Generative Hard Example Augmentation for Semantic Point Cloud Segmentation

Supplementary Material

1. Implementation Details

Converting Voxels to Point Clouds As shown in the figures *Generative Point Cloud Reshape* and *Generative Example Mixup* in the manuscript, we convert voxels to point clouds for generating semantic point clouds. This operation contains the fully connected layer $\mathcal{F}$ (see the left of Figure 1) to regress the offset $(\Delta x, \Delta y, \Delta z)$ for moving each point of the source example $\mathbf{R}^{(3)}$ to form the new points in the generated example $\mathbf{T}_o^{(3)}$. As the right of Figure 1 illustrates, we use the trilinear interpolation to compute the new label for each new point in the generated example $\mathbf{T}_o^{(3)}$. The interpolated labels are taken from the voxelized grid $\mathcal{G}(\mathbf{T}_o^{(C)})$, which is output by the generative network.

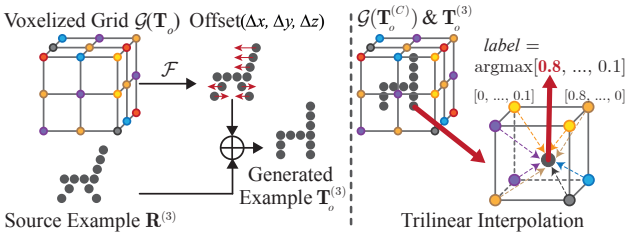


Figure 1. The illustration of operation for converting voxels to point clouds.

Trilinear Interpolation Given the source point cloud $\mathbf{R} \in \mathbb{R}^{N\times(3+C)}$ and the target point cloud $\mathbf{T} \in \mathbb{R}^{N\times(3+C)}$, we voxelize them to the voxelized grids $\mathcal{G}(\mathbf{R}), \mathcal{G}(\mathbf{T}) \in \mathbb{R}^{(M\times M\times M)\times(3+C)}$ by trilinear interpolation, respectively. For the details of trilinear interpolation, as illustrated in Figure 2(a) (mentioned in Eq. (1) in the manuscript), it set the spatial resolution of the grid as M. Specifically, these grids have $|\mathbf{N}|$ vertexes in the grids, where $\mathbf{N} \in \mathbb{R}^{(M\times M\times M)\times 3}$. And $\mathbf{N}_i \in \mathbb{R}^3$ depicts the i^{th} vertex in the grids. Also, each point with a weight w from the point to the vertex, where $w \in \mathbb{R}$. The weight will be larger if the point is close to the vertex. Thus, we aggregate the neighboring points to get the voxelized grid as follows:

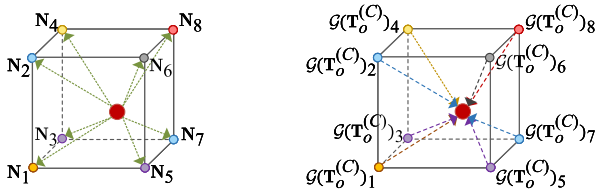
$$\begin{aligned} \mathcal{G}(\mathbf{I})_i &= \sum_{\mathbf{I}_j^{(3)}\in\mathcal{N}(\mathbf{N}_i)} \frac{w_j \cdot \mathbf{I}_j^{(3+C)}}{|\mathcal{N}(\mathbf{N}_i)|}, \\ w_j &= \prod_{\{\mathbf{N}_i,\mathbf{I}_j^{(3)}\}|_{\{x,y,z\}}} (1 - |\mathbf{N}_i - \mathbf{I}_j^{(3)}|), \end{aligned} \tag{1}$$

where $\mathbf{I}_j^{(3)} \in \mathbb{R}^3$ depicts the coordinates of a point in the point cloud examples $\mathbf{R}$ or $\mathbf{T}$, and $|\mathcal{N}(\mathbf{N}_i)|$ is the number of neighboring points in examples concerning $\mathbf{N}_i$.

As described in Eq. (5) in the manuscript, we conduct the trilinear interpolation to propagate the category scores of the voxelized grid to each point of generated examples, which is illustrated in Figure 2(b). We first compute the weight coefficients $w_i^{'}, i = 1, 2, \cdots, 8$, which are from the points to eight vertexes concerning the generated example $\mathbf{T}_o$ by Eq. 1. Then we calculate the category scores from the vertex of the grid to each point according to $w^{'}$ as follows:

$$\mathbf{T}_o^{(C)} = \text{argmax}\Big(\sum_{\mathcal{G}(\mathbf{T}_o)_i\in\mathcal{N}(\mathbf{T}_{o_j}^{(3)})} w_i^{'}\mathcal{G}(\mathbf{T}_o^{(C)})_i\Big), \tag{2}$$

where $\text{argmax}(\cdot)$ is a function to select the index with the max value in the C-dimension vector.



(a) Voxelization (b) Propagating Scores

Figure 2. The illustration of operation for Trilinear Interpolation.

Architecture of Encoder and Decoder In the manuscript, we employ an encoder to embed the discrepancy grid into latent space and adopt a decoder to get the discrepancy grid from the discrepancy representation. For the 3D grid, we follow the idea of U-Net [1] connections to implement a 3D encoder-decoder. The architecture of the encoder consists of four 3D convolution layers. Each of them has normalization, LeakyReLU activation, and max pooling operation. All convolution layers have 4^3 filters, and the padding size is 2. Finally, the encoder ends with two fully connected layers with 2048 and 1024 dimensions, respectively. As for the architecture of the decoder, it is composed of five deconvolutional layers, and the setting of filters, padding size, normalization, LeakyReLU activation, and max pooling operation are the same as the encoder.

Experimental Setups We implement our framework by PyTorch [2]. We train and evaluate the segmentation and generative networks on NVIDIA GeForce RTX 3090 GPUs with 128G memory. We set a batch size of 32 for 200 epochs in ShapeNetPart data, and a batch size of 16 for 100 epochs

in S3DIS and ScanNet V2 scenes. We adopt AdamW optimizer for the segmentation backbones (i.e., PointNet++ [3], PointNeXt (C=160) [4], PointMetaBase (C=160) [5], and DeLA [6]) with an initial learning rate of 0.001. These backbones can work with or without data augmentation modules like the conventional data augmentation (CDA), PointWOLF (PW) [7], and our GHEA. For a fair comparison, we leave the hyper-parameters of the segmentation backbones to be the same as those in the official code packages.

2. Supplementary Experiments

2.1. Sensitivity Analysis of Hyper-Parameters

Number of Generated Examples The number of generated examples determines the complexity of *Weighting of Hard Example*, which is described as Eq. (6) and Eq. (7) in the manuscript. Here, we analyze the sensitivity of it by selecting the number from $\{1, 3, 5, 7, 9, 11\}$. We evaluate the segmentation results (Cat. mIoU and Ins. IoU), GPU memory (GB), and testing time (ms) to demonstrate the changing trends of the PointMetaBase [5] backbone. As illustrated in Figure 3(a) and (b), the segmentation performances gradually improve as the number of generated examples increases. The segmentation network performs best while reaching the threshold value (e.g., 5). In the latter half of the trend, the performance decreases when more significant than the threshold. It demonstrates that an appropriate value of generated examples promotes GHEA to create examples that are challenging enough. However, a more substantial value may lead to weak challenges or general examples that suppress the learning of the segmentation network. This is because the number of examples with more significant errors will increase as the number of generated examples increases, which will cause the coefficients in the weighting operation to be very close to each other. Therefore, it is difficult to make the shape of the hard example more like the specifically generated example, which is much more challenging in the current stage. The segmentation network need stronger, complex data that achieves general results.

In Figure 3(c) and (d), both GPU memory and testing time gradually increased. Because the number of generated examples is more prominent, reflecting GHEA's computation complexity, and the higher cost needs to be paid.

Number of Target Examples In Figure 4(a) and (b), we depict the sensitivity of target examples for constructing point cloud pairs with PointMetaBase [5] backbone, which reflects the impact of the number of discrepancy grids for each source example on segmentation performances. We find that the metric scores (Cat. mIoU and Ins. mIoU) are increasingly higher as the number of target examples continuously increases. This phenomenon demonstrates that more discrepancy representations can facilitate the generative network to enrich latent space sufficiently, which helps produce complex examples. However, the more discrepancy grids for each source example, the more training data increases significantly. Thus, we set the number of target examples as 20 for the lowest number of six classes in ShapeNetPart, and the other classes are set as 5.

Table 1. The results of different corruptions for PointNet++ backbone with Ins. mIoU.

Method	Ori.	0.9	1.1	Jitt.	90°	180°
PointNet++	85.1	85.0	85.0	84.1	59.7	38.8
w/ PW	85.4	85.3	85.3	84.8	63.6	40.3
w/ GHEA	**86.4**	**86.2**	**86.2**	**85.9**	**65.6**	**43.0**

2.2. Additional Discussions for GHEA

Diversity of Hard Examples We measure the diversity of the difference between the source and generated examples from coordinates and labels in the manuscript. In this section, we evaluate the diversity between each source and generated examples, which is also measured by the Chamfer Distance of 3D points (CD in Figure 5(a)) and the L2 distance of voxelized grids (L2 in Figure 5(b)). Specifically, we select the smallest value among all pairs of each source and hard examples. Like the manuscript, GHEA leads to a significant diversity of all examples in the early epochs because the insufficient training of the generative network will produce unreasonable shapes and labels. However, the diversity of the generated examples increases when the generative network is trained better. Besides, it decreases in the final epochs. This is because those target examples' patterns naturally limit the generated examples' diversity patterns.

Robustness Analysis for GHEA We corrupt the samples of the ShapeNetPart test set to evaluate the robustness of GHEA. Specifically, we design the following six manners: exerting scaling with a ratio of 0.9 or 1.1, setting rotation with 90° or 180°, and bringing jittering with Gaussian noise ranged [-1.0, 1.0] (Jitt.). For each group, we use PointNet++ [3] as the backbone and set three different strategies: the original PointNet++ backbone, the backbone with PointWOLF [7], and the backbone with GHEA.

In Table 1, the first column illustrates the original test results (Ori.) without any corruptions, for example. Compared with the first two rows, GHEA achieves better performance consistently. This phenomenon depicts our framework can continuously generate hard examples according to the current segmentation results. Particularly, comparing the Ori. and the last row, the sensitivity of GHEA is less than other baselines, which perform higher results in rotation groups. Such a result convincingly verifies that GHEA improves the robustness of augmentation significantly.

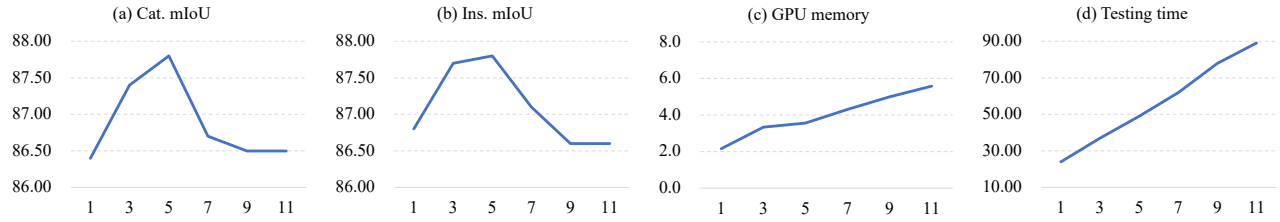


Figure 3. The sensitivity analysis of the generated examples (Cat. mIoU, Ins. mIoU, GPU memory (GB), and testing time (ms)).

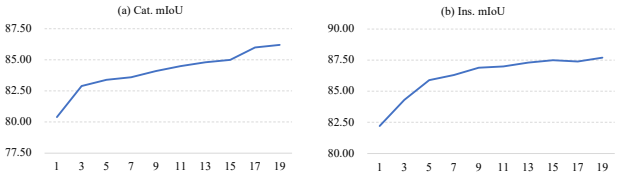


Figure 4. The sensitivity of the target examples (Cat. mIoU and Ins. mIoU).

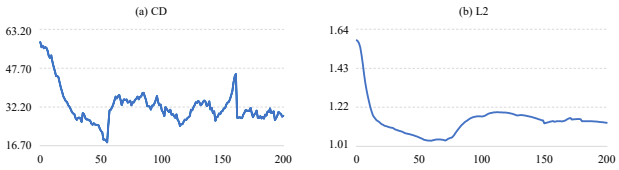


Figure 5. The diversity between each source and new examples generated in different epochs of the network training.

2.3. Ablation Study on S3DIS

Visualization of Reshaped Examples In Figure 6, we exhibit the generated examples in the training stage. Comparing the first column and the others, the latter scenes show their remarkable differences from the source examples. It depicts that GHEA can produce diverse examples to enrich the data for training the segmentation network.

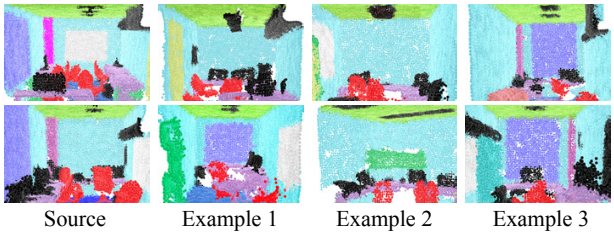


Figure 6. Visualization of reshaped examples in different epochs during training.

Effectiveness of GHEA for the Segmentation Network In Figure 7, we verify the effectiveness of the segmentation network, whether updated by GHEA or not. We depict

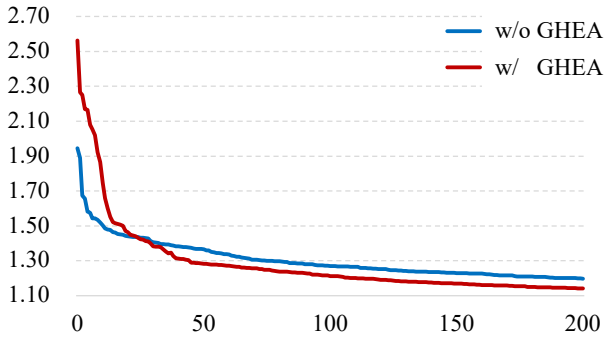


Figure 7. Cross-entropy losses of the segmentation network with-/without GHEA.

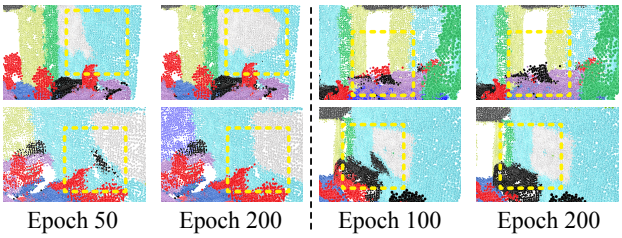


Figure 8. Visualization of hard examples and segmentation results in different epochs.

cross-entropy losses of the segmentation network in different epochs. The phenomenon on S3DIS is similar to that in ShapeNetPart. The segmentation network without GHEA converges faster than that with GHEA. However, the network with GHEA gets smaller loss values after the intersection. The generative network can gradually produce more complex examples for training the segmentation network in the latter epochs. In Figure 8, we exhibit some hard examples generated in different epochs. We also visualize their segmentation results, which are predicted in the final epoch. Following the visualizations, the segmentation network achieves better results by learning from hard examples sufficiently.

3. More Visualizations of Segmentation

We provide more segmentation results on ShapeNetPart [8], S3DIS [9], and ScanNet V2 [10] in Figure 10, 11, and 12,

respectively. These segmentation results show that the segmentation networks are trained without or with the assistance of GHEA. We find that the backbones with GHEA generally yield more accurate segmentation results.

4. Limitation Analysis

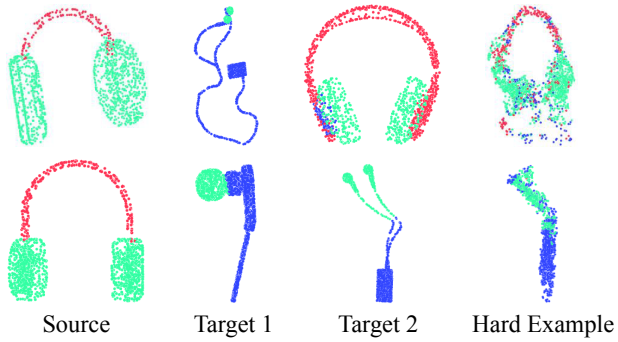


Figure 9. Mistake results of generative example mixup for GHEA.

We further discuss the limitation of GHEA, which we will focus on in future work. Although GHEA can significantly augment segmentation performances by generating diverse complex examples compared with random manipulation and discriminative networks, it sometimes produces unreasonable examples. As illustrated in Figure 9, we weigh the mixup representation for the complex example by three discrepancy representations for a source earphone example. However, the produced complex example contains problematic labels. This is because the generated examples are distinctively different in geometry. The generative example mixup strategy is not enough to solve this challenging problem. Therefore, the next step is to improve the generative example mixup method for aggregating discrepancy representations.

References

[1] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *In International Conference on Medical Image Computing and Computer-Assisted Intervention*, 2015. 1

[2] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. *In Conference on Neural Information Processing Systems*, 2019. 1

[3] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *In Conference on Neural Information Processing Systems*, 2017. 2

[4] Guocheng Qian, Yuchen Li, Houwen Peng, Jinjie Mai, Hasan Abed Al Kader Hammoud, Mohamed Elhoseiny, and Bernard Ghanem. Pointnext: Revisiting pointnet++ with improved training and scaling strategies. *In Conference on Neural Information Processing Systems*, 2022. 2

[5] Haojia Lin, Xiawu Zheng, Lijiang Li, Fei Chao, Shanshan Wang, Yan Wang, Yonghong Tian, and Rongrong Ji. Meta architecure for point cloud analysis. *In Conference on Computer Vision and Pattern Recognition*, 2023. 2

[6] Binjie Chen, Yunzhou Xia, Yu Zang, Cheng Wang, and Jonathan Li. Decoupled local aggregation for point cloud learning. *arXiv preprint arXiv: 2308.16532*, 2023. 2

[7] Sihyeon Kim, Sanghyeok Lee, Dasol Hwang, Jaewon Lee, Seong Jae Hwang, and Hyunwoo J. Kim. Point cloud augmentation with weighted local transformations. *In IEEE International Conference on Computer Vision*, 2021. 2

[8] Li Yi, Vladimir G. Kim, Duygu Ceylan, I-Chao Shen, Mengyan Yan, Hao Su, Cewu Lu, Qixing Huang, Alla Sheffer, and Leonidas J. Guibas. A scalable active framework for region annotation in 3d shape collections. *ACM Transactions on Graphics*, 2016. 3

[9] Iro Armeni, Sasha Sax, Amir R. Zamir, and Silvio Savarese. Joint 2d-3d-semantic data for indoor scene understanding. *arXiv preprint arXiv: 1702.01105*, 2017. 3

[10] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas A. Funkhouser, and Matthias Nießner. Richly-annotated 3d reconstructions of indoor scenes. *In Conference on Computer Vision and Pattern Recognition*, 2017. 3

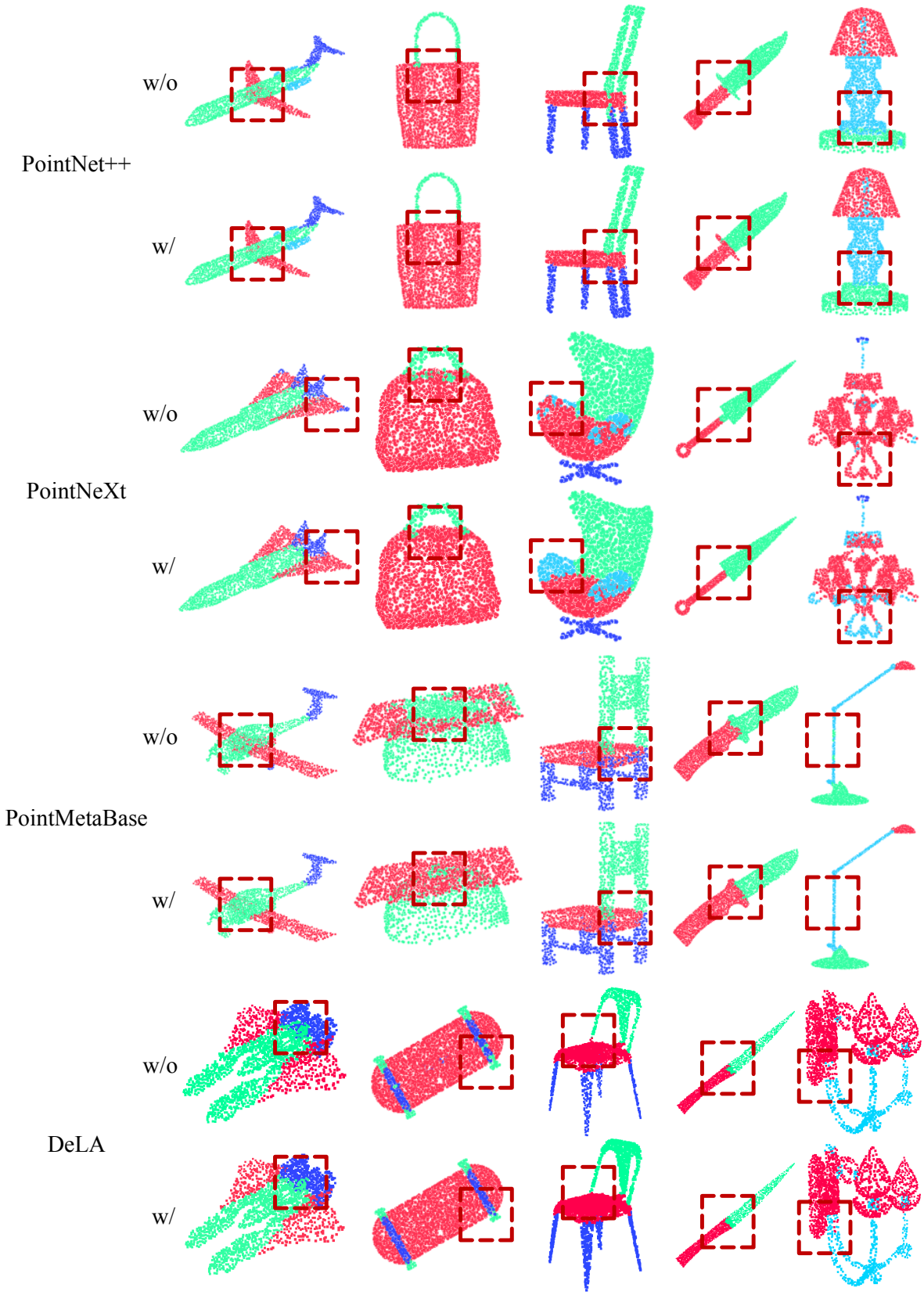


Figure 10. Part-level segmentation results on the ShapeNetPart. The red dash squares highlight the wrong/correct segmentation without/with the help of GHEA.

Figure 11. Object-level segmentation results on the S3DIS dataset. The yellow dash squares highlight the wrong/correct segmentation without/with the help of GHEA.

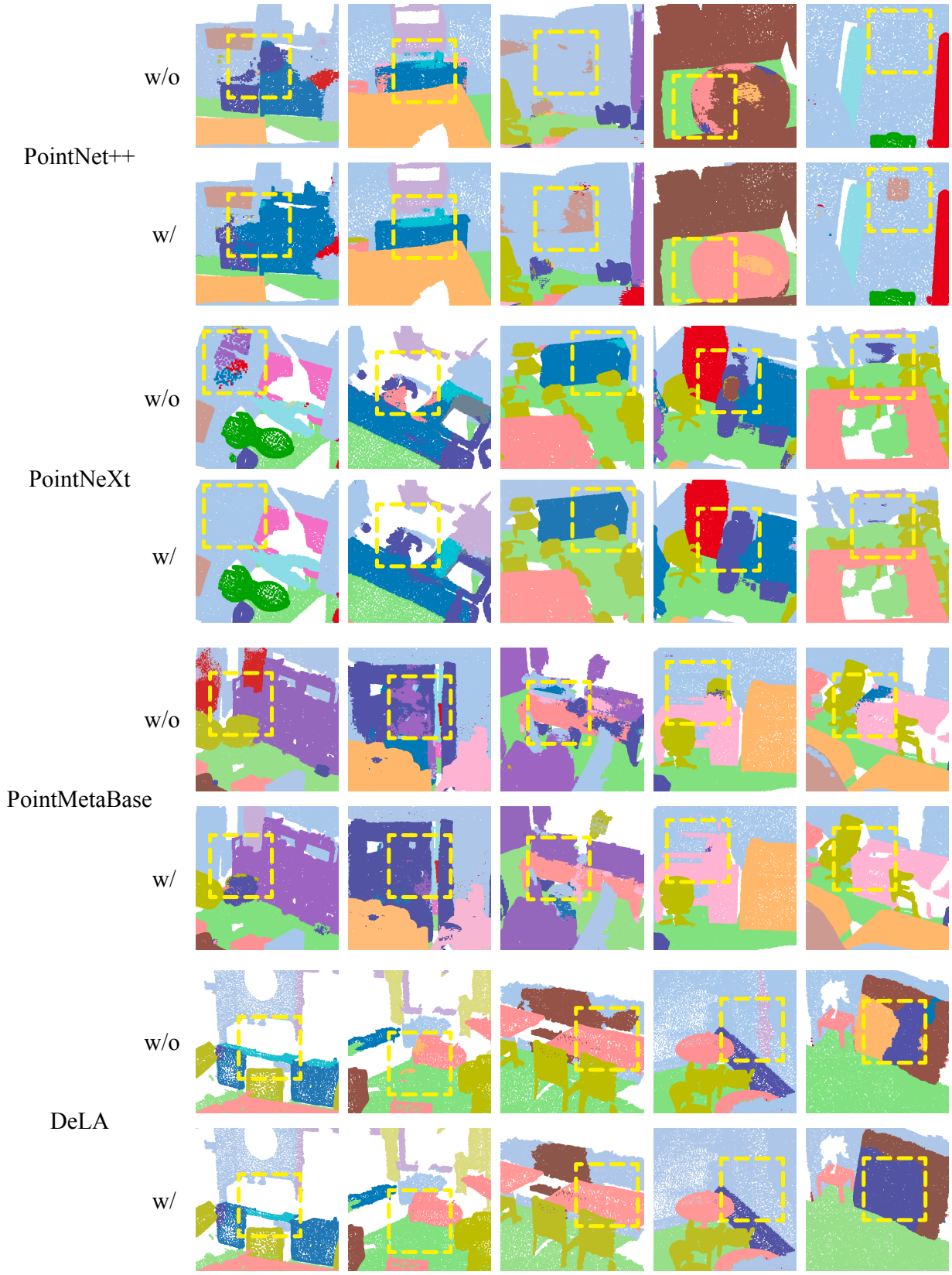


Figure 12. Object-level segmentation results on the ScanNet V2 dataset. The yellow dash squares highlight the wrong/correct segmentation without/with the help of GHEA.