although such contrastive based segmentation networks can work well on the primitive fitting task, it cannot get satisfactory results on the mobility-based part segmentation task or semantic-based part segmentation task using PointNet++ as the backbone. Possible reasons may include 1) Such network architecture is not a universal one for all segmentation tasks, considering that it is originally designed for the primitive fitting task. 2) This network architecture is not universally suitable for both PointNet++ and DGCNN. For comparison, our strategy is more universal compared with the contrastive learning based segmentation strategy.

Table 5. The performance of the contrastive learning based part segmentation networks ("Contrastive") on the mobility-based part segmentation task ("Mobility") and the semantic-based part segmentation task ("Semantic"). "PN++" refers to "PointNet++". Reported values are performance of the trained networks on out-of-domain test datsets.

	Mobility	Semantic
Contrastive	44.0	25.9
AutoGPart$_{\text{PN++}}$	66.5	33.9

4.3. Part-aware and Geometry-discriminative Supervision Feature Space

In our method, we design the supervision space to be aware of part-level information by sampling features from points having the same part labels with the target point. Moreover, the feature space is made aware of geometric features by using ground-truth geometric features to construct the input feature set. In this section, we conduct ablation study on the mobility-based part segmentation task to prove the necessity of making the calculated features aware of both part labels and geometric features.

Ablating part labels. We conduct an experiment to ablate part labels in the calculation of supervision features by using features of points sampled from the neighbourhood of each point without considering whether they belong to the same part. Resulting models are denoted as "−Part Labels" in Table 6.

Ablating geometric features. We conduct an experiment to ablate geometric features in the supervision feature calculation process by only using part labels from the sampled points as input features in the supervision feature space. Resulting models are denoted as "−Geometric Features" in Table 6.

Table 6. Ablation study w.r.t. reward function design and supervision space design. For abbreviations used, "Arch." refers to "Architecture"; "PN++" denotes "PointNet++"; In/Out-of-dist. refers to In/Out-of-distribution performance.

Ablation	Arch.	In-dist.	Out-of-dist.
/	DGCNN	83.1	**73.8**
−Part Labels		82.4	71.3
−Geometric Features		83.9	68.9

The results of such ablations are summarized in Table 6. Such results demonstrate the necessity of including geometric features and part labels in the intermediate supervision feature calculation process simultaneously. Furthermore, being aware of geometric features are more important than including part labels in the supervision feature calculation process if we compare the performance of the resulting models for their respective ablation versions. Moreover, if we compare the performance of the model "−Geometric Features" with the model optimized using no intermediate supervisions, it can be discovered that the ablated version, where only part labels are involved in the calculation process, would result very limited performance improvement. Possible reasons may include 1) Operators designed for geometric features are not that suitable to encode part labels; 2) Adding intermediate supervisions by only letting the model aware of part labels encoded in another form is not enough to help the model learn more part related cues. Letting the calculated supervision features aware of part-level geometric information can help the network learn useful cues defining parts for the part segmentation task, thus benefiting the networks' performance better than only using a part of them.

4.4. Supervision Feature Generation Strategy

In this work, we propose to generating supervision features from an operation tree operating on input part-aware geometric features. No doubt that there are many other methods that can be used to generate such features from ground-truth input features for supervision. We explore one possibility by letting a network consume such input part-aware geometric features for intermediate ground-truth supervision features. The network is optimized together with the main network by gradient descent. Resulting models are denoted as "NN" and the results are summarized in Table 7. As can be inferred from the table, using a network to generate ground-truth features for prediction is not a wise choice, with the performance improvement that can be added on the original simple segmentation networks very limited. It indicate that, using such simple gradient descent strategy, a

Table 7. Ablation study w.r.t. reward function design and supervision space design. For abbreviations used, "Arch." refers to "Architecture"; "PN++" denotes "PointNet++"; "No Loss" indicates simple segmentation networks trained without intermediate supervisions; In/Out-of-dist. refers to In/Out-of-distribution performance.

Ablation	Arch.	In-dist.	Out-of-dist.
/	PN++	87.2	**66.5**
/ (No Loss)		86.4	61.0
NN		86.5	62.5
/	DGCNN	83.1	**73.8**
/ (No Loss)		88.3	68.1
NN		89.5	67.0

network cannot "learn" the correct way to generate high-quality ground-truth features for intermediate supervisions. This can also prove an effective strategy to use a computing graph-like operation tree for ground-truth features.

4.5. Experimental Results Details

In this section, we present some details of experimental results that are not presented in the main paper.

Performance comparison between HPNet and AutoGPart$_{\text{HPNet}}$. Performance comparison between HPNet and AutoGPart$_{\text{HPNet}}$ on different data clusters is shown in Figure 1.

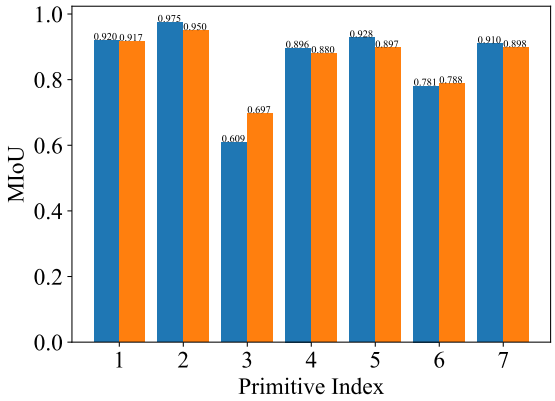

Figure 1. Performance comparison between HPNet and AutoGPart$_{\text{HPNet}}$ on different data clusters.

4.6. Segmentation Results Visualization

In this section, we visualize and present some segmentation results for the mobility-based part segmentation task and the semantic-based part segmentation task. Point clouds are normalized into a ball with radius 1.0. Thus the presented shapes may be different from those used in the training stage.

Mobility-based part segmentation. Figure 2 and 3 (presented in next few pages) show the selected segmentation visualization for the mobility-based part segmentation task, where "Baseline" denotes "DGCNN" model using no intermediate supervision while "Ours" denotes "AutoGPart$_{\text{DGCNN}}$" model using intermediate loss searched by AutoGPart .

Semantic-based part segmentation. Figure 4 and 5 (presented in next few pages) show the segmentation results of AutoGPart on shapes from several test categories for the semantic-based part segmentation task.

5. Further Discussion

In this work, we propose to automatically find useful intermediate supervisions to help with improve the generalization ability of 3D part segmentation networks. Although experiments prove the value of adding intermediate supervisions for improving networks' cross-domain performance, the network structure used in current work is still limited to a single stage end-to-end learning-based network which does not vary across different segmentation networks. However, it is valuable to explore how to include the architecture of segmentation networks into the design space and automatically select suitable network architectures for different part segmentation tasks. Making the network architecture flexible can help enlarge the design space, thus including more highly expressive networks into the search space.

Another line lies in more explorations on other tasks, not only limited to 3D part segmentation tasks. It is expected that adding intermediate supervisions a general approach to prevent network from learning shortcut features for tasks from a much broader range. However, it is still not proved and worth further exploration. Moreover, migrating the research goal from only improving the network's cross-domain generalization ability to enhancing both of its in-distribution and out-of-distribution performance is also a meaningful direction.

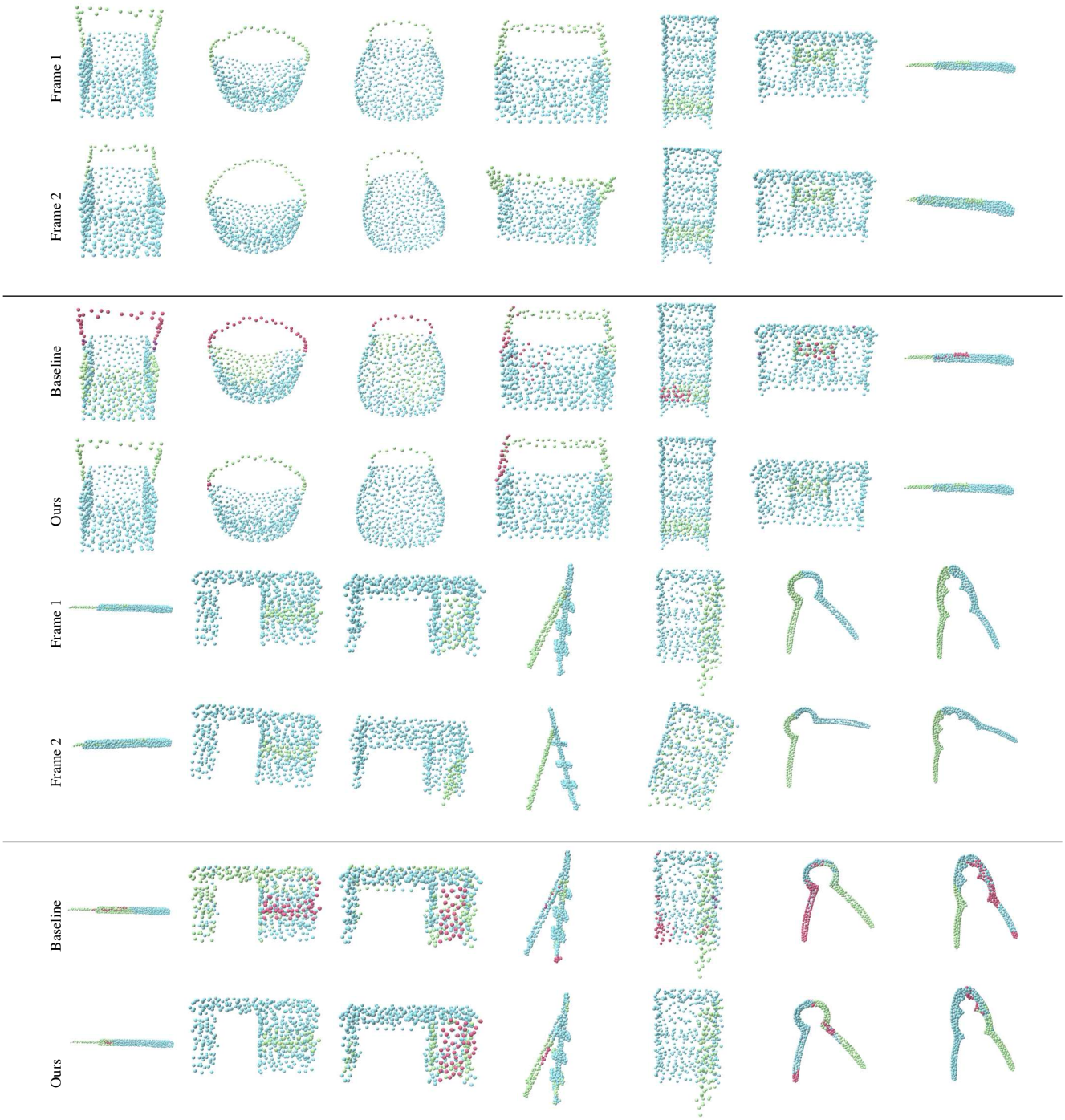

Figure 2. Mobility-based segmentation results visualization. "Baseline" refers to the segmentation network using DGCNN as its backbone without adding any intermedaite supervision. "Ours" denotes the model "AutoGPart$_{\text{DGCNN}}$".

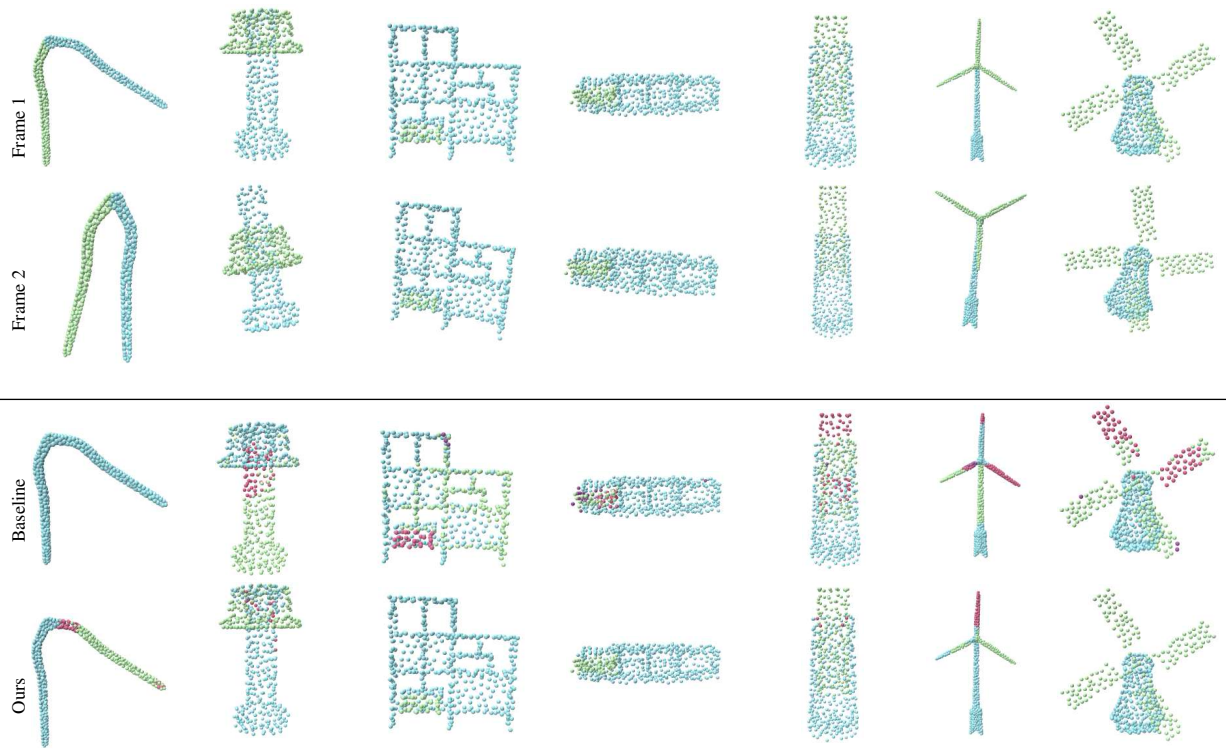

Figure 3. Mobility-based segmentation results visualization. "Baseline" refers to the segmentation network using DGCNN as its backbone without adding any intermedaite supervision. "Ours" denotes the model "AutoGPart$_{\text{DGCNN}}$".

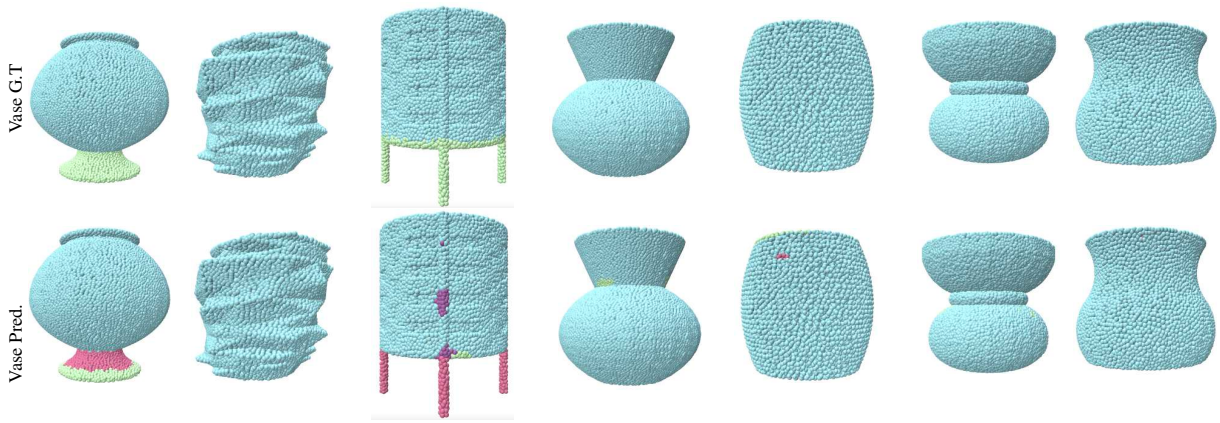

Figure 4. Semantic-based segmentation results visualization. "G.T" refers to the ground-truth segmentation results. "Pred." denotes the model "AutoGPart".

Figure 5. Semantic-based segmentation results visualization. "G.T" refers to the ground-truth segmentation results. "Pred." denotes the model "AutoGPart".

References

[1] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. Technical report, Stanford, 2006. 4

[2] Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A Wichmann. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2(11):665–673, 2020. 5

[3] Ruizhen Hu, Wenchao Li, Oliver Van Kaick, Ariel Shamir, Hao Zhang, and Hui Huang. Learning to predict part mobility from a single static snapshot. *ACM Transactions on Graphics (TOG)*, 36(6):1–13, 2017. 4

[4] Oliver Van Kaick, Noa Fish, Yanir Kleiman, Shmuel Asafi, and Daniel Cohen-Or. Shape segmentation by approximate convexity analysis. *ACM Transactions on Graphics (TOG)*, 34(1):1–11, 2014. 6

[5] Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy M Hospedales. Learning to generalize: Meta-learning for domain generalization. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. 6

[6] Lingxiao Li, Minhyuk Sung, Anastasia Dubrovina, Li Yi, and Leonidas J Guibas. Supervised fitting of geometric primitives to 3d point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2652–2660, 2019. 4, 5, 6

[7] Tiange Luo, Kaichun Mo, Zhiao Huang, Jiarui Xu, Siyu Hu, Liwei Wang, and Hao Su. Learning to group: A bottom-up framework for 3d part discovery in unseen categories. *arXiv preprint arXiv:2002.06478*, 2020. 4, 6

[8] Lucas Mansilla, Rodrigo Echeveste, Diego H. Milone, and Enzo Ferrante. Domain generalization via gradient surgery. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6630–6638, October 2021. 6

[9] Kaichun Mo, Shilin Zhu, Angel X Chang, Li Yi, Subarna Tripathi, Leonidas J Guibas, and Hao Su. Partnet: A large-scale benchmark for fine-grained and hierarchical part-level 3d object understanding. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 909–918, 2019. 4

[10] Charles R Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *arXiv preprint arXiv:1706.02413*, 2017. 7

[11] Dimitrios Tzionas and Juergen Gall. Reconstructing articulated rigged models from rgb-d videos. In *European Conference on Computer Vision*, pages 620–633. Springer, 2016. 6

[12] Weiyue Wang, Ronald Yu, Qiangui Huang, and Ulrich Neumann. Sgpn: Similarity group proposal network for 3d point cloud instance segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2569–2578, 2018. 6

[13] Siming Yan, Zhenpei Yang, Chongyang Ma, Haibin Huang, Etienne Vouga, and Qixing Huang. Hpnet: Deep primitive segmentation using hybrid representations. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 2753–2762, October 2021. 5, 6

[14] Li Yi, Haibin Huang, Difan Liu, Evangelos Kalogerakis, Hao Su, and Leonidas Guibas. Deep part induction from articulated object pairs. *arXiv preprint arXiv:1809.07417*, 2018. 4, 6

[15] Li Yi, Vladimir G Kim, Duygu Ceylan, I-Chao Shen, Mengyan Yan, Hao Su, Cewu Lu, Qixing Huang, Alla Sheffer, and Leonidas Guibas. A scalable active framework for

region annotation in 3d shape collections. *ACM Transactions on Graphics (ToG)*, 35(6):1–12, 2016. 4

[16] Li Yi, Wang Zhao, He Wang, Minhyuk Sung, and Leonidas J Guibas. Gspn: Generative shape proposal network for 3d instance segmentation in point cloud. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3947–3956, 2019. 6

[17] Qing Yuan, Guiqing Li, Kai Xu, Xudong Chen, and Hui Huang. Space-time co-segmentation of articulated point cloud sequences. In *Computer Graphics Forum*, volume 35, pages 419–429. Wiley Online Library, 2016. 6

[18] Kaiyang Zhou, Yongxin Yang, Yu Qiao, and Tao Xiang. Domain generalization with mixstyle. In *International Conference on Learning Representations*, 2021. 6

[19] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3d: A modern library for 3d data processing. *arXiv preprint arXiv:1801.09847*, 2018. 6