Implicit Autoencoder for Point-Cloud Self-Supervised Representation Learning – Supplementary Material

We provide additional descriptions of pre-training details in Section 1, and downstream task details in Section 2. We discuss hypothetical concerns in Section 3. We present more experiment results and analysis in Section 4, and proof of propositions in Section 5.

1. Pre-training Details

1.1. Scene Data Generation

To prepare the data for scene-level pre-training, we pre-process the point clouds from ScanNet dataset [2]. We utilize the sliding window strategy and crop each scene instance into $d \times d \times d$ point cloud cubes, where $d = 3.0$ m. Consistent with Qi et al. [6], we divid the dataset into training (consisting of 8k point clouds) and validation (consisting of 2.6k point clouds) splits. We randomly sub-sample 10,000 points from each point cloud to use as input to the encoder.

Given the lack of high-quality water-tight meshes, we employ the k-nearest neighbor (KNN) strategy [5] to calculate ground-truth unsigned distance functions. Specifically, for each sample point $\mathbf{x}$, we use the KNN algorithm to find the three nearest points from the point cloud and calculate the average distance as the unsigned distance value.

1.2. Decoder Details

As mentioned in Section 4.2 of the main body, the IAE encoder maps a given point cloud into a high-dimensional code, and the decoder fits an implicit function that converts the latent code and a query point to an implicit function value. We explore two different designs for the decoder: the Occupancy Network style [3] and the Convolutional Occupancy Network style [4].

Occupancy Network style: We use a fully-connected neural network with five ResNet blocks to implement the decoder, taking the latent code from the encoder and the sample point $\mathbf{x}$ as input. For detailed information, please refer to [3].

Convolutional Occupancy Network style: First, we decode the latent code from the encoder to a volumetric feature representation. Then, we use trilinear interpolation to obtain the feature of the sample point $\mathbf{x}$. Next, we use a small fully-connected occupancy network to predict the implicit function value, which comprises multiple ResNet blocks. Please refer to [4] for further details.

1.3. Training Details

We implement all models in PyTorch and use the Adam optimizer without weight decay. The learning rate is set to 10^{-4} for all datasets. We use standard data augmentation techniques proposed by [13] for both ShapeNet and ScanNet pre-training. For ShapeNet, we pre-train the models for 600 epochs. For ScanNet, we pre-train the models for 1,000 epochs.

2. Downstream Task Details

After pre-training, the pre-trained encoder is transferred to downstream tasks.

2.1. Shape Classification

We respectively sample 1,024 and 2,048 points from each 3D shape in ModelNet40 [12] and ScanObjectNN [8] and utilize the 3-channel coordinates as input. The same training settings are adopted for the two datasets. For the DGCNN [11] backbone, we fine-tune the network for 200 epochs with a batch size of 32 and set the learning rate as 1×10^{-3} with a weight decay of 5×10^{-1}. For the M2AE [14] backbone, we fine-tune the network for 300 epochs with a batch size of 32 and set the learning rate as 5×10^{-4} with a weight decay of 5×10^{-2}.

2.2. 3D Object Detection

We replicate the settings used in the original paper. To train VoteNet [6], we fine-tune the network using an Adam optimizer with a batch size of 8 and an initial learning rate of 0.001. After 80 epochs, we reduce the learning rate by a factor of 10 and then again by another factor of 10 after 120 epochs.

To train CAGroup3D [9] on both ScanNet and SUN RGB-D datasets, we set the batch size, initial learning rate, and weight decay to 16, 0.001, and 0.0001, respectively. For ScanNet, we fine-tune for 120 epochs and reduce the learning rate by a factor of 10 at the 80^{th} epoch and again at the 110^{th} epochs. For SUN RGB-D, we fine-tune for 48 epochs and reduce the learning rate at the 32^{th} and the 44^{th} epochs.

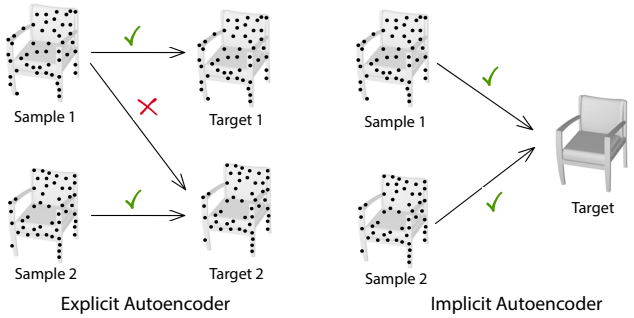

Figure 6: **Comparison between Explicit Autoencoder and Implicit Autoencoder**. Given a 3D shape, we randomly take two samples. For Implicit Autoencoder, we use 3D CAD model to represent the implicit function target for better visualization.

2.3. Indoor 3D Semantic Segmentation

We present the evaluation of our model on the S3DIS dataset using 6-fold cross-validation [1]. We replicate the settings in the original paper for consistency.

To train the DGCNN model, we adopt a batch size of 32 and an initial learning rate of 1×10^{-1} using a stochastic gradient descent optimizer. We reduce the learning rate gradually until it reaches 0.001 using cosine annealing.

For the PointNeXt [7] model, we select the PointNeXt-XL version and fine-tune it with an AdamW optimizer, where the weight decay is 10^{-4}, the initial learning rate is 0.001, and the batch size is 32. We use the cosine annealing method to reduce the learning rate during training.

3. Discussion

3.1. What is the difference between data augmentation and sampling variation?

Sampling variation refers to the fact that different point cloud samples of the same 3D shape contain different noises induced from various sources, such as intrinsic noises from sensors and interference from the environment. In the explicit autoencoding paradigm, the encoder is required to encode not only the 3D geometry but also information about the specific discrete sampling of the 3D shape into the latent code, which can lead to sampling variation. This is because the decoder must reconstruct a point cloud that matches the original point cloud perfectly. For example, as shown in Figure 6, given a 3D shape and two randomly selected samples, the explicit autoencoder forces the decoder to output the same sample as the input. If the target sample changes, the loss increases due to incorrect mapping, even though the different samples represent the same shape. Thus, the explicit autoencoding paradigm must learn the mapping that includes sampling variation. In contrast, the implicit autoencoder does not face this problem because different samples map to the same target, which is the implicit representation of the 3D shape.

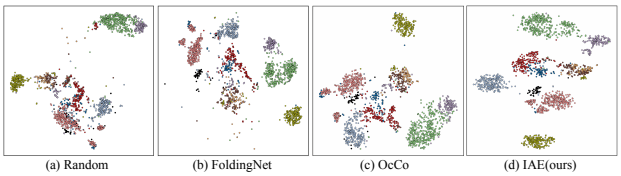

Figure 7: **Visualization of learned features.** We visualize the learned features for each sample in ModelNet10 using t-SNE. All the models use DGCNN as the encoder backbone. (a) uses random initialization. (b), (c), (d) are pre-trained on ShapeNet.

Data augmentation is a standard technique for point-cloud learning, including rotation, translation, scaling, and sub-sampling. As mentioned in the supplement's Section 1.3, IAE uses all of these standard data augmentation methods during pre-training, as other approaches do. For instance, IAE uses different sub-samples of the same 3D shape as input, but all of these sub-samples share the same target, which is the implicit function of the 3D shape.

3.2. Is it a fair comparison with existing self-supervised learning approaches?

In this subsection, we discuss the fairness of the comparison between the Implicit Autoencoder (IAE) and existing self-supervised learning approaches. Our approach differs from existing methods in that we use dense point clouds (50k points) or mesh data from ShapeNet to generate the implicit function label, while existing approaches utilize sparse, sub-sampled point clouds (i.e., 1k $\sim$ 2k points) for pre-training. We argue that the point of pre-training is to extract as much information as possible from all available data. Therefore, it is natural to include dense point clouds or mesh data in the pre-training dataset. However, explicit autoencoders lack the ability to efficiently exploit the density, as demonstrated in the experiment analysis in Section 4.5 of the main paper. Hence, we consider our approach to have an advantage rather than an unfair comparison.

We also emphasize that we use the same setting as existing self-supervised learning approaches for downstream task training to ensure a fair comparison. Therefore, we are confident that the comparison between our approach and existing approaches is fair.

4. More Results

4.1. Embedding Visualization.

We visualize the learned features of our model and baseline approaches in Figure 7. We compare with FoldingNet [13], OcCo [10], and a sanity-check baseline, random initialization. Random initialization use randomly initialized network weights to obtain the embedding, and its performance explains the network prior. The embeddings

for different categories in the ModelNet10 dataset are shown using t-SNE dimension reduction. Empirically, we observe that our pre-trained model provides a cleaner separation between different shape categories than FoldingNet [13], OcCo [10], and random initialization.

5. Proof of Propositions

Denote

$$X = (\boldsymbol{x}_1, \cdots, \boldsymbol{x}_N), \qquad X' = (\boldsymbol{x}'_1, \cdots, \boldsymbol{x}'_N).$$

To obtain closed-form expressions of the linear auto-encoding problem, we reformulate the optimization problem as

$$\min_{R,B\in\mathbb{R}^{n\times m}, R^TR=I_m} \sum_{k=1}^{N} \|RB^T\boldsymbol{x}'_k - \boldsymbol{x}_k\|^2 \tag{9}$$

The following proposition specifies the optimal solution to (9).

The optimal solution $(R^\star, B^\star), R, B \in \mathbb{R}^{n\times m}$ to (9) satisfies that the columns of $R^\star$ are the leading m eigenvectors with the largest eigenvalues of

$$(X'X)^T(X'X'^T)^+(X'X^T).$$

Moreover,

$$B^\star = (X'X'^T)^+(X'X^T)R^\star.$$

Proof. Denote $R = (\boldsymbol{r}_1, \cdots, \boldsymbol{r}_m)$ and $B = (\boldsymbol{b}_1, \cdots, \boldsymbol{b}_m)$. Then

$$\begin{aligned}
&\sum_{k=1}^{N} \|RB^T\boldsymbol{x}'_k - \boldsymbol{x}_k\|^2 \qquad (10)\\
&= \sum_{k=1}^{N} \Big(\boldsymbol{x}'^T_k BB^T\boldsymbol{x}'_k - 2(R^T\boldsymbol{x}_k)^T(R^T\boldsymbol{x}'_k) + \|\boldsymbol{x}_k\|^2 \Big)\\
&= \sum_{k=1}^{N} \Big(\sum_{j=1}^{m} \big((\boldsymbol{b}_j^T\boldsymbol{x}'_k)^2 - 2(\boldsymbol{r}_j^T\boldsymbol{x}_k)(\boldsymbol{b}_j^T\boldsymbol{x}'_k)\big) + \|\boldsymbol{x}_k\|^2 \Big)\\
&= \sum_{j=1}^{m} \Big(\boldsymbol{b}_j^T \big(\sum_{k=1}^{N} \boldsymbol{x}'_k\boldsymbol{x}'^T_k \big)\boldsymbol{b}_j - 2\big(\sum_{k=1}^{N} \boldsymbol{x}'_k\boldsymbol{x}_k^T\boldsymbol{r}_j\big)^T\boldsymbol{b}_j \Big)\\
&+ \sum_{k=1}^{N} \|\boldsymbol{x}_k\|^2 \qquad (11)
\end{aligned}$$

Therefore, define

$$\{\boldsymbol{b}_j^\star, 1 \le j \le m\} = \operatorname*{argmin}_{\boldsymbol{b}_j} \sum_{k=1}^{N} \|RB^T\boldsymbol{x}'_k - \boldsymbol{x}_k\|^2.$$

Since $\boldsymbol{b}_j$ lies in the column space of X', it is clear that the optimal solution $\boldsymbol{b}_j^\star$ is given by

$$\begin{aligned}
\boldsymbol{b}_j^\star &= \Big(\sum_{k=1}^{N} \boldsymbol{x}'_k\boldsymbol{x}'^T_k\Big)^+ \Big(\sum_{k=1}^{N} \boldsymbol{x}'_k\boldsymbol{x}_k^T\Big)\boldsymbol{r}_j\\
&= (X'X'^T)^+(X'X^T)\boldsymbol{r}_j, \qquad 1 \le j \le m. \qquad (12)
\end{aligned}$$

Substituting (12) into (10), we have that the objective function becomes

$$\begin{aligned}
&\sum_{k=1}^{N} \|RB^{\star T}\boldsymbol{x}'_k - \boldsymbol{x}_k\|^2\\
=&\sum_{j=1}^{m} \boldsymbol{r}_j^T \Big(XX'^T\Big)\Big(X'X'^T\Big)^+\Big(X'X^T\Big)\boldsymbol{r}_j\\
&- 2\sum_{j=1}^{m} \boldsymbol{q}_j^T \Big(XX'^T\Big)^T\Big(X'X'^T\Big)^+\Big(X'X^T\Big)\boldsymbol{r}_j + \sum_{k=1}^{N} \|\boldsymbol{x}_k\|^2\\
=&-\sum_{j=1}^{m} \boldsymbol{r}_j^T \Big(X'X^T\Big)^T\Big(X'X'^T\Big)^+\Big(X'X^T\Big)\boldsymbol{r}_j + \sum_{k=1}^{N} \|\boldsymbol{x}_k\|^2
\end{aligned} \tag{13}$$

Therefore, the optimization problem in (9) reduces to

$$\max_{\substack{R\in\mathbb{R}^{n\times m},\\ R^TR=I_m}} \text{Trace}\left(R^T\Big(X'X^T\Big)^T\Big(X'X'^T\Big)^+\Big(X'X^T\Big)R\right) \tag{14}$$

It is easy to see that the optimal solution $R^\star = (\boldsymbol{r}_1^\star, \cdots, \boldsymbol{r}_m^\star)$ to (9) satisfies that $\boldsymbol{r}_i^\star, i \in [m]$ are the leading m eigenvectors of $C_{X',X} := \Big(X'X^T\Big)^T\Big(X'X'^T\Big)^+\Big(X'X^T\Big)$. □

5.1. Proof of Proposition 1

We show that when $\epsilon_k \in \{L\}^\perp, k \in [N]$,

$$R^\star = B^\star = Q.$$

Therefore, the formulation of (1) is identical to that of (9). In fact, consider the singular value decomposition (SVD) of

$$X' = U\Sigma V^T.$$

First,

$$\begin{aligned}
X^TX &= X^TX'\\
&= X^TU\Sigma V^T.
\end{aligned}$$

Therefore, V is an orthonormal basis of the column space of X^T. This means we can write out the SVD of $X = U'\Sigma'V^T$. Again using $X^TX = X^TU\Sigma V^T$, we have

$$V\Sigma'^2V^T = V\Sigma'U'^TU\Sigma V^T.$$

It follows that

$$\Sigma'^2 = \Sigma'U'^TU\Sigma.$$

In other words,

$$U'\Sigma' = U\Sigma.$$

This means we can arrange the SVD of X so that

$$U' = U, \Sigma' = \Sigma.$$

We proceed to show that $(XX'^T)(X'X'^T)^+(X'X^T) = XX^T$. In fact,

$$\begin{aligned}
&(XX'^T)(X'X'^T)^+(X'X^T)\\
&= XV\Sigma U^T\big(U\Sigma V^TV\Sigma U^T\big)^+(U\Sigma V^TX^T)\\
&= XV\Sigma U^T\big(U\Sigma^2U^T\big)^+(U\Sigma V^TX^T)\\
&= XV\Sigma U^TU\Sigma^{-2}U^TU\Sigma V^TX^T\\
&= XVV^TX^T\\
&= XV\Sigma U^T(U\Sigma^{-2}U^T)U\Sigma V^TX^T\\
&= XV\Sigma U^T(U\Sigma^2U^T)^+U\Sigma V^TX^T\\
&= XX^T(XX^T)^+XX^T = XX^T.
\end{aligned}$$

Moreover,

$$\begin{aligned}
&(X'X'^T)^+(X'X^T)\\
&= (U\Sigma VV^T\Sigma U^T)^+(U\Sigma V^TV\Sigma U)\\
&= (U\Sigma^2U^T)^+(U\Sigma^2U)\\
&= I_n.
\end{aligned}$$

□

5.2. Proof of Proposition 2

Let us first consider the case where the corresponding eigenvalues of Q are distinctive. Let $\boldsymbol{q}_j, m+1 \le j \le n$ expand the columns of Q to form an orthonormal basis of $\mathbb{R}^n$. In this case, applying the derivative formula of eigenvectors to each eigenvector and the fact that $\epsilon_k \in \{\, L \,\}^\perp$, we obtain

$$d\boldsymbol{q}_i = -\sum_{j\neq i}\frac{\boldsymbol{q}_j^T\sum_{k=1}^N(\boldsymbol{x}_k\epsilon_k^T+\epsilon_k\boldsymbol{x}_k^T)\boldsymbol{q}_i}{\lambda_j-\lambda_i}\boldsymbol{q}_j \tag{15}$$

$$= -\sum_{j\neq i, j=1}^{m}\frac{\boldsymbol{q}_j^T\sum_{k=1}^N(\boldsymbol{x}_k\epsilon_k^T+\epsilon_k\boldsymbol{x}_k^T)\boldsymbol{q}_i}{\lambda_j-\lambda_i}\boldsymbol{q}_j \tag{16}$$

$$-\sum_{j=m+1}^{n}\frac{\boldsymbol{q}_j^T\sum_{k=1}^N(\boldsymbol{x}_k\epsilon_k^T+\epsilon_k\boldsymbol{x}_k^T)\boldsymbol{q}_i}{\lambda_j-\lambda_i}\boldsymbol{u}_j$$

$$= -\sum_{j=m+1}^{n}\frac{\boldsymbol{q}_j^T\sum_{k=1}^N\epsilon_k\boldsymbol{x}_k^T\boldsymbol{q}_i}{\lambda_j-\lambda_i}\boldsymbol{q}_j$$

$$= \Big(\sum_{j=m+1}^{n}\boldsymbol{q}_j\boldsymbol{q}_j^T\Big)\Big(\sum_{k=1}^N\epsilon_k\boldsymbol{x}_k^T\Big)\boldsymbol{q}_i\lambda_i^{-1} \tag{17}$$

$$= \big(I_n - QQ^T\big)\sum_{k=1}^N\epsilon_k\boldsymbol{x}_k^T\boldsymbol{q}_i\lambda_i^{-1}. \tag{18}$$

It is easy to check that

$$\begin{aligned}
d\boldsymbol{q}_i^T\boldsymbol{q}_i &= 0 \qquad 1\le i\le m\\
d\boldsymbol{q}_i^T\boldsymbol{q}_j + d\boldsymbol{q}_j^T\boldsymbol{q}_i &= 0 \qquad 1\le i\neq j\le m
\end{aligned}$$

Therefore, $Q^T\hat{Q}^\star \approx I_2$ up to second-order errors $O(\{\|\epsilon_k^2\|\})$. Therefore, the rotation matrix used to calibrate $\hat{Q}^\star$ and Q when defining $\mathcal{D}(\hat{Q}^\star, Q)$ is the identity matrix up to second-order errors $O(\{\|\epsilon_k^2\|\})$. Applying (18), we have

$$\begin{aligned}
&\frac{\partial\{\, D\,\}(\hat{Q}^\star, Q)}{\partial\epsilon_{ki}}\\
&= \big(I_n - QQ^T\big)\left(\boldsymbol{e}_k\boldsymbol{x}_k^T\boldsymbol{q}_1\lambda_1^{-1}, \cdots, \boldsymbol{e}_k\boldsymbol{x}_k^T\boldsymbol{q}_m\lambda_m^{-1}\right)\\
&= (I_n - QQ^T)\boldsymbol{e}_k\boldsymbol{x}_k^TQ\Lambda^{-1}.
\end{aligned}$$

The proof under the case where the eigenvalues of Q are repeating is similar, except that the summation in (15) shall discard (i,j) pairs where $\lambda_i = \lambda_j$. On the other hand, the uncertainties in eigenvectors when having repeating eigenvalues are addressed by the calibration rotation matrix in $\mathcal{D}(\hat{Q}^\star, Q)$.

□

References

[1] Iro Armeni, Ozan Sener, Amir R Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. 3d semantic parsing of large-scale indoor spaces. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1534–1543, 2016.

[2] Angela Dai, Angel X Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5828–5839, 2017.

[3] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4460–4470, 2019.

[4] Songyou Peng, Michael Niemeyer, Lars Mescheder, Marc Pollefeys, and Andreas Geiger. Convolutional occupancy networks. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16*, pages 523–540. Springer, 2020.

[5] Leif E Peterson. K-nearest neighbor. *Scholarpedia*, 4(2):1883, 2009.

[6] Charles R Qi, Or Litany, Kaiming He, and Leonidas J Guibas. Deep hough voting for 3d object detection in point clouds. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9277–9286, 2019.

[7] 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. *arXiv preprint arXiv:2206.04670*, 2022.

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

[9] Haiyang Wang, Lihe Ding, Shaocong Dong, Shaoshuai Shi, Aoxue Li, Jianan Li, Zhenguo Li, and Liwei Wang. Cagroup3d: Class-aware grouping for 3d object detection on point clouds. *arXiv preprint arXiv:2210.04264*, 2022.

[10] Peng-Shuai Wang, Yu-Qi Yang, Qian-Fang Zou, Zhirong Wu, Yang Liu, and Xin Tong. Unsupervised 3d learning for shape analysis via multiresolution instance discrimination. *ACM Trans. Graphic*, 2020.

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

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

[13] Yaoqing Yang, Chen Feng, Yiru Shen, and Dong Tian. Foldingnet: Point cloud auto-encoder via deep grid deformation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 206–215, 2018.

[14] Renrui Zhang, Ziyu Guo, Peng Gao, Rongyao Fang, Bin Zhao, Dong Wang, Yu Qiao, and Hongsheng Li. Point-m2ae: multi-scale masked autoencoders for hierarchical point cloud pre-training. *arXiv preprint arXiv:2205.14401*, 2022.