Supplemental Material for Self-Supervised Learning for Domain Adaptation on Point Clouds

Idan Achituve
Bar-Ilan University
Ramat Gan, Israel
idan.achituve@biu.ac.il

Haggai Maron
NVIDIA
Tel-Aviv, Israel
hmaron@nvidia.com

Gal Chechik
Bar-Ilan University, Israel
NVIDIA, Israel

A. PointSegDA dataset

We built PointSegDA dataset based on a dataset of triangular meshes of human models proposed by [7]. This dataset is made of the following four datasets, which serve as different domains: ADOBE, FAUST, MIT, and SCAPE. The datasets differ in body pose, shape, and point distribution. We generated a point cloud from each mesh in each domain by extracting all the vertices (edges were neglected) and sampling 2048 points according to farthest point sampling. We aligned the shapes with the positive Z-axis and scaled them to the unit cube as in [8]. Point labels were obtained from the polygon labels. In case of a conflict (i.e., the same point appears in two polygons having different labels) we randomly picked one label. As a result, a total of eight point-labels were obtained: foot, leg, thigh, torso, upper arm, forearm, hand, and head. Table 5 shows the train/val/test splits, and Fig. 6 depict three shapes from each domain.

Dataset	Train	Validation	Test	Total
FAUST	70	10	20	100
MIT	118	17	34	169
ADOBE	29	4	8	42
SCAPE	50	7	14	71

Table 5: Number of samples in each sets.

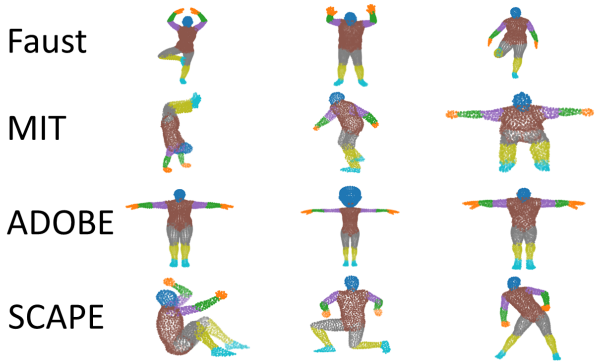


Figure 6: A comparison of typical shapes from the datasets: FAUST, MIT, ADOBE, and SCAPE.

B. Implementation details

B.1. PointDA-10 dataset

Data processing. Following several studies [6, 8, 14] we assume that the upwards direction of all point clouds in all datasets is known and aligned. Since point clouds in ModelNet are aligned with the positive Z axis, we aligned samples from ShapeNet and ScanNet in the same direction by rotating them about the x-axis. We sampled 1024 points from shapes in ModelNet and ScanNet (which have 2048 points) using farthest point sampling as in [8]. We split the training set to 80% for training and 20% for validation and scaled the shapes to the unit-cube. During training, we applied jittering as in [8] with standard deviation and clip parameters of 0.01 and 0.02 respectively, and random rotations to shapes about the Z axis only.

Network architecture. In all methods we used DGCNN [14] for the feature extractor with the following configurations: Four point cloud convolution layers of sizes [64, 64, 128, 256] respectively and a 1D convolution layer with kernel size 1 (feature-wise fully connected) with a size of 1024 before extracting a global feature vector by max-pooling. We implemented a spatial transformation network to align the input point set to a canonical space using two point-cloud convolution layers with sizes [64, 128] respectively, a 1D convolution layer of size 1024 and three fully connected layers of sizes [512, 256, 3] respectively. The classification head, h_{sup}, was implemented using three fully connected layers with sizes [512, 256, 10] respectively (where 10 is the number of classes). The same architecture was applied to both classification heads of PointDAN [9]. The SSL head, h_{SSL}, of DefRec, DAE-point and RS [11] was implemented similarly using four 1D convolution layers of sizes

Method	ModelNet to ShapeNet	ModelNet to ScanNet	ShapeNet to ModelNet	ShapeNet to ScanNet	ScanNet to ModelNet	ScanNet to ShapeNet	Avg.
Unsupervised	83.3 ± 0.7	43.8 ± 2.3	75.5 ± 1.8	42.5 ± 1.4	63.8 ± 3.9	64.2 ± 0.8	62.2 ± 1.8
DANN [1]	75.3 ± 0.6	41.5 ± 0.2	62.5 ± 1.4	46.1 ± 2.8	53.3 ± 1.2	63.2 ± 1.2	57.0 ± 1.2
DANN + PCM	74.8 ± 2.8	42.1 ± 0.6	57.5 ± 0.4	50.9 ± 1.0	43.7 ± 2.9	71.6 ± 1.0	56.8 ± 1.5
PointDAN [9]	82.5 ± 0.8	47.7 ± 1.0	77.0 ± 0.3	48.5 ± 2.1	55.6 ± 0.6	67.2 ± 2.7	63.1 ± 1.2
PointDAN + PCM	83.9 ± 0.3	44.8 ± 1.4	63.3 ± 1.1	45.7 ± 0.7	43.6 ± 2.0	56.4 ± 1.5	56.3 ± 1.2
RS [11]	81.5 ± 1.2	35.2 ± 5.9	71.9 ± 1.4	39.8 ± 0.7	61.0 ± 3.3	63.6 ± 3.4	58.8 ± 2.7
RS + PCM	79.9 ± 0.8	46.7 ± 4.8	75.2 ± 2.0	51.4 ± 3.9	71.8 ± 2.3	71.2 ± 2.8	66.0 ± 1.6
DAE-Global [2]	83.5 ± 0.8	42.6 ± 1.4	74.8 ± 0.8	45.5 ± 1.6	64.9 ± 4.4	67.3 ± 0.6	63.1 ± 1.6
DAE-Global + PCM	83.1 ± 0.5	47.2 ± 0.8	70.0 ± 1.0	52.8 ± 0.6	67.7 ± 2.1	**73.7 ± 0.6**	65.7 ± 0.9
DAE-Point	82.5 ± 0.4	40.2 ± 1.6	76.4 ± 0.7	50.2 ± 0.5	66.3 ± 1.5	66.1 ± 0.5	63.6 ± 0.9
DAE-Point + PCM	**85.0 ± 0.5**	50.2 ± 1.3	74.3 ± 0.7	50.9 ± 0.8	65.1 ± 1.7	72.2 ± 0.9	66.3 ± 1.0
DefRec (ours)	83.3 ± 0.2	46.6 ± 2.0	**79.8 ± 0.5**	49.9 ± 1.8	70.7 ± 1.4	64.4 ± 1.2	65.8 ± 1.2
DefRec + PCM (Ours)	81.7 ± 0.6	**51.8 ± 0.3**	78.6 ± 0.7	**54.5 ± 0.3**	**73.7 ± 1.6**	71.1 ± 1.4	**68.6 ± 0.8**

Table 6: Baselines methods with PCM. Test accuracy on PointDA-10 dataset, averaged over three runs (± SEM).

[256, 256, 128, 3]. The domain classifier head of DANN [1] was set similar to h_{sup}, namely three fully connected layers with sizes [512, 256, 2]. The reconstruction head of DAE-Global [2] was implemented using four 1D convolution layers of sizes [1024, 1024, 2048, 3072] respectively (where 3072: 1024×3 is the reconstruction size). In all heads the nonlinearity was ReLU and a dropout of 0.5 was applied to the first two hidden layers. Batch normalization [4] was applied after all convolution layers in Φ, h_{sup} and the auxiliary losses.

Training procedure. In all methods (baselines and ours), during training, we alternate between a batch of source samples and a batch of target samples. We used a fixed batch size of 32 per domain, ADAM optimizer [5], and a cosine annealing learning rate scheduler as implemented by PyTorch. We balanced the domains by undersampling the larger domain, source, or target, in each epoch. We applied grid search over the learning rates $\{0.0001, 0.001\}$ and weight decay $\{0.00005, 0.0005\}$. In all methods besides PointDAN, we applied a grid search over the auxiliary task weight $\lambda \in \{0.25, 1\}$. PointDAN has three loss-terms: classification, discrepancy, and MMD. Therefore, for this baseline we applied a grid search over $\{(0.33, 0.33, 0.33), (0.5, 0.25, 0.25)\}$ correspondingly. For DAE-point and DAE-Global we used a Gaussian noise sampled from $\mathcal{N}(0, 0.01)$ as suggested by [2]. We ran each configuration with 3 different random seeds for 150 epochs and used source-validation-based early stopping. The total training time of DefRec varies between 6-9 hours on a 16g Nvidia V100 GPU, depending on the datasets.

In the paper, we propose three types of deformations to the input point cloud. We implemented these methods with the following settings:

- *Volume-based deformations*. Deformations based on proximity in the input space. We examined two variants of deformations from this type: (a) Split the input space to $k \times k \times k$ equally sized voxels and pick a voxel at random. We tested this method for $k \in \{2, 3\}$ (b) The deformed region is a sphere with a fixed radius $r \in \{0.1, 0.2, ..., 1.0, 2.0\}$ that is centered around one data point selected at random.

- *Feature-based deformations*. Deformations based on proximity in the feature space. We examined deformations based on features extracted from layers $1 - 4$ of the shared feature encoder. The deformed region was set by randomly selecting a point and deforming its $\{100, 150, 200, 300, 500\}$ nearest neighbors in the feature space.

- *Sample-based deformations*. Deformations based on the sampling direction. For the gradient and the Lambertian methods, we followed the protocol suggested by [3]. For the split method, we randomly selected a cut off according to a beta distribution with parameters $a = 2.0, b = 5.0$.

B.2. PointSegDA dataset

Similar to the classification case, during training we applied jittering with standard deviation and clip parameters of 0.01 and 0.02 respectively, and random rotations to shapes about the Z axis only. The training procedure and network architecture were similar to the ones described in Section B.1 with the following exceptions:

- *Network*. We used the DGCNN feature extractor and segmentation head for segmentation tasks. Unlike the classification head, the segmentation head takes the global feature vector concatenated to feature representations of the points. It was implemented using four 1D

Method	ModelNet to ShapeNet	ModelNet to ScanNet	ShapeNet to ModelNet	ShapeNet to ScanNet	ScanNet to ModelNet	ScanNet to ShapeNet	Avg.
DefRec	84.7 ± 0.5	44.3 ± 2.3	79.3 ± 0.9	49.7 ± 1.0	66.3 ± 1.6	68.1 ± 0.9	65.4 ± 1.2
DefRec + PCM	84.0 ± 0.3	55.0 ± 1.2	74.7 ± 1.0	54.4 ± 0.1	69.7 ± 0.6	76.2 ± 0.3	69.0 ± 0.6

Table 7: Combining deformation strategies. Test accuracy on PointDA-10 dataset, averaged over three runs (± SEM).

convolution layers of sizes [256, 256, 128, 8], where 8 is the number of classes.

- *Training procedure*. The batch size was set to 16 per domain, the number of epochs was set to 200, and a grid search over the auxiliary task weight λ was done in $\{0.05, 0.1, 0.2\}$. Multi-level Adapt-SegMap [13] was implemented with two segmentation heads and two discriminators, all having the architecture described in the previous item. For Adapt-SegMap we applied grid search over the adversarial tasks weights in $\{(0.0002, 0.001), (0.0002, 0.01), (0.002, 0.001), (0.002, 0.01)\}$

- *DefRec hyperparameters*. (i) Volume-based deformations: we searched over the hyperparameters $k = 3$ and $r \in \{0.2, 0.3, 0.4, 0.5\}$. (ii) Feature-based deformations: we considered only the layers $2 - 3$. The deformed region was set by randomly selecting a point and deforming its $\{400, 600\}$ nearest neighbors in the feature space.

C. Additional experiments

C.1. PCM on baselines

Table 6 compares DefRec + PCM to the baseline methods combined with the PCM module. From the table, we notice that PCM boosts the performance of RS, DAE-Global, and DAE-Point but less so for DANN and PointDAN. Nevertheless, our proposed approach of combining PCM with DefRec is still superior. PointDAN uses a discrepancy loss which entails having two classification heads. Therefore, we speculate that adding PCM in this scenario hurts the performance. Combining PCM with several classification heads is an interesting research direction which we leave for future work.

C.2. Combining deformation strategies

DefRec procedure requires first to choose the deformation type and then hyperparameters specific for the type. This process may be cumbersome. Therefore, here we suggest an alternative protocol. Instead of choosing a specific deformation type, we propose to apply all of them with equal weight. For efficiency, this is achieved by choosing each deformation with a probability of $1/3$ in each batch. The hyperparameters of each type of deformation were set according to a sensitivity analysis based on the source-validation accuracy. As expected, we found the chosen hyperparameters to be highly correlated with the ones presented in Fig. 5 (e.g., radius of 0.2 for the volume-based). Table 7 shows the results of applying this protocol. Comparing these results with the ones in Table 1 shows that these two alternatives are comparable. On some adaptations there is a significant improvement, for example, in *ModelNet to ShapeNet* and *ModelNet to ScanNet*, when applying PCM, the accuracy increase by 2% and 3% respectively.

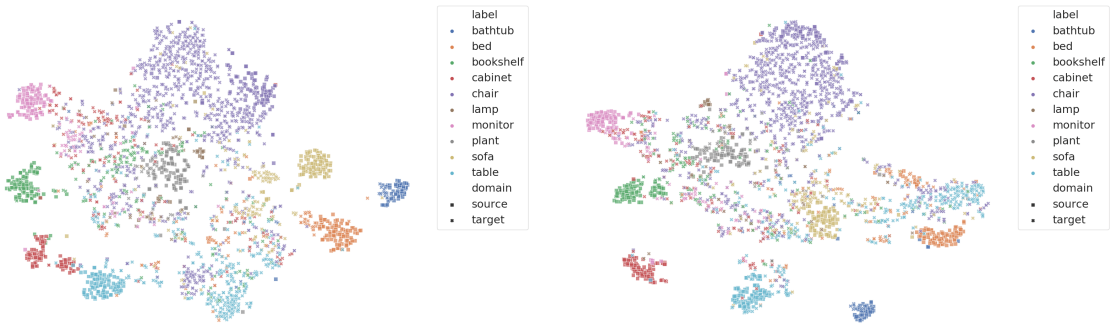
Method	Standrad Perplexity	Class-Balanced Perplexity
ModelNet to ScanNet		
PointDAN	**25.3 ± 1.4**	36.4 ± 1.1
DefRec + PCM	29.8 ± 1.9	**33.1 ± 1.4**
ModelNet to ShapeNet		
PointDAN	6.8 ± 0.4	23.6 ± 3.5
DefRec + PCM	**6.1 ± 0.3**	**20.4 ± 3.0**

Table 8: Log perplexity (± SEM). Lower is better.

D. Estimating target perplexity

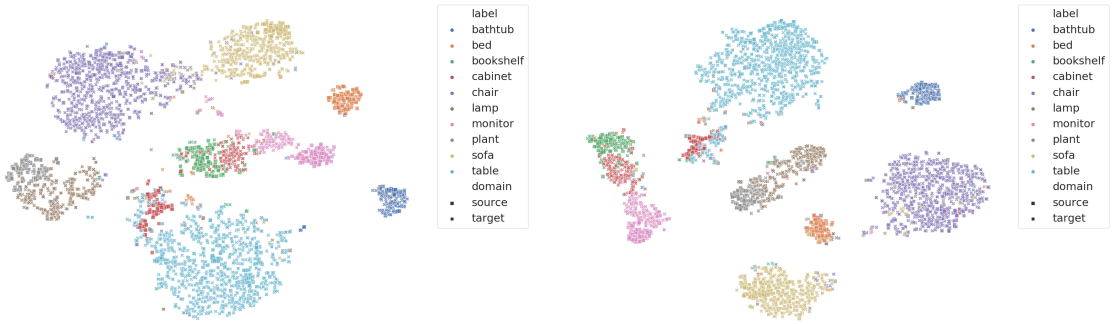
A key property of a DA solution is the ability to find an alignment between source and target distributions that is also discriminative [10]. To test that we suggest measuring the log perplexity of target test data representation under a model fitted by source test data representation. Here we consider the representation of samples as the activations of the last hidden layer in the classification network. The log perplexity measures the average number of bits required to encode a test sample. A lower value indicates a better model with less uncertainty in it.

Let $(x_1^t, y_1^t), ..., (x_1^t, y_n^t) \in T$ be a set of target instances. We note by n_c the number of target instances from class c. Using the chain rule, the likelihood of the joint distribution $p(x_j^t, y_j^t = c)$ can be estimated by finding $P(x_j^t | y_j^t = c)$ and $P(y_j^t = c)$. To model $P(x_j^t | y_j^t = c)$ we fit a Gaussian distribution $N(\mu_c, \Sigma_c)$ based on source samples from class c using maximum likelihood. To model $p(y_j^t = c)$ we take the proportion of source samples in class c.



(a) DefRec + PCM (b) PointDAN

Figure 7: The distribution of samples for the adaptation ModelNet to ScanNet.



(a) DefRec + PCM (b) PointDAN

Figure 8: The distribution of samples for the adaptation ModelNet to ShapeNet.

Modeling the class conditional distribution with a Gaussian distribution relates to the notion proposed in [12]. [12] suggested to represent each class with a prototype (the mean embeddings of samples belonging to the class) and assign a new instance to the class associated with the closest prototype. The distance metric used is the squared Euclidean distance. This method is equivalent to fitting a Gaussian distribution for each class with a unit covariance matrix.

The log perplexity of the target is (noted as standard perplexity here after):

$$L(T) = \sum_{c=1}^{10} \sum_{j=1}^{n_c} \frac{1}{n} \log \left(p(x_j^t | y_j^t = c) p(y_j^t = c) \right) \quad (4)$$

Alternatively we can measure the mean of a class-balanced log perplexity (noted as class-balanced perplexity here after):

$$L(T) = \frac{1}{10} \sum_{c=1}^{10} \sum_{j=1}^{n_c} \frac{1}{n_c} \log \left(p(x_j^t | y_j^t = c) p(y_j^t = c) \right) \quad (5)$$

Table 8 shows the standard perplexity and class-balanced perplexity of DefRec + PCM of our best model that was chosen based on the source-validation set and PointDAN [9] for the adaptations *ModelNet to ScanNet* and *ModelNet to ShapeNet*. Estimating the perplexity on the original space requires estimating a covariance matrix from a relatively small number of samples which results in a degenerate matrix. Therefore, we estimated the perplexity after applying dimensionality reduction to a 2D space using t-SNE. We ran t-SNE with the same configurations with ten different seeds and reported the mean and standard error of the mean. In Figures 7 and 8 we plot the t-SNE representations of one of the seeds.

From the table and the figures, we see that our method creates target and source representations that are more similar. In both adaptations, the class-balanced perplexity of our model is smaller. This is an indication that our model is doing a better job at learning under-represented classes. We note that PointDAN creates a denser representation of some classes (especially well-represented classes such as *Chair* and *Table*) however, they are not mixed better be-

tween source and target.

E. Shape reconstruction

Although we developed DefRec for the purpose of DA we expect it to learn reasonable reconstructions from point cloud deformations. Figures 9-11 show DefRec reconstruction of deformed shapes by the first variant of the volume-based type. Namely, we split the input space to $3 \times 3 \times 3$ voxels and pick one voxel uniformly at random.

Figure 9 demonstrate DefRec reconstruction of a shapes from all classes in the data for the simulated domains (left column) and the real domain (right column). Images of the same object are presented in the following order from left to right: the deformed shape (the input to the network), the original shape (the ground truth) and the reconstructed shape by the network. From the figure, it seems that the network manages to learn two important things: (1) It learns to recognize the deformed region and (2) it learns to reconstruct the region in a way that preserves the original shape. Note how in some cases, such as *Monitor* on the left column and *Lamp* on the right column, the reconstruction is not entirely consistent with the ground truth. The network reconstructs the object in a different (but still plausible) manner.

Figures 10 and 11 show DefRec reconstruction of *Chair* and *Table* objects respectively from deformations of different voxels in the objects. It can be seen that the network learns to reconstruct some regions nicely (such as the chair's top rail or table legs) while it fails to reconstruct well other regions (such as the chair's seat).

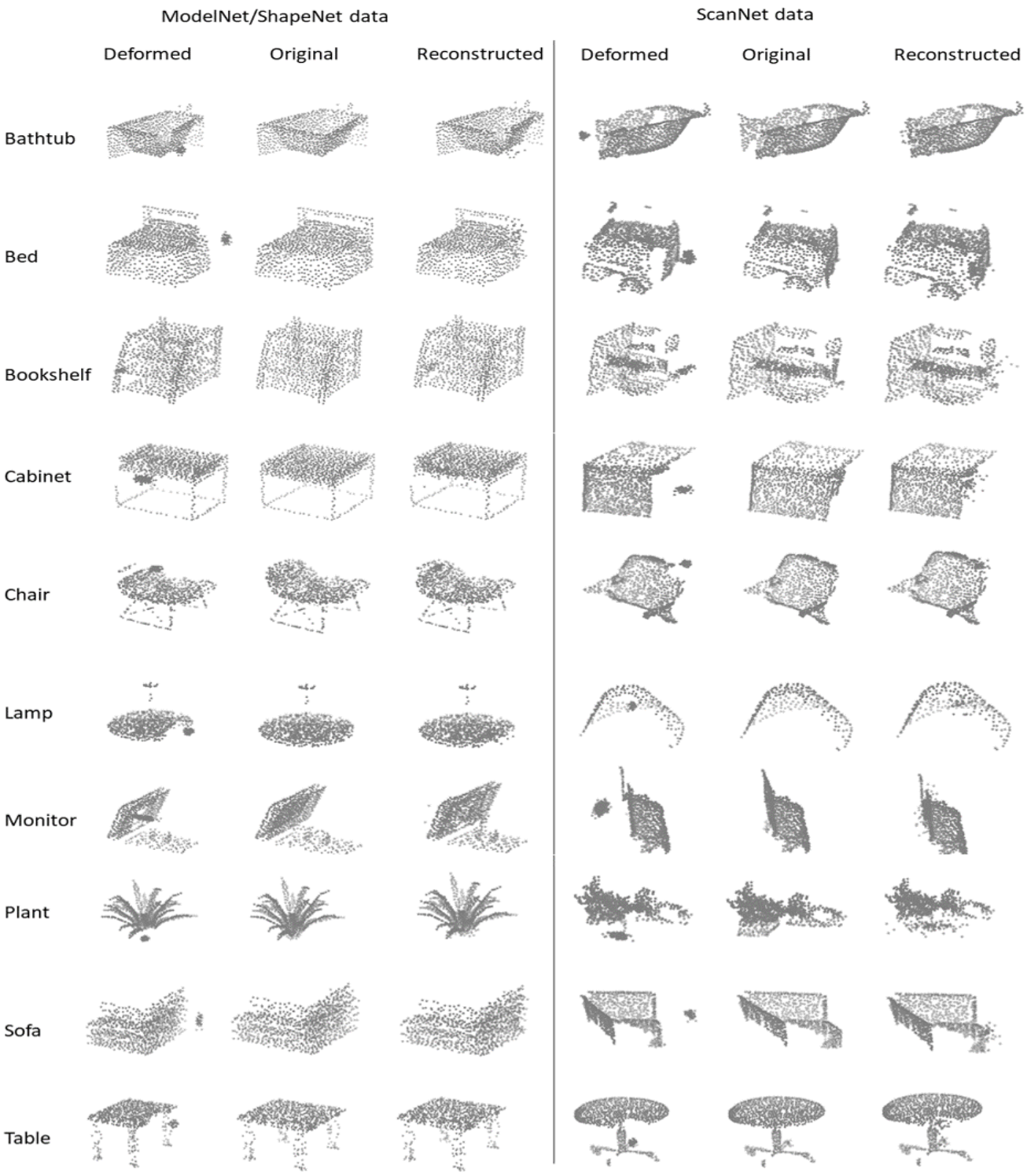


Figure 9: Illustration of target reconstruction of all classes. Each triplet shows a sample deformed using DefRec, the ground truth original, and the resulting reconstruction. Left triplets: ShapeNet/ModelNet. Right triplets: ScanNet.

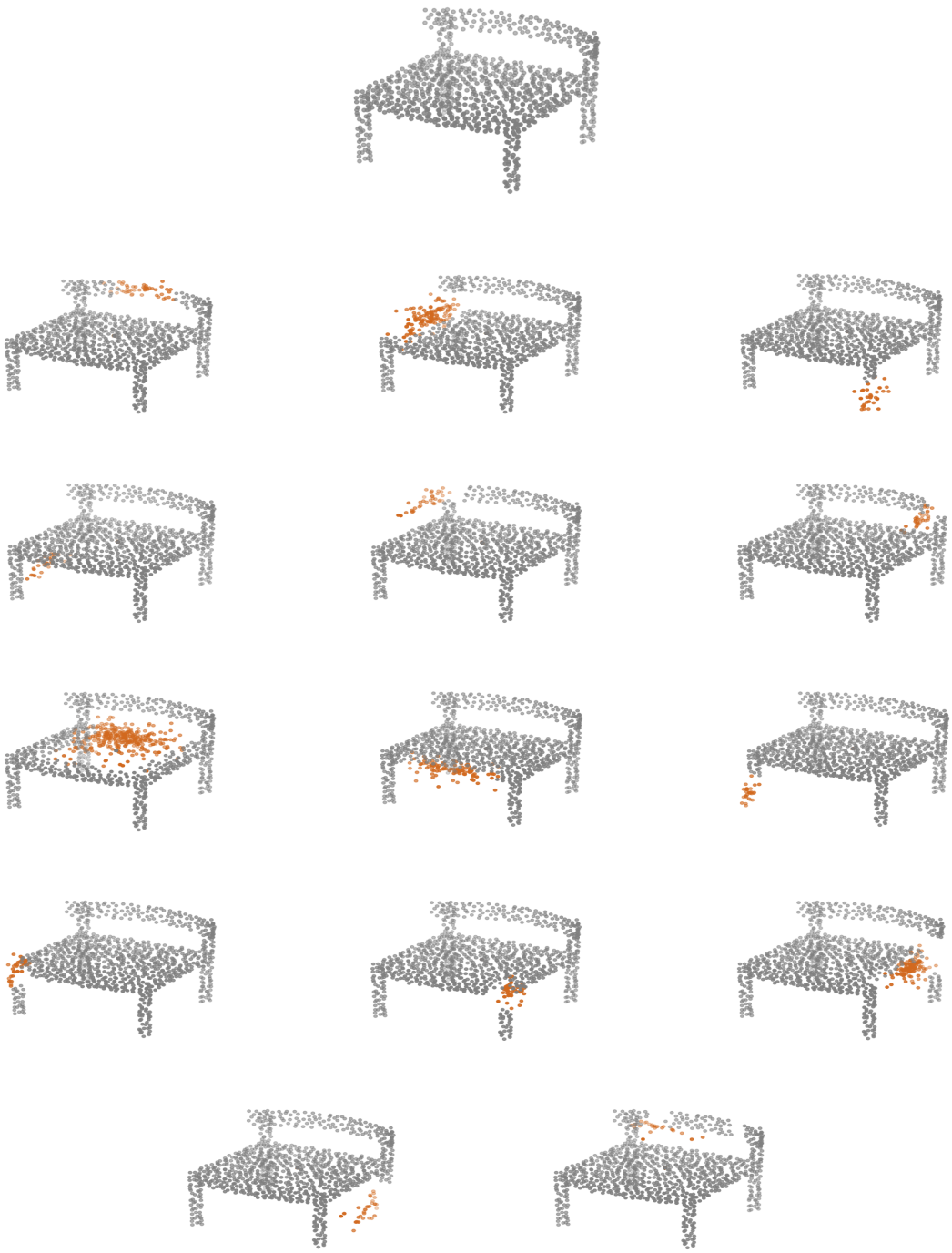


Figure 10: Reconstruction of a chair object from deformation of different regions in it by DefRec. The object in the first row is the ground truth. Below it are the reconstructed shapes, each with a deformation of different region in the object. Reconstructed region is marked by orange.

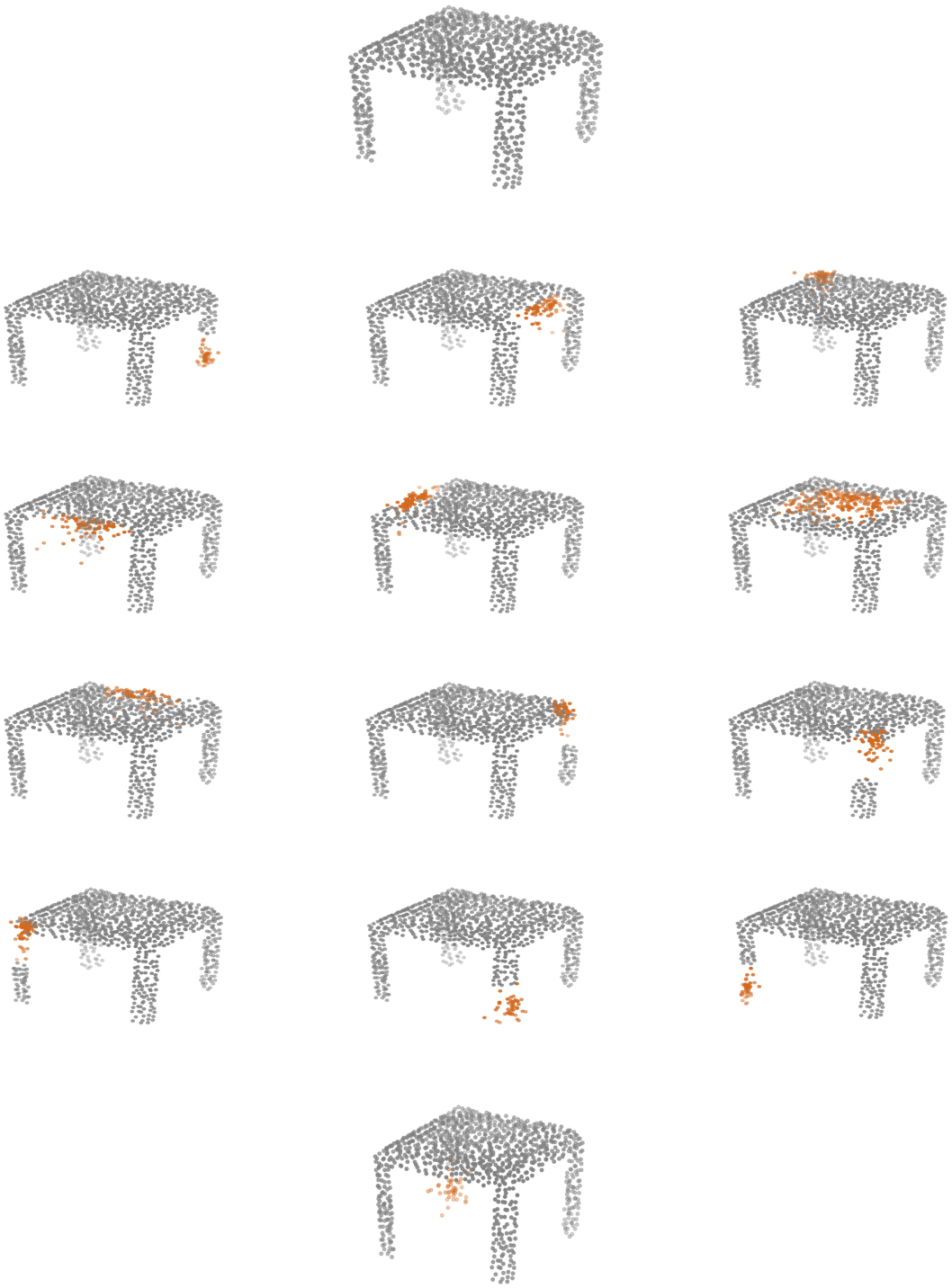


Figure 11: Reconstruction of a Table object from deformation of different regions in it by DefRec. The object in the first row is the ground truth. Below it are the reconstructed shapes, each with a deformation of different region in the object. Reconstructed region is marked by orange.

References

[1] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *The Journal of Machine Learning Research*, 17(1):2096–2030, 2016.

[2] Kaveh Hassani and Mike Haley. Unsupervised multi-task feature learning on point clouds. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 8160–8171, 2019.

[3] Pedro Hermosilla, Tobias Ritschel, Pere-Pau Vázquez, Àlvar Vinacua, and Timo Ropinski. Monte carlo convolution for learning on non-uniformly sampled point clouds. *ACM Transactions on Graphics*, 37(6):1–12, 2018.

[4] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[5] Diederik P. Kingma and Jimmy Ba. ADAM: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations*, 2014.

[6] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. PointCNN: Convolution on x-transformed points. In *Advances in neural information processing systems*, pages 820–830, 2018.

[7] Haggai Maron, Meirav Galun, Noam Aigerman, Miri Trope, Nadav Dym, Ersin Yumer, Vladimir G Kim, and Yaron Lipman. Convolutional neural networks on surfaces via seamless toric covers. *ACM Transactions on Graphics*, 36(4):1–10, 2017.

[8] 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.

[9] Can Qin, Haoxuan You, Lichen Wang, C-C Jay Kuo, and Yun Fu. PointDAN: A multi-scale 3D domain adaption network for point cloud representation. In *Advances in Neural Information Processing Systems*, pages 7190–7201, 2019.

[10] Kuniaki Saito, Kohei Watanabe, Yoshitaka Ushiku, and Tatsuya Harada. Maximum classifier discrepancy for unsupervised domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3723–3732, 2018.

[11] Jonathan Sauder and Bjarne Sievers. Self-supervised deep learning on point clouds by reconstructing space. In *Advances in Neural Information Processing Systems*, pages 12942–12952, 2019.

[12] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *Advances in neural information processing systems*, pages 4077–4087, 2017.

[13] Yi-Hsuan Tsai, Wei-Chih Hung, Samuel Schulter, Kihyuk Sohn, Ming-Hsuan Yang, and Manmohan Chandraker. Learning to adapt structured output space for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7472–7481, 2018.

[14] 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*, 38(5):1–12, 2019.