Learning Coordinate-based Convolutional Kernels for Continuous SE(3) Equivariant and Efficient Point Cloud Analysis

Supplementary Material

Algorithm 1 The coset augmentation from the coordinates

Input: coordinates set $\mathbf{X} \in \mathbb{R}^{N\times 3}$, neighborhood size K
Output: augmented cosets $\mathbf{N} \in \mathbb{R}^{N\times 3}$, s.t. $\mathbf{N} \subset S^2$

$\left\{\mathbf{x}^n,\ \{\mathbf{x}_i^n\}_{i=1}^K\right\}_{n=1}^N \leftarrow$ K-NN$(\mathbf{X}, \mathbf{X}; K)$
for $n = 1$ to N **do**
  $\{\mathbf{n}_1^n, \cdots, \mathbf{n}_K^n\} \leftarrow \{\mathbf{x}_1^n - \mathbf{x}^n, \cdots, \mathbf{x}_K^n - \mathbf{x}^n\}$
  $\mathbf{n}^n \leftarrow \frac{1}{K}\sum_i \mathbf{n}_i^n$
  if $\|\mathbf{n}^n\|_2 > 1\mathrm{e}-5$ **then**
    $\hat{\mathbf{n}}^n \leftarrow \dfrac{\mathbf{n}^n}{\|\mathbf{n}^n\|_2}$
  else
    $\hat{\mathbf{n}}^n \leftarrow$ the least eigenvector of PCA$(\{\mathbf{n}_i^n\}_{i=1}^K)$
  end if
end for
$\mathbf{N} \leftarrow \left[\hat{\mathbf{n}}^1, \cdots, \hat{\mathbf{n}}^N\right]^\top$
return $\mathbf{N}$

Algorithm 2 The process of ECKConv block

Input: coordinates $\mathcal{X}_{\text{in}} \in \mathbb{R}^{N\times 6}$, features $\mathcal{F}_{\text{in}} \in \mathbb{R}^{N\times C_{\text{in}}}$
Output: $\{x^m\}_{m=1}^M$, $\{f^n\}_{n=1}^N$

$\{x^n\}_{n=1}^N \leftarrow \mathcal{X}_{\text{in}}$; $\{f^n\}_{n=1}^N \leftarrow \mathcal{F}_{\text{in}}$
$\{x^m\}_{m=1}^M \leftarrow$ farthest-point-sampling$(\{x^n\}_{n=1}^N, M)$
Gather $\{f^m\}_{m=1}^M$ for $\{x^m\}_{m=1}^M$
$\mathcal{X}_{\text{cnt}} \leftarrow \{x^m\}_{m=1}^M$, $\mathcal{F}_{\text{cnt}} \leftarrow \{f^m\}_{m=1}^M$
$\{\mathcal{N}(x^m)\}_{m=1}^M \leftarrow$ ball-query$(\{x^m\}_{m=1}^M, \{x^n\}_{n=1}^N, k)$
Gather $\{\{f_i^m\}_{i=1}^k\}_{m=1}^M$ for $\{\mathcal{N}(x^m)\}_{m=1}^M$
$\{(f*\kappa)^m\}_{m=1}^M \leftarrow$ ECKConv$(\{x^m\}_{m=1}^M, \{\mathcal{N}(x^m)\}_{m=1}^M, \{\{f_i^m\}_{i=1}^k\}_{m=1}^M)$
$\mathcal{F}_{\text{conv}} \leftarrow \{(f*\kappa)^m\}_{m=1}^M$
return $\mathcal{X}_{\text{cnt}}$, $\mathcal{F}_{\text{cnt}}$, $\mathcal{F}_{\text{conv}}$
▷ $\mathcal{F}_{\text{cnt}}$ is utilized only in the residual connection

7. Model Architectures and Training Setups

We explain the details of ECKConv implementation and enumerate the training configurations of our models. Every ECKConv model is conducted on a single RTX 3090 GPU. Their architectures are illustrated in Figures 7a, 7b, 8a and 8b. When applying neighboring algorithms specified for point clouds, *e.g.*, farthest point sampling, k-nearest neighbors, ball query, and PCA among point groups, we utilized the implementations from PyTorch3D [28].

7.1. Details of ECKConv Block

This section delineates the detailed computation of ECKConv block, which is also illustrated in Figure 6. Given the input point cloud $\{(x^n, f^n)|x^n = (\mathbf{x}^n, \mathbf{n}^n), f^n = f(x^n)\}_{n=1}^N$, the centroids for convolution $\{(x^m, f^m)\}_{m=1}^M$, where $M \leq N$, are sampled first by farthest point sampling [26] using the coordinate $\mathbf{x}^m$s. Then, a ball query groups the local neighbors with maximal k neighbors $\mathcal{N}(x^m) = \{x_i^m | x_i^m = (\mathbf{x}_i^m, \mathbf{n}_i^m)\}_{i=1}^k$ from the input points for each centroid x^m. It gathers features in $\{f^n\}_{n=1}^N$ into $\{\{f_i^m\}_{i=1}^k\}_{m=1}^M$ by every $\mathcal{N}(x^m)$, and the ECKConv layer operates on every neighborhood and its gathered feature, yielding the output features $\{(f*\kappa)^m\}_{m=1}^M$. After the block, batch normalization and GELU activation are applied on the $\{(f*\kappa)^m\}_{m=1}^M$, and the pair of $\{(x^m, (f*\kappa)^m)\}_{m=1}^M$ is propagated as a new point cloud towards the next layer. The above process is summarized in Algorithm 2.

7.2. Coset Augmentation of Point Clouds

If the ground truth normal vector were not given in the inputs, we heuristically lift the 3D coordinate to the coset in S^2 as Algorithm 1. It averages every difference between the neighbor points and the centroid and normalize the average into a unit vector unless its norm is not too small. If it is, it exploits PCA on the neighbors to estimate the normal vector for the centroid. We also experimented exploiting only the normal vectors estimated by PCA before the ECKConv layers, but Algorithm 1 empirically performed better in the ModelNet40 experiments. We applied this to every our models except **ECKConv-Normal** in the ModelNet40 classification and the S3DIS segmentation experiments.

7.3. Point Feature Propagation Module

We adopt this module suggested by Qi et al. [26] in our segmentation models, up-sampling point features from a *coarse* point cloud $\mathcal{X}' \in \mathbb{R}^{M\times 3}$ onto a *fine* one $\mathcal{X} \in \mathbb{R}^{N\times 3}$ such that $N > M$. Querying the top-K nearest points from the coarse point cloud per each fine point, local features from queried points are gathered with normalized weights reciprocal to distances and concatenated to features of the query point:

$$f_i^{\text{FP}} = \frac{\sum_{k=1}^K w_{ik} f'_{ik}}{\sum_{k=1}^K w_{ik}}, \quad w_{ik} = \frac{1}{\|x_i - x'_{ik}\|_2^2}, \tag{13}$$

where $x_i \in \mathcal{X}$, and $x'_{ik} \in \mathcal{X}'$ is queried to x_i by KNN with corresponding feature f'_{ik}. Then the upsampled feature f_i^{FP} is concatenated to f_i, the feature of x_i before the feature

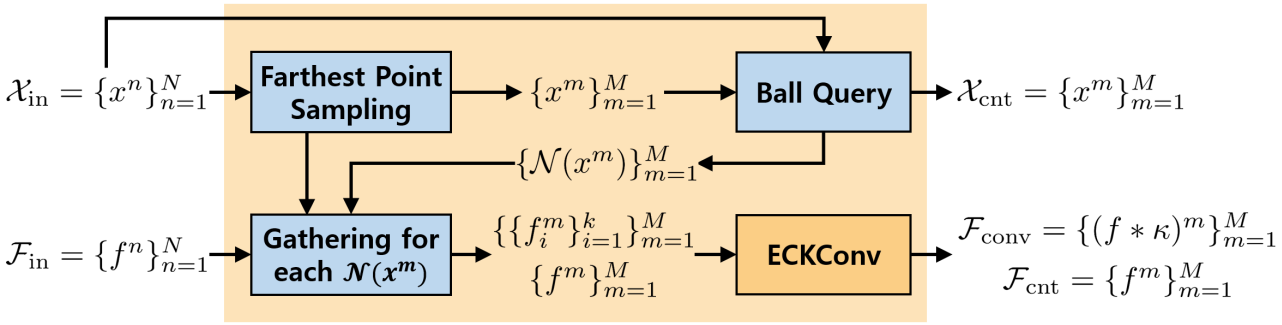


Figure 6. Detailed architectures of ECKConv block. We substitute abstract variables from Figure 3 with actual computational variables.

propagation, and the module propagates a new point feature set $\left\{\left[f_i, f_i^{\text{FP}}\right]\right\}_{i=1}^N$ to the following ECKConv layer. We set $K = 1$ and $K = 3$ for object part and indoor semantic segmentation model each.

7.4. Training Configurations

7.4.1. ModelNet40 Classification

We let every baseline training follow its original configuration of training: the number of input points, whether to utilize a normal vector, augmentation except SO(3) rotation, and other optimizer hyperparameters. For our method, we sampled 1024 points for each point cloud and applied the scaling augmentation during the training, *i.e.*, the scales of the object in XYZ directions were randomly rescaled in range $(\frac{2}{3}, 1.5)$. Each minibatch contained 16 point clouds, and its cross entropy loss was smoothed with parameter 0.2 and minimized by Adam optimizer, whose initial learning rate was $1\text{e}-4$ and scheduled to $1\text{e}-6$ by cosine annealing scheduler during 200 epochs.

7.4.2. ModelNet40 Pose Registration

Our ECKConv layers extracted local features from the two distinct inputs, named the source and target. We adopted Deep Closest Point [36] to determine the proper interpolation value of source coordinates that match the target coordinates. A variant of the transformer encoder-decoder structure plays a critical role in DCP to find the interpolation score between points. Every deep learning module here shares its parameters for both source and target. Finally, the relative pose $\mathbf{T}_{\text{pred}} = (\mathbf{R}_{\text{pred}}, \mathbf{t}_{\text{pred}})$ from the source to target can be readily computed by applying the singular value decomposition and mean difference. Following Wang and Solomon [36], the following loss was minimized:

$$\mathcal{L}(\mathbf{T}_{\text{pred}}, \mathbf{T}_{\text{gt}}) = \|\mathbf{R}_{\text{pred}}\mathbf{R}_{\text{gt}}^\top - \mathbf{I}\|_2^2 + \|\mathbf{t}_{\text{pred}} - \mathbf{t}_{\text{gt}}\|_2^2. \quad (14)$$

The training was conducted for 50 epochs, and each minibatch contains 16 pairs without augmentation. Adam optimizer minimizes the loss with $1\text{e}-4$ initial learning rate scheduled to $1\text{e}-6$ by cosine annealing scheduler.

7.4.3. ShapeNet Part Segmentation

We configured the number of points in a point cloud to 2048 in every model for fair comparison, since the number of samples in a point cloud affects the IoU metrics. It might influence the performance discrepancy of PRLC [22] between its reported result and our experiment, whose number of points is 1024 in its original implementation. We applied an identical augmentation protocol from our classification model training. A cross-entropy loss was computed on every point of each input and smoothed with parameter 0.2. The loss on minibatch containing 16 point clouds was minimized by Adam optimizer with hyperparameter $\beta_1 = 0.5$, and its initial learning rate was scheduled from $1\text{e}-4$ to $1\text{e}-6$ by the cosine annealing during 250 epochs.

7.4.4. S3DIS Semantic Segmentation

As stated in Section 5.4, we adopted the training protocols by Dong et al. [10], Su et al. [32], Zhang et al. [44] to crop a $4m^2$ region and sample 4096 points within it. A single room in the area was resampled multiple times in a single epoch to cover its whole scene with the cropped regions. The cropped regions were randomly scaled in range $(\frac{2}{3}, 1.5)$ along the XY directions and flipped along the X-axis with probability 0.5 during the training. Besides, the label weight for cross-entropy loss was precomputed since the imbalance among the semantic category is severe in the S3DIS. The cross entropy loss was minimized by Adam optimizer with learning rate $1\text{e}-4$ during 80 epochs.

8. Setups for CSEConv Baselines

Since CSEConv [20] is our main baseline with respect to the scalability, we detail the setups for CSEConv baselines in the pose registration and part segmentation, which were not experimented in its original paper. Every CSEConv baseline followed the initial feature generation protocol suggested by Kim et al. [20].

Table 6. Memory and time costs during the training, compared between ECKConv and CSEConv baselines from the pose registration and part segmentation tasks.

	Models	Memory (GB) ↓	#Batch/sec ↑	batch size	#params
pose reg	CSEConv	39.09	3.37	16	10.8M
	ECKConv	5.37	7.39	16	4.1M
part seg	CSEConv	66.46	2.04	6	14.9M
	ECKConv	10.26	3.80	16	22.5M

8.1. Pose Registration

As stated in the main paper, it shared the identical model and training configurations to the ECKConv model, e.g., the basis dimension of coordinate-based networks (though CSEConv uses random Fourier feature [27]), the number of layers, the dimension of each hidden layer, the cardinality of neighbors per layer, and training hyperparameters. However, 6 GPUs were required to train this model in our workstation. It shifted both source and target point clouds to the origin by subtracting geometric means since CSEConv only preserves SO(3) symmetry. After the extraction, the model replaced down-sampled point clouds to the original center coordinates before DCP method.

8.2. Part Segmentation

Our CSEConv baseline also followed U-Net style architecture but had only three down-sampling and up-sampling layers each. It shared the identical training hyperparameters to the ECKConv experiment except for the batch size and training epochs, which reduced to 6 point clouds and 100 epochs. This configuration also utilized 6 GPUs for the training in our workstation.

8.3. Scalability Comparisons

We additionally measure the cost of CSEConv baselines during the training and compare them with our models in Table 6. Since the training setups which affected their efficiency varied among experiments, their batch and parameter sizes are reported as well.

We observed the consistency in their training efficiency, which matches the scalability statement in Proposition 4.1. When models shared the identical architecture as the pose registration task, ECKConv was more efficient than CSEConv both in memory and time costs. Our method also consumed fewer resources than CSEConv while processing more parameters and inputs in the part segmentation experiment. These results supplement the scalability of ECKConv, which effectively exploits computation resources and is scalable to obtain sufficient model capacity.

Table 7. Ablation experiments on the hyperparamters of ECKConv in the ModelNet40 classification task, where models received randomly rotated inputs only during the evaluation ($I/\mathrm{SO}(3)$). Default values are $A = 22$, $\Psi \in \mathbb{R}^{64}$, and $K = 32$.

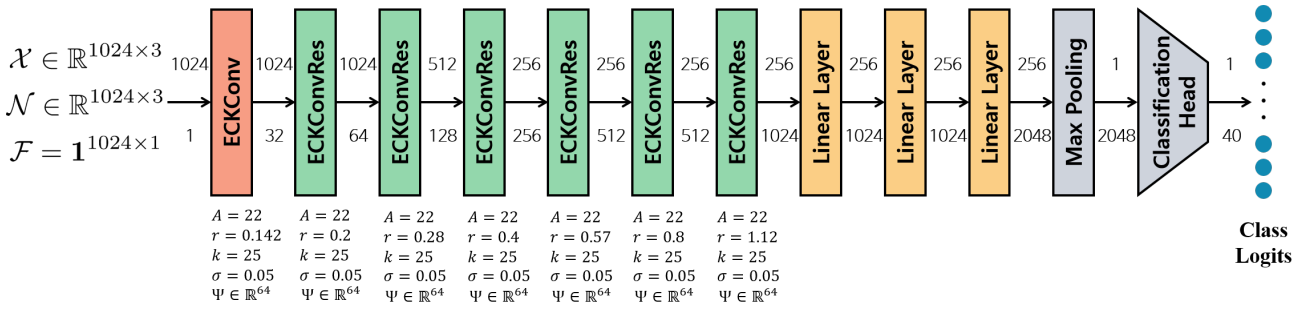
Ablation Studies	#Anchor Bases		Gau$(\cdot)$ Bandwidth		#Neighbors for Alg 1.	
	$A = 1$	$A = 64$	$\Psi \in \mathbb{R}^4$	$\Psi \in \mathbb{R}^{256}$	$K = 8$	$K = 128$
Acc (%)	89.22	90.19	88.86	90.32	90.32	89.95
Mem (GB) ↓	4.36	6.57	1.90	14.51	4.90	4.90
#Batch/sec ↑	10.20	5.62	10.65	2.69	7.58	6.83

9. Ablations on Hyperparameter Sensitivity

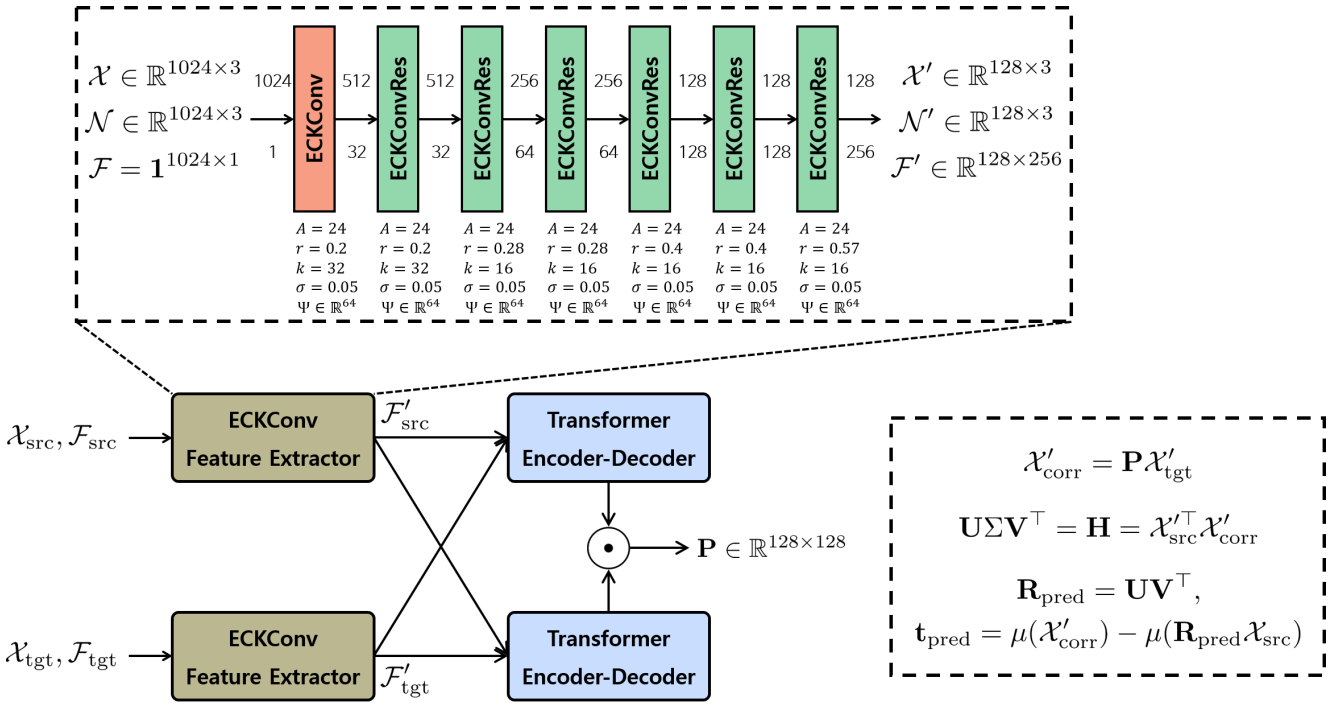
As the kernel structure and the coordinate lifting to coset space of ECKConv have diverse hyperparameters, we conducted ablation experiments in the ModelNet40 classification task to verify the sensitivity of ECKConv against such hyperparameters. The experiments varied the number of anchor bases (A), the bandwidth of Gaussian embedding (Ψ), and the number of neighbors during the coset lifting in Algorithm 1 (K) to extreme values. Table 7 demonstrates that the efficiencies of the variants were affected significantly when the hyperparameters related to the kernel (A, Ψ) varied, but their performances remained relatively robust. We conjecture that linear layers from skip connections in our method help maintain their performances.

10. Discussion on the Limitations

Although ECKConv proposes an SE(3)-symmetric and scalable convolution applicable to large-scale point cloud problems, its kernel is isotropic to SE(3) actions due to the utilization of the double coset element, which is composed of coset elements grouped by left actions of SO(2), and the confinement to scalar-type features. These factors decrease the directional expressivity of ECKConv to certain types of point cloud tasks where the point feature should be anisotropic, *e.g.*, normal estimation, molecule structure prediction, and n-body problems. Expanding the resolvable domain of ECKConv would be an important future research direction that balances our scalable architecture with the inefficient yet expressive kernel designs from the previous steerable convolution studies.

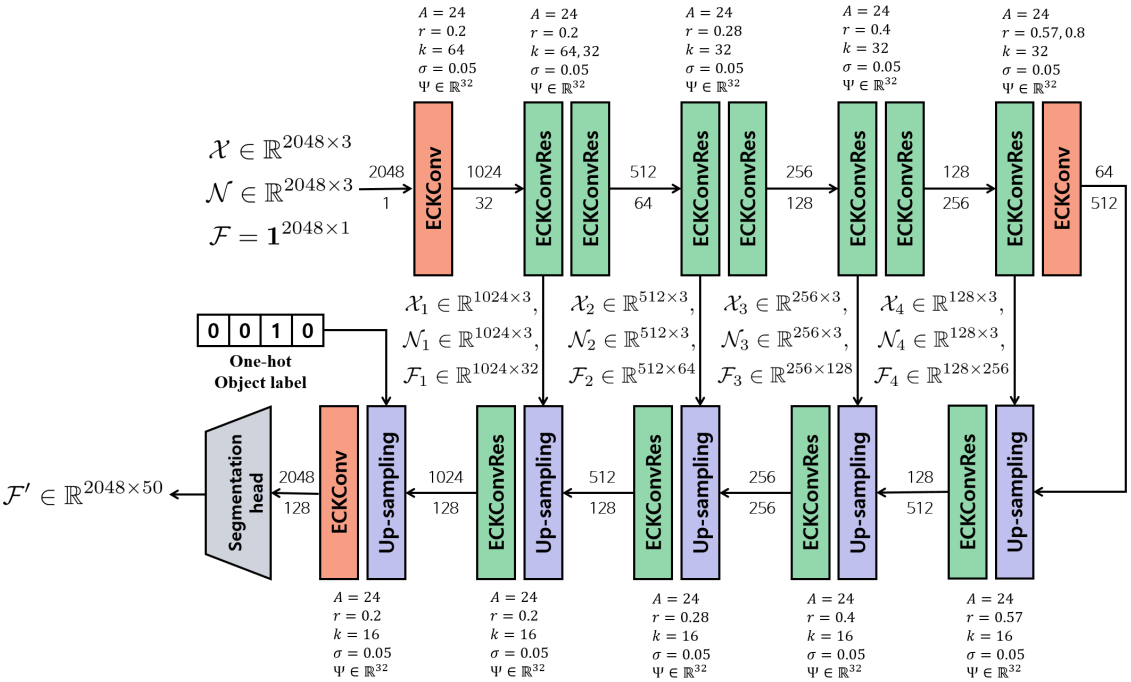


(a) Classification model.

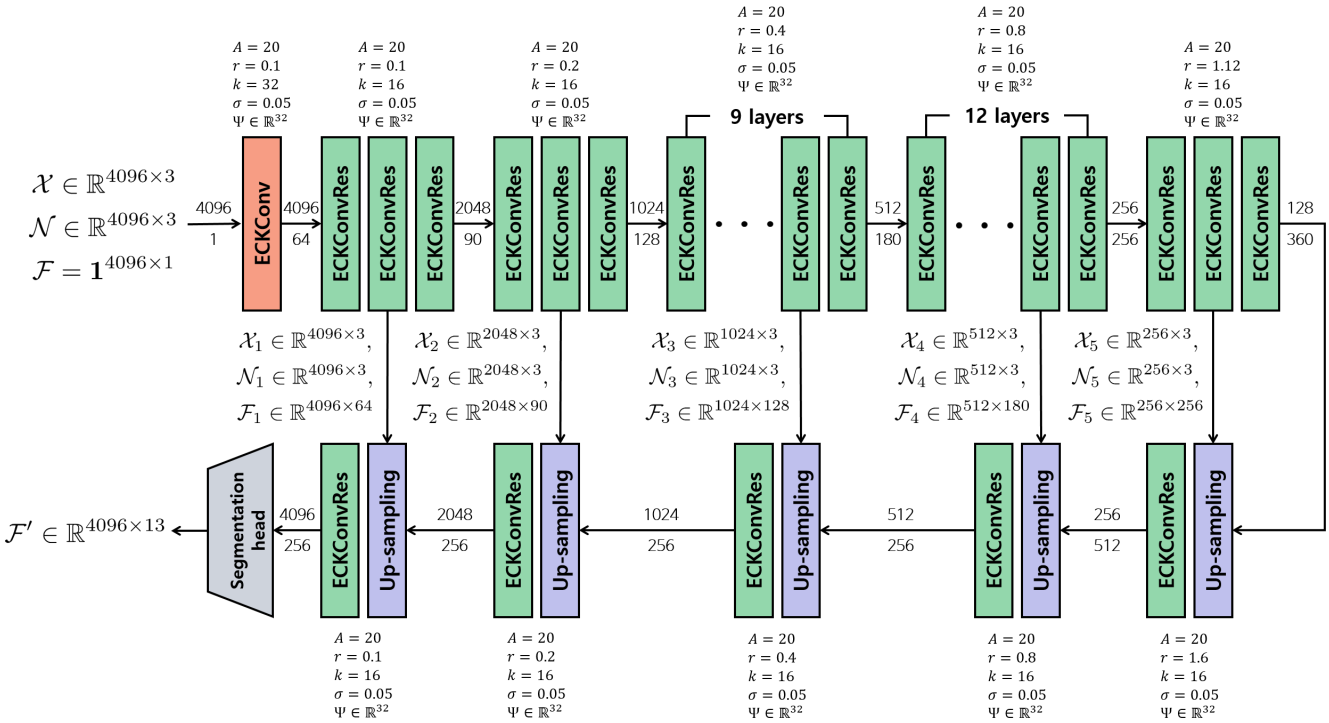


(b) Pose registration model.

Figure 7. Architectures of ECKCov models for ModelNet40 [42] experiments. Since point clouds only contain 3D coordinates, we initialize its point feature as one vector per every point. An arrow between ECKConv layers designates the number of points and the dimension of each feature on its above and below. We also specify ECKConv hyperparameters below, such as the number of anchor points (A), a ball query radius (r), the maximal number of sampled points (k), the standard deviation of Gaussian kernel (σ), and the basis dimension of Gaussian embedding (Ψ) [47, 48]. Linear layers map every point feature with weight-sharing networks, and we omit batch normalization and activation next to these layers in the above figures.



(a) Part segmentation model for ShapeNet [3].



(b) Semantic segmentation model for S3DIS [1].

Figure 8. Architectures of ECKConv models fors segmentation tasks in the ShapeNet and S3DIS. We applied U-Net [31] style architecture and up-sampled point features using Point Feature Propagation [26] module, which assigns coarse point features to its top-K closest neighbors skipped from previous layers, weighted by the reciprocal of the point distance in the coordinate space.

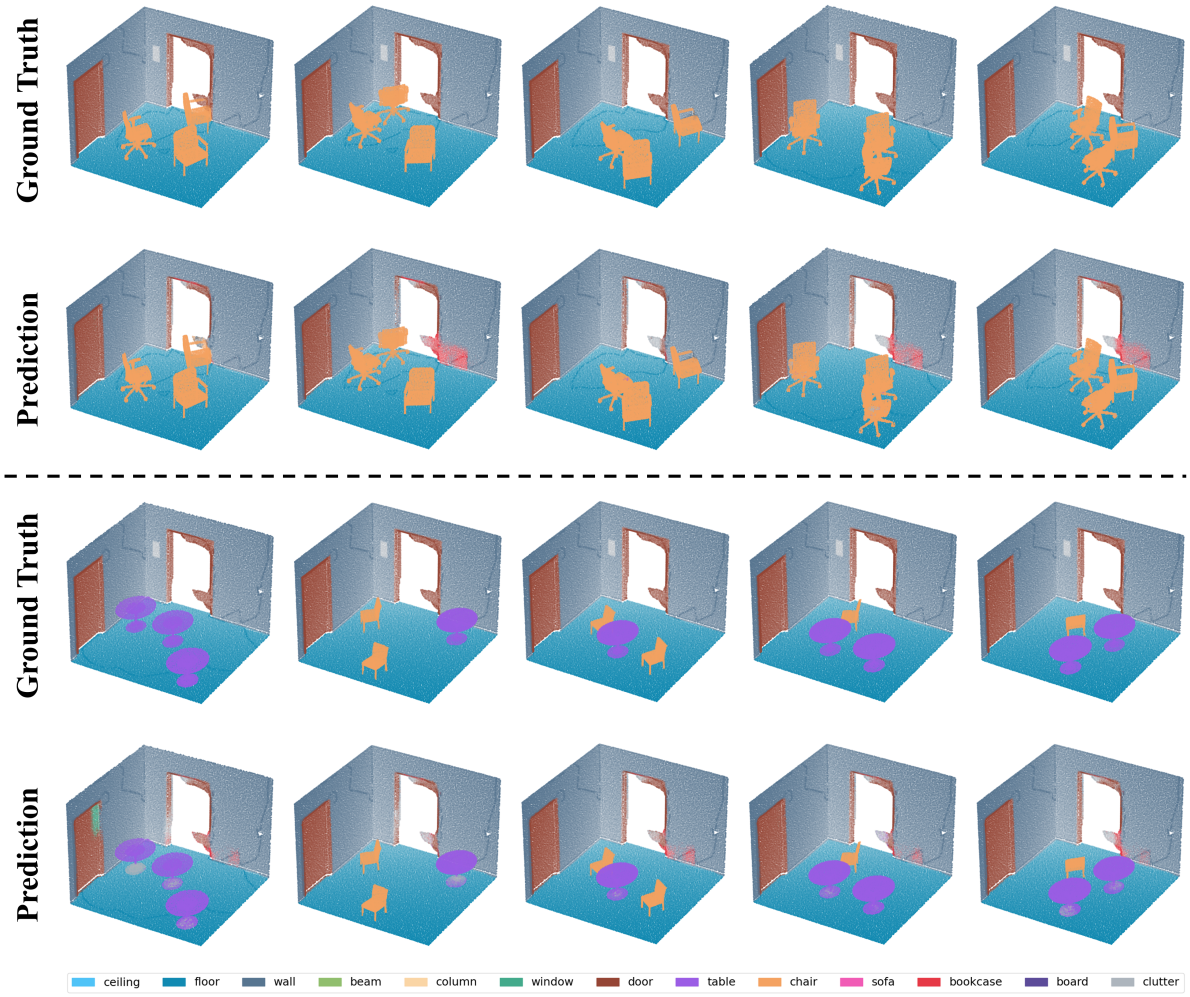


Figure 9. Qualitative results in S3DIS simulation experiments. (top) We diversified the objects from the chair category of ModelNet40. Our model consistently succeeded to discern chair objects with different shapes and poses. (bottom) We diversified the object category, adding the table object from ModelNet40. Our model showed consistent results on segmenting the table objects regardless of their numbers or poses, although noisy results on the bottom part of tables were also observed.