



# Unsupervised Point Cloud Representation Learning by Clustering and Neural Rendering

Guofeng Mei<sup>1,3</sup> · Cristiano Saltori<sup>2</sup> · Elisa Ricci<sup>2,3</sup> · Nicu Sebe<sup>2</sup> · Qiang Wu<sup>1</sup> · Jian Zhang<sup>1</sup>  · Fabio Poiesi<sup>3</sup>

Received: 3 April 2023 / Accepted: 2 February 2024  
© The Author(s) 2024

## Abstract

Data augmentation has contributed to the rapid advancement of unsupervised learning on 3D point clouds. However, we argue that data augmentation is not ideal, as it requires a careful application-dependent selection of the types of augmentations to be performed, thus potentially biasing the information learned by the network during self-training. Moreover, several unsupervised methods only focus on uni-modal information, thus potentially introducing challenges in the case of sparse and textureless point clouds. To address these issues, we propose an augmentation-free unsupervised approach for point clouds, named CluRender, to learn transferable point-level features by leveraging uni-modal information for soft clustering and cross-modal information for neural rendering. Soft clustering enables self-training through a pseudo-label prediction task, where the affiliation of points to their clusters is used as a proxy under the constraint that these pseudo-labels divide the point cloud into approximate equal partitions. This allows us to formulate a clustering loss to minimize the standard cross-entropy between pseudo and predicted labels. Neural rendering generates photorealistic renderings from various viewpoints to transfer photometric cues from 2D images to the features. The consistency between rendered and real images is then measured to form a fitting loss, combined with the cross-entropy loss to self-train networks. Experiments on downstream applications, including 3D object detection, semantic segmentation, classification, part segmentation, and few-shot learning, demonstrate the effectiveness of our framework in outperforming state-of-the-art techniques.

**Keywords** Unsupervised learning · Point cloud · Data-augmentation · Clustering · Neural rendering

## 1 Introduction

Point clouds are collections of spatially arranged points that approximate the surfaces of objects in three dimensions. Their effectiveness in understanding 3D data has resulted in progress in different fields, including robotic navigation (Biswas & Veloso, 2012; Zhou et al., 2022), autonomous driving (Li et al., 2020) and exploration (Wang et al., 2019; Wang & Bue, 2020), as well as augmented and virtual reality (Park et al., 2008; Mei et al., 2022). Nevertheless, point cloud data can face difficulties in intricate settings because of its sparse and textureless nature (Yan et al., 2022). Multi-modal data learning techniques, which integrate various data sources such as RGB images, depth maps, and sensor data

---

Communicated by Bodo Rosenhahn.

---

✉ Jian Zhang  
jian.zhang@uts.edu.au

Guofeng Mei  
guofeng.mei@student.uts.edu.au

Cristiano Saltori  
cristiano.saltori@unitn.it

Elisa Ricci  
eliricci@fbk.eu

Nicu Sebe  
niculae.sebe@unitn.it

Qiang Wu  
qiang.wu@uts.edu.au

Fabio Poiesi  
poiesi@fbk.eu

<sup>1</sup> Faculty of Engineering and IT, University of Technology Sydney, Ultimo, Sydney, NSW 2007, Australia

<sup>2</sup> Department of Information Engineering and Computer Science (DISI), University of Trento, Via Sommarive 9, 38123 Trento, Italy

<sup>3</sup> Fondazione Bruno Kessler, Via Sommarive 18, 38123 Trento, Italy

with point cloud data, can help mitigate these issues (Afham et al., 2022). Furthermore, the applications of point clouds in different fields emphasize the critical importance of learning discriminative and transferable features for 3D shape understanding (Lin et al., 2021; Poiesi & Boscaini, 2022). However, deep learning heavily depends on large-scale annotated data, which can be expensive, time-consuming, and may require domain expertise. Unsupervised representation learning has thus emerged as a promising method for learning features without human intervention (Eckart et al., 2021).

Unsupervised learning methods can be broadly classified as either generative or discriminative (Grill et al., 2020). Self-reconstruction or auto-encoding (Yang et al., 2018), generative adversarial networks (Sarmad et al., 2019), and auto-regressive (Sun et al., 2020; Mei, 2021) methods are examples of the former. These methods can use an encoder to map an input point cloud into a global latent representation (Rao et al., 2020; Shi et al., 2020), or a latent distribution in the variational case (Hassani & Haley, 2019; Han et al., 2019), and then use a decoder to attempt to reconstruct the input. To model high-level and structural properties of input point clouds, generative methods can be effective. However, because they are sensitive to Euclidean transformations, they usually assume that all 3D objects in a given category lie in the same reference system and have the same canonical pose (Sanghi, 2020). This assumption can lead to limitations, as recovering the original point cloud from rotation-invariant features may be unrealistic. Moreover, the handling of high-dimensional data or large datasets can present challenges, often necessitating significant computational resources.

Discriminative methods, as opposed to generative methods, learn to predict or discriminate augmented versions of the input. These methods can produce rich latent representations for subsequent tasks (Wang et al., 2020). For example, contrastive methods have produced promising results for unsupervised representation learning (Du et al., 2021; Rao et al., 2020; Sanghi, 2020) and can promote learning of rotation-invariant representations through data augmentation in the case of point cloud data (Poursaeed et al., 2020). These methods typically require negative mining to circumvent the issue of model collapse, and in particular, they rely on ad-hoc negative sampling criteria. They frequently require large batch sizes, memory banks, or customized strategies (Grill et al., 2020). Furthermore, any modification to the original geometry, such as cropping or view-based occlusions, can affect its semantics (Eckart et al., 2021). For instance, if a point cloud is randomly cropped, it could represent different objects, which could introduce conflicting information for learning. As a result, contrastive techniques require humans to carefully design artificial combinations of semantic-preserving data augmentations that preserve similar semantic information to learn informative representations. On the other hand, training on complete object instances can result in the learning

of global representations, which can lead to fewer discriminant representations because local geometric differences are ignored (Rao et al., 2020; Han et al., 2019; Xie et al., 2020).

Therefore, our first aim is to design a data augmentation-free learning strategy in order to avoid the limitation of generating chains of ad-hoc combinations of data augmentations. Our second aim is to achieve the optimization of local features, as opposed to global ones, in an unsupervised manner, allowing the network to learn 3D geometric information from point clouds and images. Although point cloud data can provide more comprehensive geometric information about the object than 2D images, there is key texture information that can be exploited from 2D images. Several studies (Afham et al., 2022; Jing et al., 2021; Wiles & Zisserman, 2019) have started leveraging multiple streams' synchronization to convey semantic information between different modalities, which has produced encouraging outcomes. These findings motivated us to introduce rendering to encode diverse visual cues from images without manual annotation.

In this paper, we propose to learn informative point-level representations of 3D point clouds in an unsupervised manner by using neural rendering and avoiding data augmentation. In particular, our framework learns cluster affiliation scores to softly group the 3D points of each point cloud into a given number of geometric partitions (soft clustering) and render images using learned features (neural rendering) (Aliev et al., 2020). We then learn these point-level feature representations by jointly minimizing the cross-entropy loss for soft clustering and fitting loss for neural rendering. The *cross-entropy loss* is the result of an EM (Expectation-Maximization)-like algorithm (Moon, 1996). We refer to this process as an EM-like algorithm, as it involves two steps, such as pseudo-label generation and network optimization. The E step employs an optimal transport (Peyré & Cuturi, 2019) based clustering algorithm to generate the point-level pseudo-labels, i.e. focusing on local geometric information. Our CluRender achieves this by softly labeling points based on their distance from the centroids in geometric space, with the constraint that labels partition data in equally-sized subsets (uniform distribution). Optimal transport is then used to compare probability distributions with each other and to produce optimal mappings by minimizing distances (Peyré & Cuturi, 2019). The M step employs the optimal mappings of E-step to form a point-to-cluster loss, which aids in optimizing the metric learning network. The *fitting loss* is formulated by comparing the rendered images with real images, enabling the networks to be pre-trained in 2D domains, contributing to superior 3D representations and appearance cues, resulting in more discerning features.

Our approach self-learns the partitioning network and softly assigns points to geometrically coherent overlapping clusters, overcoming the weakness of conventional GMM and K-means that involve computationally expensive

iterative procedures. Our approach is inspired by DeepCluster (Caron et al., 2018), SeLa (Asano et al., 2020) and SwAV (Caron et al., 2020) but it differs from them, as they implement clustering in the feature space at the instance level, they use data augmentation, and they may degrade the geometric information when used with 3D data. We show that pre-training on datasets using CluRender can improve the performance of a range of downstream tasks and outperforms the current state-of-the-art methods without any data augmentation.

This paper extends our earlier work (Mei et al., 2022) in several aspects. Specifically, we introduce a neural rendering module that allows 3D networks to perform cross-modal translation, converting a given 3D object point cloud into 2D rendered images from various camera perspectives. It employs a view-specific translation mechanism, which effectively captures extra geometric patterns and appearance cues, enabling the learning of more distinctive features. We also expanded the related work of unsupervised point cloud feature learning from multi-modal data. Besides, we significantly expand our experimental evaluation and analysis, considering additional recent methods, adding new backbones, and including new qualitative and quantitative results. To summarize, our contributions are:

- We propose a data augmentation-free unsupervised method, which does not rely on data augmentations, negative pair sampling, and large batches, to learn transferable point-level features on a 3D point cloud;
- We extend the pseudo-label prediction to an optimal transport problem, which can be efficiently solved by using an efficient variant of the Sinkhorn-Knopp (Cuturi, 2013) algorithm;
- We introduce neural rendering module to enable 3D networks to perform cross-modal translation from a given 3D object point cloud to its 2D rendered images from different camera views;
- We conduct thorough experiments, and CluRender achieves state-of-the-art performance without having to use data augmentation.

## 2 Related Work

In this section, we review research works related to unsupervised learning on point clouds, which can be classified into two categories: generative and discriminative methods. We also review research works related to unsupervised point cloud learning on multimodal data since our rendering module involves image and point cloud modalities.

### *Generative methods*

Generative methods learn features via self-reconstruction (Han et al., 2019). For instance, FoldingNet (Yang et al.,

2018) leverages a graph-based encoder and a folding-based decoder to deform a canonical 2D grid onto the surface of a point cloud. L2g (Liu et al., 2019) uses a local-to-global auto-encoder to learn the local and global structure of point clouds simultaneously. In Chen et al. (2019), a graph-based decoder with a learnable graph topology is used to push the codeword to preserve representative features. In Achlioptas et al. (2018), a combination of hierarchical Bayesian and generative models is trained to generate plausible point clouds. GraphTER (Gao et al., 2020) self-trains a feature encoder by reconstructing node-wise transformations from the representations of the original and transformed graphs. Recently, the recovery of missing parts from masked input has also been studied as a pre-text task in 3D point cloud learning, achieving remarkable performance (Yu et al., 2022; Pang et al., 2022; Liu et al., 2022). However, generative models are sensitive to transformations, weakening the learning of robust point cloud representations for different downstream tasks. Moreover, it is not always feasible to reconstruct back the shape from pose-invariant features (Sanghi, 2020).

### *Discriminative methods*

Discriminative methods are based on auxiliary hand-crafted prediction tasks to learn point cloud representations. Jigsaw3D (Sauder & Sievers, 2019) uses a 3D jigsaw puzzle approach as the self-supervised learning task. Recently, contrastive approaches (Rao et al., 2020; Sanghi, 2020; Chen and He, 2021; Xie et al., 2020; Huang et al., 2021), which are robust to transformation, achieved state-of-the-art performance. Info3D (Sanghi, 2020) maximizes the mutual information between the 3D shape and a geometric transformed version of the 3D shape. PointContrast (Xie et al., 2020) is the first to research a unified framework of the contrastive paradigm for 3D representation learning. ProposalContrast (Yin et al., 2022) enhances proposal representations by analyzing the geometric point relationships within each proposal. It achieves this by optimizing for inter-cluster and inter-proposal separation, resulting in better accommodation of 3D detection properties. MaskPoint (Liu et al., 2022) involves converting the point cloud into discrete occupancy values, where the value of 1 denotes the presence of a point in the cloud, and 0 indicates its absence. To accomplish this, the proxy task they use is binary classification, which distinguishes between the masked object points and the sampled noise points. FAC (Liu et al., 2023) forms advantageous point pairs belonging to the same foreground segment and possessing similar semantics, and subsequently, it efficiently acquires feature correlations within and across different point cloud views using adaptive learning. We argue that the success of contrastive methods relies on the correct design of negative mining strategies and on the correct choice of data augmentations that should not affect the semantics of the input. To mitigate these issues, we propose an unsupervised learning method, CluRender, formulated by implementing

clustering and rendering that provides point-level supervision to extract discriminative point-wise features. CluRender operates on the premise that an individual data point can be clustered into object parts, a concept also discussed in Leopart (Ziegler & Asano, 2022). However, a key difference lies in the approach to Leopart that segments images into various global and local views through cropping, CluRender eliminates the need for such data augmentation processes to derive informative representations.

#### Multimodal data-based methods

Recently, a few works have started to explore the pre-training pipeline with multi-modality data of 2D images and 3D point clouds. Pri3D (Hou et al., 2021) uses 3D point cloud and multi-view images to pre-train the 2D image networks, where multiple modalities are learned correspondences without the need for manual labeling. CrossPoint (Afham et al., 2022) aligns the 2D image features and 3D point cloud features through a contrastive learning pipeline. Li and Heizmann (2022) proposes a unified framework for exploring the invariances with different input data formats, including 2D images and 3D point clouds. P2p (Wang et al., 2022) involves converting point clouds into colored images, which are then input into a pre-trained image model with fixed weights to extract relevant features for subsequent tasks. Different from previous methods, most of which attempt to align 2D images and 3D point clouds in the feature space, our method proposes to connect 2D and 3D in the RGB-D image domain via differentiable rendering.

### 3 Method Overview

Our approach to unsupervised point cloud representation learning involves a joint soft clustering of the 3D points within each point cloud, dividing them into geometric partitions, and rendering images using learned features through differentiable rendering (Aliev et al., 2020). Figure 1 illustrates CluRender framework. Let  $\mathcal{P} = \{\mathbf{p}_i\}_{i=1}^N$  be a 3D point cloud and its associated images  $I = \{I_i\}_{i=1}^{N_r}$  obtained by rendering  $\mathcal{P}$  from  $N_r$  camera viewpoints  $\mathcal{V} = \{\mathcal{V}^i\}_{i=1}^{N_r}$ .  $N$  is the number of points of  $\mathcal{P}$ . Each point  $\mathbf{p}_i \in \mathbb{R}^3$  is represented by a 3D coordinate  $\mathbf{p}_i = \{x, y, z\}$ , and each image has shape  $W \times H \times 3$ . Our goal is to train a feature encoder  $f_\varphi$  with parameters  $\varphi$  (e.g. PointNet) that extracts informative point-level features  $\mathcal{F} = \{f_\varphi(\mathbf{p}_i)\}_{i=1}^N$  from  $\mathcal{P}$  without supervision. To achieve this, we employ a joint learning objective of soft clustering and neural rendering by minimizing  $\mathcal{L}_{tot}$ . In the former, we first use a segmentation head  $\phi_\alpha$ , which takes  $\mathcal{F} \in \mathbb{R}^{N \times d}$  as input and produces joint log probabilities. We then apply a softmax operator  $\sigma$  to the log probabilities to create a classification score matrix  $S$ , which is used to segment the point cloud into a specified number of geomet-

ric partitions. Since no labels are available to supervise  $S$ , we use a soft-labeling mechanism to generate pseudo-labels automatically. Soft-labeling first uses a prototype computation block to calculate  $J$  cluster prototypes  $\mathbf{C}$  to represent each partition. Next, we assign soft labels  $\gamma_{ij} \in \gamma$  to individual input points  $\mathbf{p}_i$  by leveraging these prototypes. This assignment of soft labels is framed as an optimal transport problem, which can be effectively solved using the Sinkhorn-Knopp algorithm. Each  $\gamma_{ij} \in [0, 1]$  is a soft-label score indicating the likelihood that point  $\mathbf{p}_i$  belongs to cluster  $j$ . The ultimate goal of the optimization process is to reduce the average cross-entropy loss, denoted as  $\mathcal{L}_c$ , between the predicted class probability  $S$  and the soft-label  $\gamma$ . In the latter, we first employ a prediction network to operate on the learned features, generating pseudo-color for each point, followed by rendering images from given  $N_r$  camera perspectives (views). Afterward, we compute the fitting loss  $\mathcal{L}_r$  by comparing the differences between the ground truth and rendered images. Finally, we train the network by minimizing the joint loss,  $\mathcal{L}_{tot}$ , which is defined as

$$\mathcal{L}_{tot} = \mathcal{L}_c + \mathcal{L}_r. \quad (1)$$

The implementation of CluRender is described in Algorithm 1 and in the next sections we will explain all its elements in detail. Specifically, in Sect. 4, we introduce our point cloud soft-clustering approach. In Sect. 5, we explain how point-wise features are utilized for point-based neural rendering.

### 4 Point Cloud Soft-Clustering

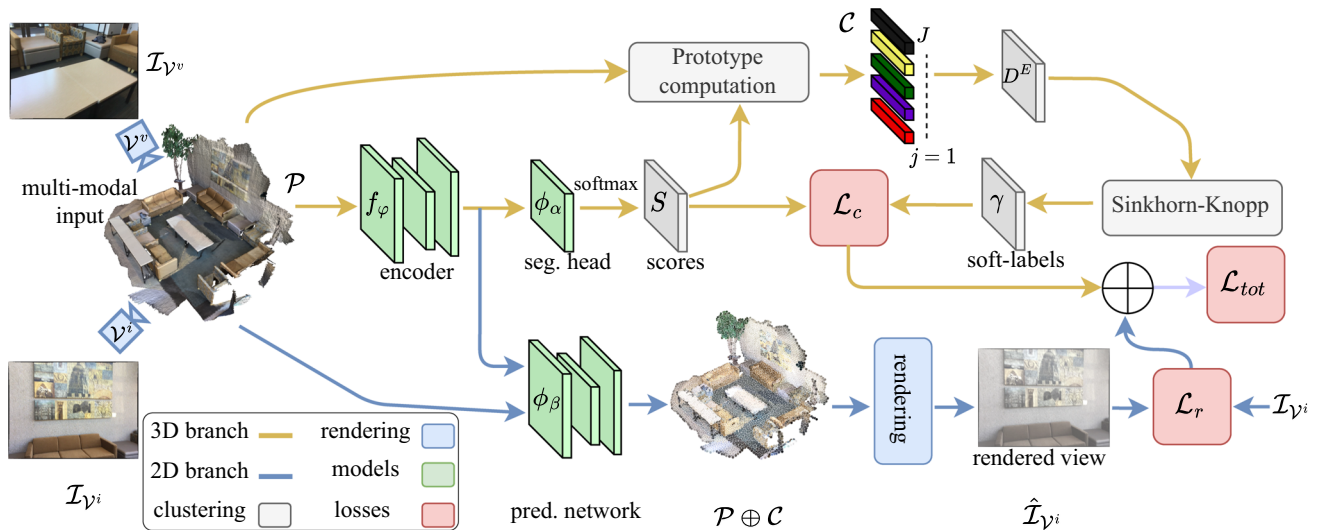
Our soft-clustering consists of three steps: prototype computation, soft-labeling, and optimization. These steps are described in the following sections.

#### 4.1 Prototype Computation

Our approach starts by learning a clustering task, which involves computing a prototype for each cluster, or partition, that serves as the representative feature for a group of points. For each feature  $\mathbf{f}_i \in \mathcal{F}$ ,  $\phi_\alpha$  produces a probability score  $s_{ij}$  indicating the likelihood that  $\mathbf{p}_i$  belongs to partition  $j$ . We use prototypes as the representatives for each partition in the geometric space. Specifically, we compute  $J$  prototypes as the weighted average of the 3D coordinates  $\mathcal{P}$  based on the scores  $S$ . Let  $\mathbf{C} = \{\mathbf{c}_j\}_{j=1}^J$  be the set of prototypes in the geometric space, respectively, which are defined as

$$\mathbf{c}_j = \frac{1}{\sum_{i=1}^N s_{ij}} \sum_{i=1}^N s_{ij} \mathbf{p}_i. \quad (2)$$





**Fig. 1** Architecture overview for our CluRender. Our approach consists of two parallel modules, namely, point cloud soft-clustering (represented by orange lines) and rendering (represented by light blue lines). CluRender requires a multi-modal input that includes a 3D point cloud  $\mathcal{P}$ , a 2D RGB image  $\mathcal{I}_{\mathcal{V}^i}$ , and its corresponding camera view  $\mathcal{V}^i$ . Our

approach jointly learns how to cluster the points in  $\mathcal{P}$  softly and generate a corresponding image  $\hat{\mathcal{I}}_{\mathcal{V}^i}$  based on the given camera view  $\mathcal{V}^i$ . The network is trained by minimizing the joint loss,  $\mathcal{L}_{tot}$ , which is the sum of the cross-entropy loss  $\mathcal{L}_c$  and the fitting loss  $\mathcal{L}_r$  (Color figure online)

**Algorithm 1** CluRender (Python pseudocode).

```

Input: A dataset  $\{(\mathcal{P}, \mathcal{I}, \mathcal{V})\}$  with 3D point cloud  $\mathcal{P} \in \mathbb{R}^{N \times 3}$ , image  $\mathcal{I} \in \mathbb{R}^{W \times H \times 3}$  and camera view  $\mathcal{V}$ ;  $K$  number of optimization steps.
Output: The backbone  $f_\phi$  pretrained by using our algorithm.
1: for  $i$  in range(0,  $K$ ) do
2:    $\mathcal{L}_{tot} = 0$ 
3:   for  $\mathcal{P} \in \{\mathcal{P}\}$  do
4:     # compute classification scores  $S$ 
5:      $S = \text{softmax}(\phi_\alpha(f_\phi(\mathcal{P})))$ 
6:     # compute prototypes
7:      $C = \left\{ \frac{1}{\sum_{i=1}^N s_{ij}} \sum_{i=1}^N s_{ij} p_i \right\}_{j=1}^N$ 
8:     # compute  $D$ 
9:      $D = \{\|p_i - c_j\|_2\}_{i,j}^{N,J}$ 
10:    # compute  $\gamma$ 
11:     $\gamma = \text{SINKHORN}(\text{stopgrad}(D), 1e-3, 20)$ 
12:     $\gamma = N \cdot \gamma$ 
13:    # rendering
14:     $\hat{\mathcal{I}}_r = \mathcal{R}(\phi_\beta(\mathcal{P}, f_\phi(\mathcal{P}), \mathcal{V}))$ 
15:    # compute loss
16:     $\mathcal{L}_{tot} += \mathcal{L}_c + \mathcal{L}_r$ 
17:  end for
18:  # update backbone and segmentation head
19:   $f_\phi, \phi_\alpha, \phi_\beta \leftarrow \text{optimize}\left(\frac{\mathcal{L}_{tot}}{N}\right)$ 
20: end for
21: return  $f_\phi$ 

```

**4.2 Soft-Labeling**

**4.2.1 Formulating Soft-Labeling as Assignment Problem**

Soft-labeling involves the assignment of each point to a certain prototype based on the distance estimated in Eq. (2). Our soft-label assignment enhances the ability of network to learn geometric and location information from point clouds. We base the assignment of soft-labels to the respective points on the prototypes  $C$ , and by following two assumptions:

- (i) *Cluster cohesion* If a point  $p_i$  is assigned to partition  $j$ , it should have the shortest distance to the prototype  $c_j$  compared to its distances with other prototypes in  $C$ .
- (ii) *Uniform distribution* Each point cloud is assumed to be segmented into equally-sized partitions of  $\lfloor \frac{N}{J} \rfloor$  elements, where  $\lfloor \cdot \rfloor$  indicates the greatest integer less than or equal to its argument.

Relying solely on cluster cohesion for soft-label assignment can lead to a degenerate solution, where all data points are assigned to a single label, consequently leading to the learning of a constant representation. To circumvent this issue, we thus introduce the constraint of a uniform distribution.

Based on assumption (i) we label points based on their distance from the centroids. Formally, if  $p_i$  belongs to cluster  $j$ , then  $\|p_i - c_j\|_2 \leq \|p_i - c_k\|_2$  and  $\gamma_{ij} \geq \gamma_{ik}, k \neq j, k = 1, \dots, J$ .  $\|\cdot\|_2$  is the  $L_2$  norm. This can be achieved by minimizing the objective function

$$\min_{\boldsymbol{y}} \frac{1}{N} \sum_{i=1}^N \sum_{j=1}^J \| \boldsymbol{p}_i - \boldsymbol{c}_j \|_2^2 \boldsymbol{y}_{ij}, \quad (3)$$

We define the matrix  $\boldsymbol{D}$  as follows for ease of use:

$$\boldsymbol{D} = \{ \| \boldsymbol{p}_i - \boldsymbol{c}_j \|_2^2 \}_{i,j}^{N,J}, \quad (4)$$

where the matrices are of size  $N \times J$ . With this definition, Eq. (3) can be rewritten as:

$$\min_{\boldsymbol{y}} \left\langle \frac{\boldsymbol{y}}{N}, \boldsymbol{D} \right\rangle, \quad (5)$$

where  $\langle \cdot, \cdot \rangle$  is the Frobenius matrix dot product.

Assumption (ii) is expressed under a constrained condition as  $\sum_{i=1}^N \boldsymbol{y}_{ij} = \frac{1}{J}$ , which can alleviate the issue of clustering all the data points into a single (arbitrary) label. Therefore, based on  $\sum_{i=1}^N \boldsymbol{y}_{ij} = \frac{1}{J}$  and the property of the posterior probability  $\sum_{j=1}^J \boldsymbol{y}_{ij} = 1$ ,  $\boldsymbol{y}$  satisfies the following constraints

$$\frac{1}{N} \boldsymbol{y}^\top \mathbf{1}_N = \frac{1}{J} \mathbf{1}_J, \quad \frac{1}{N} \boldsymbol{y} \mathbf{1}_J = \frac{1}{N} \mathbf{1}_N, \quad (6)$$

where  $\mathbf{1}_k$  ( $k = N, J$ ) denotes the vector of ones in dimension  $k$ . Therefore, the soft-label assignment ultimately leads to solving the subsequent optimization problem:

$$\begin{aligned} \min_{\boldsymbol{y}} \left\langle \frac{\boldsymbol{y}}{N}, \boldsymbol{D} \right\rangle, \\ \text{s.t. } \frac{1}{N} \boldsymbol{y}^\top \mathbf{1}_N = \frac{1}{J} \mathbf{1}_J, \quad \frac{1}{N} \boldsymbol{y} \mathbf{1}_J = \frac{1}{N} \mathbf{1}_N. \end{aligned} \quad (7)$$

This optimization problem can be addressed by formulating it as a linear program, which can be solved in polynomial time, as demonstrated by Cuturi (2013). The complexity of this problem presents a challenge for traditional algorithms, which may struggle to handle it due to the numerous data points and classes involved. To overcome this challenge, we convert the problem into an optimal transport problem (Peyré & Cuturi, 2019). This transformation allows us to use a more efficient version of the Sinkhorn-Knopp algorithm to solve the problem (Cuturi, 2013).

#### 4.2.2 Solving Assignment as Optimal Transport

The joint objective of assumptions (i) and (ii) can be framed as an optimal transport (OT) problem. In this context, we can view the assignment of data points to cluster centroids as a

transportation plan, assuming that both points and centroids adhere to uniform distributions. In this analogy, each data point represents a resource that requires transportation to a cluster centroid. The transportation cost, in this context, symbolizes the dissimilarity or distance between the data point and the centroid.

In particular, the discrete formulation of OT involves finding the optimal transportation plan between two discrete probability measures  $\boldsymbol{\mu}_X$  and  $\boldsymbol{\mu}_Y$ , which are defined on finite sets  $X$  and  $Y$  respectively. Formally, starting from Eq. (4), we can consider  $\boldsymbol{D} : X \times Y \rightarrow \mathbb{R}_+$  be the cost function representing the cost of transporting mass from a point  $x \in X$  to a point  $y \in Y$ . We aim to find a transportation plan  $\boldsymbol{\Gamma} \in \mathbb{R}_+^{X \times Y}$  that minimizes the total cost of transportation while satisfying the marginal constraints, such that

$$\min_{\boldsymbol{\Gamma}(X,Y)} \langle \boldsymbol{\Gamma}, \boldsymbol{D} \rangle, \quad (8)$$

$$\text{s.t. } \begin{cases} \sum_{y \in Y} \boldsymbol{\Gamma}(x, y) = \boldsymbol{\mu}_X(x), \\ \sum_{x \in X} \boldsymbol{\Gamma}(x, y) = \boldsymbol{\mu}_Y(y), \end{cases} \quad (9)$$

where  $\boldsymbol{\Gamma}(x, y)$  represents the amount of mass transported from  $x$  to  $y$ , and the constraints ensure that the total mass at each point in  $X$  and  $Y$  is preserved.

As for the assignment problem in Eq. (7), we set  $X = \mathcal{P}$ ,  $Y = \mathcal{C}$ ,  $\boldsymbol{\Gamma} = \frac{\boldsymbol{y}}{N}$ ,  $\boldsymbol{\mu}_X = \frac{1}{N} \mathbf{1}_N$ , and  $\boldsymbol{\mu}_Y = \frac{1}{J} \mathbf{1}_J$ . Then, the soft-label assignment leads to solving the optimal transport problem:

$$\begin{aligned} \min_{\boldsymbol{\Gamma}(\mathcal{P},\mathcal{C})} \langle \boldsymbol{\Gamma}, \boldsymbol{D} \rangle, \\ \text{s.t. } \boldsymbol{\Gamma}^\top \mathbf{1}_N = \frac{1}{J} \mathbf{1}_J, \quad \boldsymbol{\Gamma} \mathbf{1}_J = \frac{1}{N} \mathbf{1}_N. \end{aligned} \quad (10)$$

Sinkhorn-Knopp (Cuturi, 2013) is then employed to solve Eq. (10). This requires the following regularization term:

$$\begin{aligned} \min_{\boldsymbol{\Gamma}} \langle \boldsymbol{\Gamma}, \boldsymbol{D} \rangle - \epsilon H(\boldsymbol{\Gamma}), \\ \text{s.t. } \boldsymbol{\Gamma}^\top \mathbf{1}_N = \frac{1}{J} \mathbf{1}_J, \quad \boldsymbol{\Gamma} \mathbf{1}_J = \frac{1}{N} \mathbf{1}_N, \end{aligned} \quad (11)$$

where  $H(\boldsymbol{\Gamma}) = \langle \boldsymbol{\Gamma}, \log \boldsymbol{\Gamma} - 1 \rangle$  denotes the entropy of  $\boldsymbol{\Gamma}$  and  $\epsilon > 0$  is a regularization parameter. When  $\epsilon$  is very small, optimizing Eq. (11) is equivalent to optimizing Eq. (10). However, even for moderate values of  $\epsilon$ , the objective function tends to approximate the same solution. According to Cuturi (2013), larger values of  $\epsilon$  lead to faster convergence. Our primary concern is the final results of clustering and representation learning rather than an exact solution to the transport problem. Therefore, using a fixed value of

$\epsilon = 1e - 3$  as in Mensch and Peyré (2020) is appropriate to achieve the balance between speed and accuracy. As mentioned in Cuturi (2013), the solution to Eq. (11) can be expressed using a normalized exponential matrix:

$$\Gamma = \text{diag}(\boldsymbol{\mu}) \exp(\mathbf{D}/\epsilon) \text{diag}(\mathbf{v}), \tag{12}$$

where  $\boldsymbol{\mu} = (\mu_1, \mu_2, \dots, \mu_N)$  and  $\mathbf{v} = (v_1, v_2, \dots, v_J)$  are re-normalization vectors in  $\mathbb{R}^N$  and  $\mathbb{R}^J$ . The vectors  $\boldsymbol{\mu}$  and  $\mathbf{v}$  can be obtained by iterating the updates via  $\mu_i = [\exp(\mathbf{D}/\epsilon) \mathbf{v}]_i^{-1}$  and  $v_j = [\exp(\mathbf{D}/\epsilon)^\top \boldsymbol{\mu}]_j^{-1}$  with initial values  $\boldsymbol{\mu} = \frac{1}{N} \mathbf{1}_N$  and  $\mathbf{v} = \frac{1}{J} \mathbf{1}_J$ , respectively. The choice of distribution for initializing  $\boldsymbol{\mu}$  and  $\mathbf{v}$  can be arbitrary, but using the constraint values as initial values can speed up the convergence process. The notation  $[\cdot]_j^{-1}$  denotes the inverse of the  $j^{\text{th}}$  element of its argument. In our experiments, we have found that using 20 iterations works well in practice. Once we have solved Eq. (12), we can obtain the soft-label matrix as  $\boldsymbol{\gamma} = N \cdot \Gamma$ . For the Sinkhorn-Knopp algorithm, we provide a detailed pseudo-code in Algorithm 2.

### 4.3 Optimization

The optimization procedure utilizes an EM-like framework, in which the Expectation step E optimizes soft labels and prototypes, and the Maximization step M optimizes the parameters for representation learning. The following are the specifics of each step:

- *Step E* Given the current encoder and segmentation layer, we compute prototypes  $\mathbf{C}$  following Eq. (2), and obtain soft-labels  $\boldsymbol{\gamma}$  through  $\boldsymbol{\gamma} = N \cdot \Gamma$ .
- *Step M* Given the current soft-labels  $\boldsymbol{\gamma}$  from step E, we optimize the encoder  $f_\varphi$  and segmentation layer  $\phi_\alpha$  parameters.

During E, we solve the OT problem with the Sinkhorn-Knopp algorithm. During M, we minimize the segmentation loss based on the resulting soft labels, such as

$$\mathcal{L}_{soft}(\boldsymbol{\gamma}, \mathbf{S}) = -\frac{1}{N} \sum_{i=1}^N \sum_{j=1}^J \boldsymbol{\gamma}_{ij} \log s_{ij}, \tag{13}$$

which corresponds to the minimization of the standard cross-entropy loss between soft-labels  $\boldsymbol{\gamma}$  and predictions  $\mathbf{S}$ .

Note that, this EM-like algorithm incorporates both feature and coordinate information to generate pseudo-labels, enabling the network to learn more informative features. Alternatively, one could cluster points based on their coordi-

### Algorithm 2 Sinkhorn-Knopp algorithm (Python pseudocode).

---

**Input:**  $D$  distance matrix,  $\epsilon=1e-3$  and  $niters$  iterations.

```

1: function SINKHORN( $D, \epsilon, niters$ )
2:    $\Gamma = \exp(D/\epsilon)$ 
3:    $\Gamma / = \text{sum}(\Gamma)$ 
4:    $N, J = \Gamma.\text{shape}$ 
5:    $\mathbf{u}, \boldsymbol{\mu}, \mathbf{v} = \text{zeros}(N), \text{ones}(N)/N, \text{ones}(J)/J$ 
6:   for  $\_$  in range(0,  $niters$ ) do
7:      $\mathbf{u} = \text{sum}(\Gamma, \text{dim}=1)$ 
8:      $\Gamma * = (\boldsymbol{\mu}/\mathbf{u}).\text{unsqueeze}(1)$ 
9:      $\Gamma * = (\mathbf{v}/\text{sum}(\Gamma, \text{dim}=0)).\text{unsqueeze}(0)$ 
10:  end for
11: return  $\Gamma$ 
12: end function

```

---

nates once at the beginning for the generation of pseudo-labels, and then use the same pseudo-labels throughout training. Although this approach is more efficient, as it does not require the execution of clustering at each iteration, it suffers from two main limitations: (i) It may lead to the formation of less informative clusters because it only considers the distance between points in the coordinate space. As a result, it may cluster the same object into different parts. (ii) It may produce inconsistent pseudo-labels since the same objects from different point clouds may be clustered with different cluster indexes. As a result, it may produce different pseudo-labels for the same group of points. Therefore, our EM-like algorithm is advantageous as it can mitigate these two limitations by performing clustering in the feature space and then mapping the results into the coordinate space. We perform this process at each iteration because the quality of features improves over time. Moreover, our EM-like algorithm employs a network to learn clustering, which helps to mitigate the problem of inconsistent pseudo-labeling for each point cloud.

The objective function of Eq. (13) does not prevent the potential collapse of prototypes into an identical prototype. In the event of such a prototype collapse, the equal partition constraint is unable to prevent the encoder  $f_\varphi$  from predicting identical features for all points. To enhance the separation of prototypes, we minimize the orthogonal regularization loss  $\mathcal{L}_{orth}(\mathbf{C}) = \|\mathbf{C}_*^\top \mathbf{C}_* - \mathbf{E}\|_{Fr}$ , where  $\mathbf{C}_*$  represents the normalized version of  $\mathbf{C}$  obtained by performing  $\mathbf{C}_* = [\frac{c_1}{\|c_1\|_2}, \frac{c_2}{\|c_2\|_2}, \dots, \frac{c_J}{\|c_J\|_2}]$ , and  $\|\cdot\|_{Fr}$  signifies the Frobenius norm.  $\mathbf{E}$  denotes an identity matrix.

We define the final clustering loss for the M step as

$$\mathcal{L}_c = \mathcal{L}_{soft} + \omega \mathcal{L}_{orth}, \tag{14}$$

where  $\omega = 0.01$  is a weighting parameter. We set the value of  $\omega$  empirically and found that  $\omega \leq 0.01$  can slightly improve the performance. The minimization of this loss leads to the

maximization of the expected number of points correctly classified, associating the correct neighboring prototypes. This facilitates the encoder to learn more local geometric information.

## 5 Differentiable Renderer

The rendering module makes full use of both 3D point cloud and 2D image modalities, and thus facilitates the network in extracting both photometric and structural information from the appearance of 2D images. The basis for our neural rendering-based unsupervised learning draws inspiration from the works of Aliev et al. (2020), Insafutdinov and Dosovitskiy (2018) and Wiles et al. (2020). We aim to generate a set of  $N_r$  images by rendering a point cloud from a given collection of views, denoted as  $\mathcal{V} = \{\mathcal{V}^v\}_{v=1}^{N_r}$ , where each view is defined by transformation  $T_v$ . It is assumed that each resulting image will be a pixel grid of dimensions  $W \times H$ , and  $N_r$  represents the number of input views. Initially, a color network  $\phi_\beta$  is applied to  $\mathcal{F}$  to generate RGB color vectors  $\mathcal{C}_i$  for point  $\mathbf{p}_i$ . The 3D coordinates of the raw point cloud are then transformed into the standard coordinate frame using the projective transformation that corresponds to the camera pose. Next, we utilize the point-based differentiable renderer  $\mathcal{R}$ , which projects the 3D point cloud data into 2D view images based on the camera pose settings. Finally, a Wasserstein (Cuturi, 2013) distance-based fitting loss is applied to measure the consistency between the rendered pixels and the corresponding pixels in the ground-truth image. Next, a detailed illustration will be provided for  $\phi_\beta$  along with an explanation of renderer  $\mathcal{R}$  and fitting loss  $\mathcal{L}_r$ .

### 5.1 Prediction Network $\phi_\beta$

Our approach involves utilizing a Transformer-based UNet (Ronneberger et al., 2015) architecture, accepting 3D point coordinates along with features from the encoder as inputs. The model, denoted as  $\phi_\beta$ , employs three downsampling and upsampling layers to generate a prediction output that coincides with the spatial resolution of the input, thereby assigning a color to each point in a point cloud. Each stage of the downsampling process uses a distinct rate and employs the farthest point sampling technique in the coordinate space. Specifically, the rates are [1, 4, 4], leading to point set cardinalities of [ $N$ ,  $N/4$ ,  $N/16$ ]. Each stage of the design incorporates a layered format similar to that of Point Transformer (Zhao et al., 2021), with a total of six layers having output dimensions of [512, 256, 128, 128, 64, 32]. The resulting output is subsequently fed through an MLP with an output dimension of 3, followed by a sigmoid layer, and finally, a renormalization step. The purpose of the renor-

malization step is to ensure that the predicted colors remain within the specified color minimum and maximum values.

### 5.2 Renderer $\mathcal{R}$

Traditional rendering techniques experience discontinuities in splatting and  $z$ -buffering due to their use of a  $z$ -buffer to consider only the closest source points when contributing to a target image pixel. This approach is non-differentiable and leads to sparse gradients in the  $xy$ -plane of the rendered view when small neighborhoods are considered. Therefore, it is unsuitable for our framework. To fix this gap, we adopt a splatting technique to project and splat 3D points onto a disk of varying influence, with the degree of influence determined by a hyperparameter  $\nu > 0$ . The Euclidean distance from the center of the region is proportional to the impact of a point on a given pixel  $\mathbf{l}_{xy}^v$  located at  $(x, y)$  for view  $\mathcal{V}^v$ . Each 3D point  $\mathbf{p}_i$  is projected and splatted onto a region with a center  $\mathbf{p}_{i_c}$  and a radius  $r$ , which can be expressed as:

$$\mathcal{G}(\mathbf{p}_i, \mathbf{l}_{xy}^v) = \begin{cases} 0, & \|\mathbf{p}_{i_c} - \mathbf{l}_{xy}^v\|_2^2 > r, \\ \exp\left(-\frac{\|\mathbf{p}_{i_c} - \mathbf{l}_{xy}^v\|_2^2}{2\nu^2}\right), & \text{otherwise,} \end{cases}$$

where  $r$  and  $\nu$  determine the range and decay of the influence of a 3D point to a target image pixel. The projected points are then gathered in a  $z$ -buffer, sorted based on their distance from the new camera view, and finally, only the  $K$  closest points are retained for each pixel in the new view. The sorted points are combined using the alpha over-compositing algorithm (Porter & Duff, 1984), which can be expressed as:

$$\begin{aligned} \rho_{i_{xy}} &= \mathcal{G}(\mathbf{p}_i, \mathbf{l}_{xy}^v), \\ \bar{\mathcal{C}}_{xy}^v &= \sum_{i=1}^K \rho_{i_{xy}}^\eta \mathcal{C}_i \prod_{j=1}^{i-1} (1 - \rho_{j_{xy}}^\eta), \end{aligned} \quad (15)$$

where  $\eta$  is a hyperparameter that controls the blending and  $\bar{\mathcal{C}}_{xy}^v$  is the projected feature map in the new view. The resulting feature map is then used to construct a new RGB image  $\hat{\mathcal{I}}_v$  with three color channels. The whole process can be described as

$$\hat{\mathcal{I}}_v = \mathcal{R}(\phi_\beta(\mathcal{P}, \mathcal{F}), \mathcal{V}^v). \quad (16)$$

### 5.3 Fitting Loss $\mathcal{L}_r$

After rendering the point clouds into multi-view images, we utilize the Wasserstein distance (Peyré & Cuturi, 2019) to calculate a loss  $\mathcal{L}_{r_v}$  that evaluates the consistency between



the rendered images' pixels and the corresponding pixels in the ground-truth images. The total rendering (fitting) loss  $\mathcal{L}_r$  is then obtained by summing up all individual view losses  $\mathcal{L}_{r_v}$ , where  $v$  ranges from 1 to  $N_r$ . Then total loss  $\mathcal{L}_r$  for rendering is the sum of all all  $\mathcal{L}_{r_v}$  as

$$\mathcal{L}_r = \sum_{v=1}^{N_r} \mathcal{L}_{r_v}. \quad (17)$$

The Wasserstein loss between rendered image  $\hat{I}_v$  and ground-truth image  $I_v$  can be described as follows.

$$\begin{aligned} \mathcal{L}_{r_v}(\hat{I}_v, I_v) &= \sum_{i=1}^{WH} \sum_{j=1}^{WH} \mathcal{D}(\mathbf{x}_i, \mathbf{y}_j) \tau_{ij}, \\ \text{s.t., } \sum_{i=1}^{WH} \tau_{ij} &= \frac{1}{WH}, \sum_{j=1}^{WH} \tau_{ij} = \frac{1}{WH}. \end{aligned} \quad (18)$$

Where  $\mathbf{x}_i$  is the location of pixel  $i$  in  $\hat{I}_v$  and  $\mathbf{y}_j$  is the location of pixel  $j$  in  $I_v$ , respectively. The objective is to find a transportation matrix  $\tau$  that minimizes the overall transportation cost  $L_r$ . where  $\tau_{ij}$  and  $\mathcal{D}(\mathbf{x}_i, \mathbf{y}_j)$  denote the transportation amount and unit transportation cost from pixel location  $\mathbf{x}_i$  of  $\hat{I}_v$  to location  $\mathbf{y}_j$  of  $I_v$ . The unit transportation cost is set as the sum of a color distance and a positional distance:

$$\mathcal{D}(i, j) = \lambda (\mathbf{x}_i - \mathbf{y}_j)^2 + (1 - \lambda) (\hat{\mathbf{c}}_i - \mathbf{c}_j)^2,$$

where  $\hat{\mathbf{c}}_i$  and  $\mathbf{c}_j$  indicate the colors of the rendered image  $\hat{I}$  at pixel  $\mathbf{x}_i$  and the ground truth image  $I$  at pixel  $\mathbf{y}_j$ , respectively. To balance the relative importance of color and position, we use a weighting parameter  $\lambda$ . In our experiments, we set  $\lambda = 0.5$ , which is consistent with the approach taken in Xing et al. (2022).

## 6 Experiments

In this section, we present the implementation details, the setup of pre-training and the downstream fine-tuning.

### 6.1 Pre-training Setup

#### 6.1.1 Implementation Details

PyTorch was used to implement CluRender, and experiments were conducted on two Tesla V100-PCI-E-32G GPUs. During pre-training, we set the values of  $J$ ,  $N_r$ , and  $\epsilon$  to 64, 8, and

0.001, respectively, following (Wiles et al., 2020). Additionally, we used  $\eta = 1$ ,  $r = 4$  pixels,  $K = 128$ ,  $W = H = 256$ , and conducted pre-training for 250 epochs using the AdamW (Loshchilov & Hutter, 2018) optimizer. The batch size was set to 32, and the initial learning rate was set to 0.001, which was then decayed by a factor of 0.7 every 20 epochs. No data augmentations were used during the pre-training stage. The segmentation head  $\phi_\alpha$  was composed of three fully connected layers, each consisting of a linear layer followed by layer normalization and a rectified linear unit, except for the final layer. The output dimensions of the hidden layer and the final linear layer were set to half the dimensions of the encoder output and the number of clusters, respectively.

#### 6.1.2 Datasets

We explore pre-training strategies on complex scenes with multiple objects (ScanNet (Dai et al., 2017)) and single objects (ShapeNet (Chang et al., 2015)) to evaluate the effectiveness of CluRender.

*ScanNet* (Dai et al., 2017) is a dataset of real-world indoor scenes with multiple objects and contains 2.5M views in 1,513 indoor scans for 707 distinct spaces. We use ScanNet as the pre-training dataset for 3D object detection and semantic segmentation. With a frame rate of 25, we gather the paired RGB image data and 3D point cloud data from the scanned RGB-D videos, thus obtaining 100K high-quality frames in total. The data are split into the training and testing sets as done in VoteNet (Qi et al., 2019). We sample 4,096 points from each input point cloud using a farthest point sampling algorithm.

*ShapeNet* (Chang et al., 2015) is a collection of single-object CAD models and contains more than 50K synthetic objects from 55 object categories. We use it as our pre-training dataset for semantic segmentation, object classification, part segmentation, and few-shot classification. We utilize the same rendering approach as DISN (Xu et al., 2019) to produce RGB images, which involves a dataset of 24 images and one point cloud for each single-object CAD model.

#### 6.1.3 Backbones

To ensure a fair comparison, we select three different types of encoders as backbones: point-based PointNet (Qi et al., 2017), graph-based DGCNN (Wang et al., 2019), and Transformer (Vaswani et al., 2017). Notably, we employed the Transformer encoder provided by Point-M2AE (Zhang et al., 2022), which is an end-to-end hierarchical encoder that downsamples the input point clouds three times. However, since our method relies on point-level features, following Point-M2AE, we use a propagation module (Qi et al., 2017a) to upsample the point clouds that were previously downsampled by the Transformer encoder, thus restoring the original

number of points in the raw point cloud. We remove the masking and replace the final prediction head of PointM2AE with two heads,  $\phi_\alpha$  and  $\phi_\beta$ . Lastly, we apply the clustering and rendering to train the Transformer. We use the pre-trained weights of the Transformer encoder as initialization for downstream tasks, including object detection, object segmentation, object classification, part segmentation, and few-shot learning.

## 6.2 Evaluation on Downstream Tasks

For fine-tuning downstream tasks, we remove the clustering and rendering structures from pre-training and add various task-specific heads to the backbones.

### 6.2.1 3D Object Detection

Object detection on a scene-level downstream task is challenging for 3D models. We first assess the effectiveness of our pre-training approach on 3D point cloud detection, utilizing a Transformer as the feature extractor and selecting 3DETR (Misra et al., 2021) and 3DETR-m (Misra et al., 2021) as our benchmark models. We pre-train the Transformer encoder on ScanNet and evaluate the downstream task 3D object detection on ScanNetV2 (Dai et al., 2017). To compare the performance of CluRender with other relevant methods, we include Point-Bert (Yu et al., 2022) and MaskPoint (Liu et al., 2022), both of which also utilize Transformer as a backbone pre-trained on ScanNet. The evaluation metrics employed are average precision with 3D detection IoU thresholds of 0.25 (AP<sub>25</sub>) and 0.5 (AP<sub>50</sub>), maintaining consistency with the metrics used in Misra et al. (2021).

3DETR (Misra et al., 2021) is an end-to-end transformer-based pipeline for 3D object detection. It combines the conventional transformer structure with non-parametric queries and Fourier positional embeddings to detect the bounding boxes from 3D point clouds. The backbone structure for 3DETR corresponds to the pre-trained model, which contains 12-layer transformer blocks with a transformer dimension of 384.

ScanNetV2 (Dai et al., 2017) consists of 1,201 training scenes, 312 validation scenes and 100 hidden test scenes. Axis-aligned bounding box labels are provided for 18 object categories. We follow the settings of 3DETR (Misra et al., 2021) for point sampling and data augmentation.

Results Table 1 presents the 3D detection results, with CluRender consistently outperforming its competitors. Notably, our method surpasses 3DETR without pre-training by a substantial margin, yielding a 1.6% enhancement in AP<sub>25</sub> and a 4.8% improvement in AP<sub>50</sub>. Moreover, our detection encoder, when initialized with CluRender, exceeds the performance of the state-of-the-art pre-training method, MaskPoint. Specifically, in comparison to MaskPoint, our

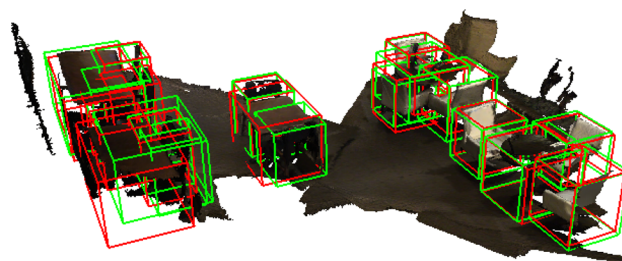


Fig. 2 Qualitative comparison results of 3D object detection in ScanNetV2. The color red represents the ground truth, and the color green represents the predictions (Color figure online)

method achieves increments of 0.9% and 1.7% in AP<sub>25</sub> and AP<sub>50</sub>, respectively, further substantiating the effectiveness of our pre-training strategy. This can be attributed to capacity of CluRender to enable the backbone to assimilate both photometric and geometric information from the scene, thereby enhancing accuracy. Furthermore, CluRender also surpasses our previous pre-training approach, SoftClu, illustrating that neural rendering is capable of capturing finer photometric details in the learned features of point clouds, thereby affirming the efficacy of the newly introduced rendering module. Our methodology also realizes superior accuracy compared to pre-training methods based on 3DETR-m (Misra et al., 2021). Figure 2 provides a qualitative comparison of the 3D bounding box prediction outcomes on the ScanNetV2 validation set.

### 6.2.2 3D Semantic Segmentation

Semantic segmentation on large-scale 3D scenes is challenging, showing the understanding of contextual semantics and local geometric relationships. To assess the effectiveness of pre-trained features using CluRender for semantic segmentation, we conducted experiments on the S3DIS (Armeni et al., 2016) benchmark dataset. To ensure a fair comparison, we evaluated our method alongside Jigsaw3D (Sauder & Sievers, 2019) and OcCo (Wang et al., 2021), which also use the same three backbones. To measure the quality of segmentation, we reported Intersection over Union (mIoU) averaged over all classes.

S3DIS comprises 3D scans collected with the Matterport scanner in six indoor areas, featuring 271 rooms and 13 semantic classes. Following the pre-processing, post-processing and training settings as in (Wang et al., 2020), we split each room into  $1m \times 1m$  blocks and use 4,096 points as the model input. Following common practice, we finetune the pre-trained model in areas 1,2,3,4 and 6 and test them in area 5. We use an SGD optimizer with a momentum of 0.9 and a weight decay of  $1e-4$ . The learning rate starts from 0.1 and then decays using cosine annealing with the minimum value  $1e-3$ . We train the models for 250 epochs with batch

**Table 1** 3D object detection results on ScanNetV2 validation set

| Method                                    | Pre-training | AP25 ↑      | AP50 ↑      |
|-------------------------------------------|--------------|-------------|-------------|
| <i>VoteNet-based pre-training methods</i> |              |             |             |
| VoteNet (Qi et al., 2019)                 | ×            | 58.6        | 33.5        |
| PointContrast (Xie et al., 2020)          | ✓            | 59.2        | 38.0        |
| DepthContrast (Zhang & Misra, 2021)       | ✓            | 61.3        | –           |
| RandomRooms (Rao et al., 2021)            | ✓            | 61.3        | 36.2        |
| STRL (Huang et al., 2021)                 | ✓            | 59.5        | 38.4        |
| PC-FractalDB (Yamada & Ogata, 2022)       | ✓            | 61.9        | 38.3        |
| <i>3DETR-based pre-training methods</i>   |              |             |             |
| 3DETR (Misra et al., 2021)                | ×            | 62.7        | 37.5        |
| Point-Bert (Yu et al., 2022)              | ✓            | 61.0        | 38.3        |
| MaskPoint (Liu et al., 2022)              | ✓            | 63.4        | 40.6        |
| SoftClu (Mei et al., 2022)                | ✓            | 63.9        | 42.0        |
| CluRender                                 | ✓            | <b>64.3</b> | <b>42.3</b> |
| <i>3DETR-m based pre-training methods</i> |              |             |             |
| 3DETR-m (Misra et al., 2021)              | ×            | 65.0        | 47.0        |
| Point-M2AE (Zhang et al., 2022)           | ✓            | 66.3        | 48.3        |
| SoftClu (Mei et al., 2022)                | ✓            | 66.2        | 48.3        |
| CluRender                                 | ✓            | <b>66.9</b> | <b>50.1</b> |

Our CluRender outperforms all previous state-of-the-art pretraining methods. Bold values indicate best performance

size 16. We use the same post-processing during testing as (Wang et al., 2021) for a fair comparison.

**Results** Table 2 reports the segmentation results of CluRender and that of the other baselines on S3DIS (Armeni et al., 2016). CluRender outperforms all the other approaches with PointNet, DGCNN, and Transformer encoders. With the PointNet encoder, CluRender achieves 56.3% mIoU, outperforming both the state-of-the-art OcCo (55.3% mIoU) and Jigsaw3D (52.6% mIoU). With the DGCNN encoder, CluRender achieves 60.4% mIoU, outperforming CrossPoint (Afham et al., 2022) (58.4% mIoU), OcCo (58.0% mIoU) and Jigsaw3D (55.6% mIoU). As for the transformer-based backbone, our pre-training approach achieves a mIoU of 61.9% on ShapeNet and 62.6% on ScanNet. This outperforms both the state-of-the-art ACT (Pang et al., 2022) (61.2%) and our previous SoftClu (Mei et al., 2022) method (61.6%). Note that, the rendering module is the primary distinction between SoftClu and CluRender. The superior performance of CluRender compared to SoftClu across all backbones underscores the effectiveness of our rendering module in promoting the learning of meaningful representations. Figure 3 presents qualitative examples of semantic segmentation on S3DIS using DGCNN as the encoder. The black dashed box indicates the areas where segmentation failed. From this, it can be observed that the failure regions are ambiguous and lack distinct geometric structures. In com-

parison to SoftClu, the introduction of the rendering module significantly improves the segmentation performance. This further suggests that the rendering module aids in facilitating the network to learn more discriminative features that are helpful for accurate segmentation. Although these examples demonstrate our approach's limitations in some areas where DGCNN struggles to extract distinct features. This challenge arises because these regions share similar surfaces and geometric characteristics, which makes it difficult for DGCNN to differentiate between them. Despite this limitation, the overall results of the segmentation task demonstrate the potential effectiveness of our approach.

### 6.2.3 Classification

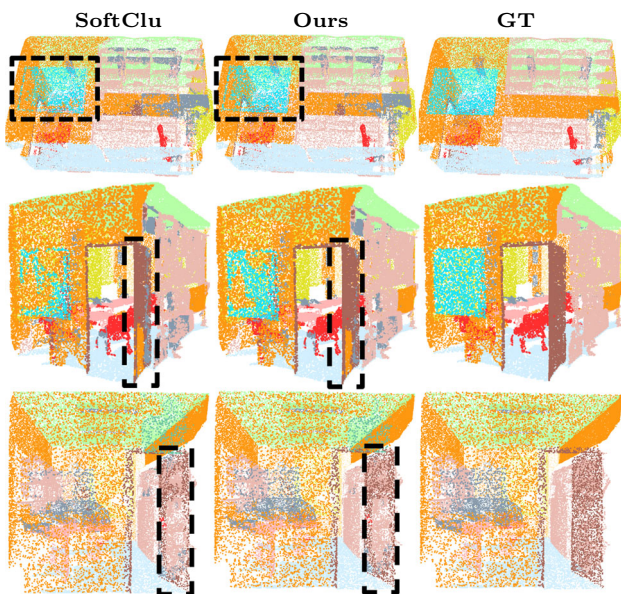
We use linear Support Vector Machine (SVM) classification on ModelNet40 (Sharma et al., 2016) benchmark to evaluate the quality of their pre-trained versions on ShapeNet.

*ModelNet40* (Sharma et al., 2016) is composed of 12,331 meshed models from 40 object categories split into 9,843 training meshes and 2,468 testing meshes, where the points are sampled from CAD models. We randomly sample 1,024 points for each shape as in (Sauder & Sievers, 2019). The pre-trained backbones are frozen and used to extract point cloud features. Then, we train SVM on the features of the train set and evaluate it on the test set.

**Table 2** 3D semantic segmentation mIoU results on the S3DIS dataset using different pre-trained backbones

| Method                            | Input     | Pre. data       | mIoU $\uparrow$ |
|-----------------------------------|-----------|-----------------|-----------------|
| <i>PointNet backbone</i>          |           |                 |                 |
| From Scratch                      | $xyz+rgb$ | –               | 47.0            |
| Jigsaw3D (Sauder & Sievers, 2019) | $xyz+rgb$ | <i>ShapeNet</i> | 52.6            |
| OcCo (Wang et al., 2020)          | $xyz+rgb$ | <i>ShapeNet</i> | 54.9            |
| SoftClu (Mei et al., 2022)        | $xyz+rgb$ | <i>ShapeNet</i> | 55.6            |
| CluRender                         | $xyz+rgb$ | <i>ShapeNet</i> | <b>56.3</b>     |
| <i>DGCNN backbone</i>             |           |                 |                 |
| From Scratch                      | $xyz+rgb$ | –               | 54.9            |
| Jigsaw3D (Sauder & Sievers, 2019) | $xyz+rgb$ | <i>ShapeNet</i> | 55.6            |
| OcCo (Wang et al., 2020)          | $xyz+rgb$ | <i>ShapeNet</i> | 58.0            |
| SoftClu (Mei et al., 2022)        | $xyz+rgb$ | <i>ShapeNet</i> | 59.7            |
| CluRender                         | $xyz+rgb$ | <i>ShapeNet</i> | <b>60.4</b>     |
| <i>Transformer backbone</i>       |           |                 |                 |
| From Scratch (Yu et al., 2022)    | $xyz$     | –               | 60.0            |
| Point-Bert (Yu et al., 2022)      | $xyz$     | <i>ShapeNet</i> | 60.8            |
| Point-MAE (Pang et al., 2022)     | $xyz$     | <i>ShapeNet</i> | 61.0            |
| ACT (Dong et al., 2022)           | $xyz$     | <i>Image</i>    | 61.2            |
| SoftClu (Mei et al., 2022)        | $xyz$     | <i>ShapeNet</i> | 61.6            |
| CluRender                         | $xyz$     | <i>ShapeNet</i> | 61.9            |
| CluRender                         | $xyz$     | <i>ScanNet</i>  | <b>62.6</b>     |

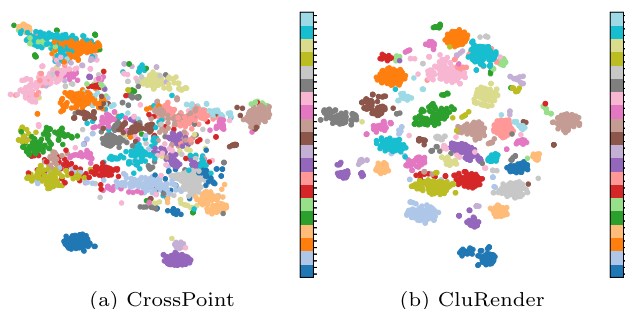
Pre. data indicates the data used as pre-training. The methods use different inputs, i.e.  $xyz+rgb$  stands for colored point cloud, while in  $xyz$ , no color is used. The bold values indicate the best performance



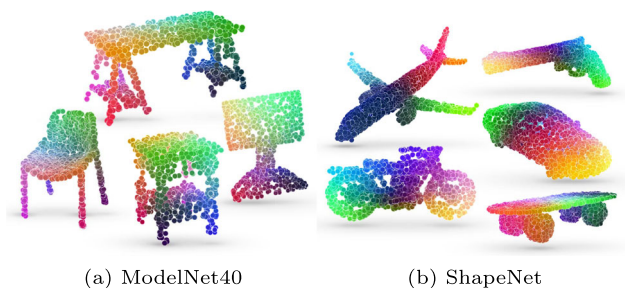
**Fig. 3** Semantic segmentation results on S3DIS (Armeni et al., 2016) of CluRender using the DGCNN encoder (left column) compared to the ground-truth annotations (right column). The black dashed box indicates the areas where segmentation failed

*Results* Table 3 reports the classification accuracy of CluRender, compared to the other approaches. Results show that the CluRender is more effective than the alternative pre-training methods on both datasets. Specifically, on ModelNet40, CluRender with PointNet backbone achieves the higher classification accuracy (90.4%) than the generative method ParAE (Eckart et al., 2021) the contrastive approach STRL (Huang et al., 2021) (88.3%). The linear SVM classification performance of our method even surpasses the performance of the fully supervised PointNet, which achieves an 89.2% test accuracy. With the DGCNN encoder, our method achieves a 92.3% test accuracy, outperforming the second-best MAE3D (Jiang et al., 2022) (92.1%) by 0.2%, and completion model OcCo (Wang et al., 2020) by 2.6%. CluRender outperforms OcCo (Wang et al., 2020) and Jigsaw3D (Sauder & Sievers, 2019) with both the encoding networks. Compared to jigsaw tasks that coarsely segment a point cloud into disjoint partitions, CluRender learns the partitioning function itself to softly assign point clouds into coherent clusters. Furthermore, CluRender surpasses all dedicated Transformer models in performance. For instance, it exceeds the accuracy of the intricate Point-M2AE model (Zhang et al., 2022) by 0.3%. Additionally, compared to our prior method, SoftClu, CluRender exhibits enhanced classifi-





**Fig. 4** T-SNE embeddings of (a) CrossPoint (Afham et al., 2022) and (b) CluRender on ModelNet40. Our method produces better separated and grouped clusters for different categories



**Fig. 5** Color-coded points based on PCA projections of the learned features: (left) ModelNet40, (right) ShapeNet

cation accuracy across PointNet, DGCNN, and Transformer backbones, thereby validating the efficacy of our rendering module. We also utilize visualization techniques to explore the pre-trained features before fine-tuning with the DGCNN encoder. Figure 4 shows the features visualized with T-SNE (Van der Maaten & Hinton, 2008) of CrossPoint and CluRender on ModelNet40. Our method yields a better separation of the features than CrossPoint, which indicates a better ability of CluRender in clustering objects in the feature space. Fig. 5 displays points, colored according to the PCA projections of network features, illustrating the effective embedding of geometric information by the pre-trained encoder. Despite implementing a softly equal partition constraint, our method does not enforce a rigorously equal partition.

#### 6.2.4 Part Segmentation

We evaluate CluRender on ShapeNetPart (Yi et al., 2016) for part segmentation, which predicts per-point part labels and requires a detailed understanding of local patterns. *ShapeNetPart* (Yi et al., 2016) contains 16,881 objects of 2,048 points from 16 categories with 50 parts in total. To ensure a fair comparison with prior work, we utilized the sampled point sets produced by (Pang et al., 2022) and trained the linear fully connected layer for 100 epochs using the AdamW optimizer (Loshchilov & Hutter, 2018) with a batch size of 24, an initial learning rate of 0.001, and a learning rate decay

**Table 3** Overall accuracy (OAcc) of linear SVM classification on ModelNet40 using pre-training on ShapeNet

| Method                              | OAcc $\uparrow$ |
|-------------------------------------|-----------------|
| <i>PointNet backbone</i>            |                 |
| DeepCluster (Caron et al., 2018)    | 86.3            |
| Jigsaw3D (Sauder & Sievers, 2019)   | 87.3            |
| Rotation3D (Poursaeed et al., 2020) | 88.6            |
| SwAV (Caron et al., 2020)           | 85.4            |
| OcCo (Wang et al., 2020)            | 88.7            |
| SimCLR (Chen and Hinton, 2020)      | 88.4            |
| STRL (Huang et al., 2021)           | 88.3            |
| ParAE (Eckart et al., 2021)         | 90.3            |
| CrossPoint (Afham et al., 2022)     | 89.1            |
| SoftClu (Mei et al., 2022)          | 90.3            |
| CluRender                           | <b>90.4</b>     |
| <i>DGCNN backbone</i>               |                 |
| DeepCluster (Caron et al., 2018)    | 90.4            |
| Jigsaw3D (Sauder & Sievers, 2019)   | 90.6            |
| Rotation3D (Poursaeed et al., 2020) | 90.8            |
| SwAV (Caron et al., 2020)           | 90.3            |
| OcCo (Wang et al., 2020)            | 90.7            |
| SimCLR (Chen and Hinton, 2020)      | 90.1            |
| STRL (Huang et al., 2021)           | 90.9            |
| ParAE (Eckart et al., 2021)         | 91.6            |
| CrossPoint (Afham et al., 2022)     | 91.2            |
| SoftClu (Mei et al., 2022)          | 91.9            |
| MAE3D (Jiang et al., 2022)          | 92.1            |
| CluRender                           | <b>92.3</b>     |
| <i>Transformer backbone</i>         |                 |
| ViT-OcCo (Wang et al., 2020)        | 89.6            |
| Point-BERT (Yu et al., 2022)        | 87.4            |
| Point-MAE (Pang et al., 2022)       | 91.0            |
| Point-M2AE (Zhang et al., 2022)     | 92.9            |
| SoftClu (Mei et al., 2022)          | 92.7            |
| CluRender                           | <b>93.2</b>     |

Bold values indicate best performance

of 0.5 every 20 epochs. We evaluated the segmentation quality of PointNet and DGCNN using overall accuracy (OAcc) and mean Intersection over Union (mIoU) metrics. mIoU is calculated by averaging IoUs for each part in an object before averaging the obtained values for each object class as defined in (Wang et al., 2020).

*Results* Table 4 reports the part segmentation results of CluRender in comparison with alternative approaches on ShapeNetPart (Yi et al., 2016). Notably, CluRender surpasses all competing approaches when employed with both PointNet and DGCNN encoders, excelling in terms of both OAcc and mIoU. When coupled with the PointNet encoder,



CluRender achieves an OAcc of 94.1% and an mIoU of 84.3%, marking an improvement over the state-of-the-art CrossPoint, which registers 93.2% OAcc and 82.7% mIoU, by 0.9% and 1.6% in OAcc and mIoU respectively. Similarly, using the DGCNN encoder, we attain 94.9% OA and 85.9% mIoU, outstripping the second-best approach, SoftClu (with 94.6% OAcc and 85.7% mIoU), by approximately 0.3% and 0.2% in OA and mIoU, respectively.

Table 4 also presents the results of part segmentation utilizing Transformer-based backbones initialized with various pre-training methods. We re-implemented the transformer results, employing both OcCo pretraining and random initialization, and achieved improved outcomes. By utilizing models pretrained by Point-MAE and Point-M2AE, which were provided by the authors, we were able to extract the overall accuracy for both Point-MAE and Point-M2AE. The proposed method, CluRender, achieves an exemplary 95.4% overall accuracy and an 86.9% instance mIoU, surpassing the second-best Point-M2AE (Zhang et al., 2022) by margins of 0.5% and 0.4%, respectively. These outcomes emphatically underscore the pivotal role of the proposed pre-training in enhancing segmentation tasks.

Figure 6 presents a collection of examples showcasing the part segmentation results, demonstrating a high success level in accurately identifying and segmenting most of the parts. However, the lights in the motorcycle cases were not successfully segmented because of the significant difficulty in distinguishing them from the handle parts. This difficulty can be attributed to the complexity of the image and the similarity in appearance between the light and handle parts, which pose a significant challenge for the segmentation algorithm. Despite this limitation, the overall performance of the segmentation algorithm is commendable, as it successfully identifies and separates the majority of the parts.

### 6.2.5 Few-Shot Learning

Few-shot learning (FSL) aims to train a model that generalizes with limited data. We conduct FSL ( $N$ -way  $K$ -shot learning) for the classification task on ModelNet40 (Sharma et al., 2016), where the model is evaluated on  $N$  classes that are randomly selected from the dataset, and each class contains  $K$  samples randomly sampled for each class. We use the same setting and train/test split as previous works (Wang et al., 2020; Afham et al., 2022) and report the mean and standard deviation across 10 runs. Table 5 shows the FSL results on ModelNet40, where CluRender outperforms prior works in all the FSL settings in the DGCNN backbone. Our method with PointNet backbone slightly underperforms in the 10-way 20-shot settings compared to the results of CrossPoint with PointNet. As for the Transformer backbone, our CluRender consistently achieves the best performance compared to other unsupervised methods.

**Table 4** Part segmentation results on the ShapeNetPart dataset using the pre-trained PointNet, DGCNN and Transformer backbones

| Method                            | OAcc $\uparrow$ | mIoU $\uparrow$ |
|-----------------------------------|-----------------|-----------------|
| <i>PointNet backbone</i>          |                 |                 |
| Random                            | 92.8            | 82.2            |
| Jigsaw3D (Sauder & Sievers, 2019) | 93.1            | 82.2            |
| OcCo (Wang et al., 2020)          | 93.4            | 83.4            |
| CrossPoint (Afham et al., 2022)   | 93.2            | 82.7            |
| SoftClu (Mei et al., 2022)        | 93.9            | 83.8            |
| CluRender                         | <b>94.1</b>     | <b>84.3</b>     |
| <i>DGCNN backbone</i>             |                 |                 |
| Random                            | 92.2            | 84.4            |
| Jigsaw3D (Sauder & Sievers, 2019) | 92.7            | 84.3            |
| OcCo (Wang et al., 2020)          | 94.4            | 85.0            |
| CrossPoint (Afham et al., 2022)   | 93.2            | 85.5            |
| SoftClu (Mei et al., 2022)        | 94.6            | 85.7            |
| CluRender                         | <b>94.9</b>     | <b>85.9</b>     |
| <i>Transformer backbone</i>       |                 |                 |
| From Scratch (Yu et al., 2022)    | 94.4            | 85.5            |
| PointViT-OcCo (Wang et al., 2020) | 94.4            | 85.6            |
| Point-Bert (Yu et al., 2022)      | –               | 85.6            |
| Point-MAE (Pang et al., 2022)     | 94.6            | 86.1            |
| MaskPoint (Liu et al., 2022)      | –               | 86.0            |
| Point-M2AE (Zhang et al., 2022)   | 94.9            | 86.5            |
| SoftClu (Mei et al., 2022)        | 94.9            | 86.1            |
| CluRender                         | <b>95.4</b>     | <b>86.9</b>     |

Bold values indicate best performance

## 6.3 Ablation Study and Analysis

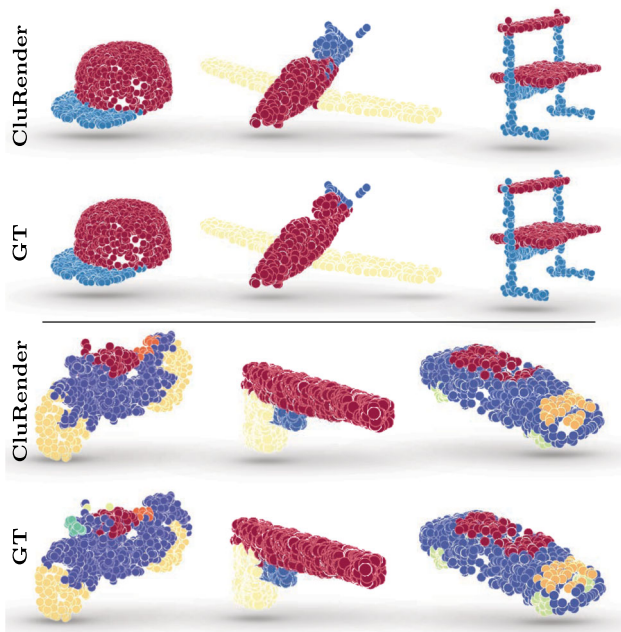
### 6.3.1 Impact of the Joint Learning

As our CluRender training networks using a joint learning objective, we thus first study the contribution of clustering in intra-modal learning-based clustering and cross-modal learning-based rendering. We conducted pre-training experiments by training the networks using (i) only clustering, (ii) only neural rendering, and (iii) both clustering and rendering. Our experiments were performed on the ModelNet40 dataset. The results of our study are presented in Table 6. We observed that using only clustering achieved the second-best performance, indicating that spatial information is still crucial in feature learning. Our method, which employs clustering and rendering, achieved the best performance on ModelNet40 with PointNet and DGCNN.

**Table 5** Few-shot object classification results on ModelNet40 using different backbones

| Backbone    | Method                          | 5-way             |                   | 10-way            |                   |
|-------------|---------------------------------|-------------------|-------------------|-------------------|-------------------|
|             |                                 | 10-shot           | 20-shot           | 10-shot           | 20-shot           |
| PointNet    | Rand                            | 52.0 ± 3.8        | 57.8 ± 4.9        | 46.6 ± 4.3        | 35.2 ± 4.8        |
|             | Jigsaw (Sauder & Sievers, 2019) | 66.5 ± 2.5        | 69.2 ± 2.4        | 56.9 ± 2.5        | 66.5 ± 1.4        |
|             | cTree (Sharma & Kaul, 2020)     | 63.2 ± 3.4        | 68.9 ± 3.0        | 49.2 ± 1.9        | 50.1 ± 1.6        |
|             | OcCo (Wang et al., 2020)        | 89.7 ± 1.9        | 92.4 ± 1.6        | 83.9 ± 1.8        | 89.7 ± 1.5        |
|             | CrossPoint (Afham et al., 2022) | 90.9 ± 4.8        | 93.5 ± 4.4        | 84.6 ± 4.7        | <b>90.2 ± 2.2</b> |
|             | SoftClu (Mei et al., 2022)      | 90.6 ± 4.0        | 93.8 ± 3.2        | 84.7 ± 3.6        | 90.1 ± 4.5        |
|             | CluRender                       | <b>91.1 ± 3.6</b> | <b>94.1 ± 2.7</b> | <b>85.0 ± 3.2</b> | 90.2 ± 3.9        |
| DGCNN       | Rand                            | 31.6 ± 2.8        | 40.8 ± 4.6        | 19.9 ± 2.1        | 16.9 ± 1.5        |
|             | Jigsaw (Sauder & Sievers, 2019) | 34.3 ± 1.3        | 42.2 ± 3.5        | 26.0 ± 2.4        | 29.9 ± 2.6        |
|             | cTree (Sharma & Kaul, 2020)     | 68.4 ± 3.4        | 71.6 ± 2.9        | 42.4 ± 2.7        | 43.0 ± 3.0        |
|             | OcCo (Wang et al., 2020)        | 90.6 ± 2.8        | 92.5 ± 1.9        | 82.9 ± 1.3        | 86.5 ± 2.2        |
|             | CrossPoint (Afham et al., 2022) | 92.5 ± 3.0        | 94.9 ± 2.1        | 83.6 ± 5.3        | 87.9 ± 4.2        |
|             | SoftClu (Mei et al., 2022)      | 93.6 ± 3.3        | 97.3 ± 2.0        | 89.1 ± 1.4        | 93.2 ± 3.4        |
|             | CluRender                       | <b>93.7 ± 3.1</b> | <b>97.4 ± 1.5</b> | <b>90.2 ± 1.4</b> | <b>93.6 ± 3.1</b> |
| Transformer | Rand                            | 87.8 ± 5.2        | 95.9 ± 2.3        | 89.4 ± 5.1        | 92.4 ± 4.6        |
|             | ViT-OcCo (Wang et al., 2020)    | 34.3 ± 1.3        | 42.2 ± 3.5        | 26.0 ± 2.4        | 29.9 ± 2.6        |
|             | Point-BERT (Yu et al., 2022)    | 94.6 ± 3.1        | 96.3 ± 2.7        | 91.0 ± 5.4        | 92.7 ± 5.1        |
|             | MaskPoint (Liu et al., 2022)    | 95.0 ± 3.7        | 97.2 ± 1.7        | 91.4 ± 4.0        | 93.4 ± 3.5        |
|             | Point-MAE (Pang et al., 2022)   | 96.3 ± 2.5        | 97.8 ± 1.8        | 92.6 ± 4.1        | 95.0 ± 3.0        |
|             | Point-M2AE (Zhang et al., 2022) | 96.8 ± 1.8        | 98.3 ± 1.4        | 92.3 ± 4.5        | 95.0 ± 3.0        |
|             | SoftClu (Mei et al., 2022)      | 96.5 ± 2.5        | 98.0 ± 1.8        | 93.1 ± 4.1        | 95.3 ± 3.3        |
|             | CluRender                       | <b>97.2 ± 2.3</b> | <b>98.4 ± 1.3</b> | <b>93.7 ± 4.0</b> | <b>96.0 ± 2.9</b> |

We report mean and standard error over 10 runs for each setting and report mean accuracy (%) with standard deviation. Bold values indicate best performance



**Fig. 6** Part segmentation results on ShapeNetPart (Yi et al., 2016) of CluRender using the DGCNN encoder (top row) compared to the ground-truth annotations (bottom row)

**Table 6** Ablation study

| Encoder  | Cluster | Render | Accuracy    |
|----------|---------|--------|-------------|
| PointNet | ✓       | ×      | 90.0        |
|          | ×       | ✓      | 88.9        |
|          | ✓       | ✓      | <b>90.4</b> |
| DGCNN    | ✓       | ×      | 91.4        |
|          | ×       | ✓      | 91.2        |
|          | ✓       | ✓      | <b>92.3</b> |

Bold values indicate best performance. We investigate the effects of different designs of CluRender, and report the classification accuracy (%) after fine-tuning on ModelNet40. All models are tested with 1024 points

### 6.3.2 Computation of Soft-Labels

We assess our strategy for soft-label assignment based on optimal transport (OT) by comparing it with a typical L2 distance-based approach on ModelNet40. Therefore, we assess CluRender by using  $\Gamma$  computed with Eq. (12) and by using the L2 approach in Caron et al. (2018). In this experiment, we do not consider the rendering module. Table 7 shows that OT achieves the best performance on all the

**Table 7** Ablation study of CluRender on ModelNet40 with soft-labels computed with our approach (OT) and with a typical distance-based assignment (L2)

| Dataset    | Encoder  | Accuracy L2 | OT          |
|------------|----------|-------------|-------------|
| ModelNet40 | PointNet | 86.5        | <b>90.3</b> |
|            | DGCNN    | 90.4        | <b>91.9</b> |

Bold values indicate best performance

**Table 8** Ablation study results of CluRender with different number of clusters  $J$ 

| Clusters | 16   | 32   | 48   | 64   | 72   | 96   | 112  | 128  |
|----------|------|------|------|------|------|------|------|------|
| PointNet | 92.4 | 93.0 | 93.1 | 93.5 | 93.4 | 93.3 | 93.2 | 93.1 |
| DGCNN    | 94.2 | 94.8 | 94.6 | 94.8 | 94.7 | 94.6 | 94.6 | 94.5 |

**Table 9** Ablation study results of CluRender with different number of views in multi-view rendering

| Method   | 4-views | 8-views | 12-views | 24-views |
|----------|---------|---------|----------|----------|
| PointNet | 88.7    | 88.9    | 88.9     | 88.8     |
| DGCNN    | 91.2    | 91.2    | 91.0     | 91.1     |

datasets with both PointNet and DGCNN encoders. This is due to the equal partition constraint which prevents solutions from being assigned to the same cluster and affecting the performance.

### 6.3.3 Number of Clusters

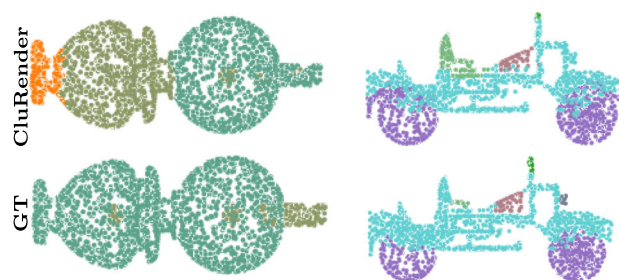
We assess the effect of selecting different numbers of cluster partitions  $J$  by using ModelNet40. In this experiment, we do not consider the render module. We pre-train CluRender with different values of  $J$ , i.e. from 16 to 128 with 8 different number clusters (16, 32, 48, 64, 72, 96, 112, 128), and report the results in Table 8. CluRender achieves the best results with  $J = 64$  for both PointNet and DGCNN. We observed stability with the results throughout different values of  $J$ .

### 6.3.4 Number of Views

We analyze the influence of multi-view rendering in our pre-training performance. We perform object classification tasks on ModelNet40 using shapes rendered with 4, 8, 12, and 24 views. The clustering module is not taken into consideration for this experiment. The results are presented in Table 9. For PointNet, the performance is optimal when 8 views are used, while for DGCNN, using 4 or 8 views is generally sufficient.

### 6.3.5 Computational Time

CluRender is exclusively utilized for pre-training, wherein each iteration encompasses two components: a forward pass

**Fig. 7** Part segmentation failure cases using CluRender on ShapeNet-Part (Yi et al., 2016) with the Transformer (top row), compared to the corresponding ground-truth annotations (bottom row)**Table 10** Pre-training time of CluRender for Transformer backbone and part segmentation results

| Method     | Random | Point-M2AE | CluRender (Ours) |
|------------|--------|------------|------------------|
| Times (s)  | 0.0    | 583.10     | 188.99           |
| Part(mIoU) | 85.5%  | 86.5%      | 86.9%            |

through the backbone and Sinkhorn optimization and rendering. The Sinkhorn component, specifically employed for pseudo-label generation, does not notably impact training time as it does not involve gradient backpropagation. Given that the cluster quantity is relatively minimal, the time complexity remains at  $O(N)$ , with  $N$  denoting the number of data points, thereby not significantly influencing overall training time. Note that the inference time for each utilized backbone remains unaltered. Table 10 reports the time of pretraining a transformer backbone and the mIoU results of the part segmentation. Experiments executed on one RTX A4500 GPU (20G) and one Intel(R) Core(TM) i7-13700K CPU, adds an average overhead of 188.99s per iteration on ShapeNet with a Transformer backbone. In addition, a comparative analysis, which includes results without pre-training (via random initialization), unequivocally attests to the efficacy of pre-training, evidenced by a performance augmentation exceeding 1.4 percent part segmentation tasks. It is worth noting that the “pre-training + fine-tuning” procedure does consume considerable time, akin to other pre-training methods. Enhancing the efficiency of the training process emerges as a compelling avenue for future exploration.

### 6.3.6 Limitations

CluRender presumes a uniform distribution prior, possibly overlooking the intrinsic imbalance often present in real-world data. Some point cloud scenarios, such as indoor spaces and street scenes, frequently exhibit notable content imbalances, thus highlighting the potential merit of exploring non-uniform clustering approaches. Figure 7 presents two instances of failure in part segmentation on the ShapeNet-Part dataset, attributable to this uniformity assumption. This

adherence compels the encoder to learn disparate features to comply with the constraint. Therefore, relaxing the uniform distribution assumption in favor of employing arbitrary prior distributions to enhance performance constitutes a pivotal direction for our future work.

## 7 Conclusions

We presented CluRender, a novel unsupervised representation learning method for 3D point cloud understanding that does not require data augmentation. CluRender leverages clustering and neural rendering techniques to train the feature encoders in an implicit manner. Our results demonstrate that the pre-trained representations obtained from our approach can be effectively transferred to various downstream 3D understanding tasks, including 3D object detection, semantic segmentation, classification, part segmentation, and few-shot learning. Additionally, CluRender is not reliant on specific deep network architectures, making it a versatile method for feature extraction from raw point cloud data that can enhance the performance of other 3D models.

**Acknowledgements** This work was supported by the PNRR project FAIR - Future AI Research (PE00000013), under the NRRP MUR program funded by the NextGenerationEU.

**Funding** Open Access funding enabled and organized by CAUL and its Member Institutions.

**Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

## References

- Achlioptas, P., Diamanti, O., Mitliagkas, I., & Guibas, L. (2018). *Learning representations and generative models for 3d point clouds*. ICML.
- Afham, M., Dissanayake, I., Dissanayake, D., Dharmasiri, A., Thilakarathna, K., & Rodrigo, R. (2022). *CrossPoint: Self-supervised cross-modal contrastive learning for 3D point cloud understanding*. CVPR.
- Aliiev, K. A., Sevastopolsky, A., Kolos, M., Ulyanov, D., & Lempitky, V. (2020). *Neural point-based graphics neural point-based graphics* (pp. 696–712). ECCV.
- Armeni, I., Sener, O., Zamir, A. R., Jiang, H., Brilakis, I., Fischer, M., & Savarese, S. (2016). *3d semantic parsing of large-scale indoor spaces*. CVPR.
- Asano, Y. M., Rupprecht, C., & Vedaldi, A. (2020). *Self-labelling via simultaneous clustering and representation learning*. ICLR.
- Biswas, J., & Veloso, M. (2012). *Depth camera based indoor mobile robot localization and navigation*. ICRA.
- Caron, M., Bojanowski, P., Joulin, A., & Douze, M. (2018). *Deep clustering for unsupervised learning of visual features*. ECCV.
- Caron, M., Misra, I., Mairal, J., Goyal, P., Bojanowski, P., & Joulin, A. (2020). Unsupervised learning of visual features by contrasting cluster assignments. *NeurIPS*, 33, 9912–9924.
- Chang, A. X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., & Yu, F. (2015). ShapeNet: An information-rich 3D model repository.
- Chen, S., Duan, C., Yang, Y., Li, D., Feng, C., & Tian, D. (2019). Deep unsupervised learning of 3D point clouds via graph topology inference and filtering. *TIP*293183–3198
- Chen, T., & Hinton, G. (2020). A simple framework for contrastive learning of visual representations. ICML
- Chen, X., & He, K. (2021). *Exploring simple Siamese representation learning*. CVPR.
- Cuturi, M. (2013). Sinkhorn distances: Lightspeed computation of optimal transport. *NeurIPS*
- Dai, A., Chang, A. X., Savva, M., Halber, M., Funkhouser, T., & Nießner, M. (2017). Scannet: Richly-annotated 3d reconstructions of indoor scenes. *CVPR* (5828–5839).
- Dong, R., Qi, Z., Zhang, L., Zhang, J., Sun, J., Ge, Z., & Ma, K. (2022). Autoencoders as cross-modal teachers: Can pretrained 2D image transformers help 3D representation learning? *arXiv preprint arXiv:2212.08320*
- Du, B., Gao, X., Hu, W., & Li, X. (2021). Self-contrastive learning with hard negative sampling for self-supervised point cloud learning. *ACM MM* (3133–3142).
- Eckart, B., Yuan, W., Liu, C., & Kautz, J. (2021). *Self-supervised learning on 3D point clouds by learning discrete generative models*. CVPR.
- Gao, X., Hu, W., & Qi, G. J. (2020). *GraphTER: Unsupervised learning of graph transformation equivariant representations via auto-encoding node-wise transformations*. CVPR.
- Grill, J. B., Strub, F., Altché, F., Tallec, C., Richemond, P., Buchatskaya, E., & Gheshlaghi Azar, M. (2020). *Bootstrap your own latent: A new approach to self-supervised learning*. *NeurIPS*.
- Han, Z., Wang, X., Liu, Y. S., & Zwicker, M. (2019). Multi-angle point cloud-vae: Unsupervised feature learning for 3D point clouds from multiple angles by joint self-reconstruction and half-to-half prediction. *ICCV* (10441–10450).
- Hassani, K., & Haley, M. (2019). *Unsupervised multi-task feature learning on point clouds*. ICCV.
- Hou, J., Xie, S., Graham, B., Dai, A., & Nießner, M. (2021). Pri3d: Can 3D priors help 2D representation learning? *ICCV* (5693–5702).
- Huang, S., Xie, Y., Zhu, S. C., & Zhu, Y. (2021). *Spatio-temporal self-supervised representation learning for 3D point clouds*. ICCV.
- Insafutdinov, E., & Dosovitskiy, A. (2018). Unsupervised learning of shape and pose with differentiable point clouds. *Neurips*31
- Jiang, J., Lu, X., Zhao, L., Dazeley, R., & Wang, M. (2022). *Masked autoencoders in 3D point cloud representation learning*. ECCV.
- Jing, L., Zhang, L., & Tian, Y. (2021). Self-supervised feature learning by cross-modality and cross-view correspondences. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (1581–1591).
- Li, L., & Heizmann, M. (2022). A closer look at invariances in self-supervised pre-training for 3D vision. In *European conference on computer vision* (656–673).
- Li, Y., Ma, L., Zhong, Z., Liu, F., Chapman, M. A., Cao, D., & Li, J. (2020). Deep learning for LiDAR point clouds in autonomous driving: A review. *TNNLS*



- Lin, X., Chen, K., & Jia, K. (2021). Object point cloud classification via poly-convolutional architecture search. *ACM MM* (807–815).
- Liu, H., Cai, M., & Lee, Y. J. (2022). Masked discrimination for self-supervised learning on point clouds. *ECCV* (657–675).
- Liu, K., Xiao, A., Zhang, X., Lu, S., & Shao, L. (2023). Fac: 3d representation learning via foreground aware feature contrast. *arXiv preprint arXiv:2303.06388*.
- Liu, X., Han, Z., Wen, X., Liu, Y. S., & Zwicker, M. (2019). L2g auto-encoder: Understanding point clouds by local-to-global reconstruction with hierarchical self-attention. *ACM MM* (989–997).
- Loshchilov, I., & Hutter, F. (2018). *Decoupled weight decay regularization*. ICLR.
- Mei, G. (2021). Point cloud registration with self-supervised feature learning and beam search. *DICTA* (01–08).
- Mei, G., Huang, X., Zhang, J., & Wu, Q. (2022). Overlap-guided coarse-to-fine correspondence prediction for point cloud registration. *ICME* (1–6).
- Mei, G., Saltori, C., Poesi, F., Zhang, J., Ricci, E., Sebe, N., & Wu, Q. (2022). *Data augmentation-free unsupervised learning for 3D point cloud understanding*. BMVC.
- Mensch, A., & Peyré, G. (2020). Online sinkhorn: Optimal transport distances from sample streams. *Neurips331657–1667*
- Misra, I., Girdhar, R., & Joulin, A. (2021). An end-to-end transformer model for 3d object detection. *CVPR* (2906–2917).
- Moon, T. K. (1996). The expectation-maximization algorithm. *IEEE Signal processing magazine*
- Pang, Y., Wang, W., Tay, F. E., Liu, W., Tian, Y., & Yuan, L. (2022). Masked autoencoders for point cloud self-supervised. *arXiv preprint arXiv:2203.06604*.
- Park, Y., Lepetit, V., & Woo, W. (2008). *Multiple 3D object tracking for augmented reality*. ISMAR.
- Peyré, G., & Cuturi, M. (2019). Computational optimal transport: With applications to data science. *Foundations and Trends® in Machine Learning*.
- Poesi, F., & Boscaini, D. (2022). Learning general and distinctive 3D local deep descriptors for point cloud registration. *TPAMI*.
- Porter, T., & Duff, T. (1984). Compositing digital images. *ACM SIG-GRAPH* (253–259).
- Poursaeed, O., Jiang, T., Qiao, H., Xu, N., & Kim, V. G. (2020). Self-supervised learning of point clouds via orientation estimation. *3DV*.
- Qi, C. R., Litany, O., He, K., & Guibas, L. J. (2019). Deep hough voting for 3D object detection in point clouds. *ICCV* (9277–9286).
- Qi, C. R., Su, H., Mo, K., & Guibas, L. J. (2017). Pointnet: Deep learning on point sets for 3D classification and segmentation. *CVPR* (652–660).
- Qi, C. R., Yi, L., Su, H., & Guibas, L. J. (2017). Pointnet++: Deep hierarchical feature learning on point sets in a metric. *NeurIPS* (5099–5108).
- Rao, Y., Liu, B., Wei, Y., Lu, J., Hsieh, C. J., & Zhou, J. (2021). Randomrooms: Unsupervised pre-training from synthetic shapes and randomized layouts for 3d object detection. *ICCV* (3283–3292).
- Rao, Y., Lu, J., & Zhou, J. (2020). *Global-local bidirectional reasoning for unsupervised representation learning of 3d point clouds*. CVPR.
- Ronneberger, O., Fischer, P., & Brox, T. (2015). U-net: Convolutional networks for biomedical image. *MICCAI* (234–241).
- Sanghi, A. (2020). *Info3d: Representation learning on 3d objects using mutual information maximization and contrastive learning*. *Eccv: ECCV*.
- Sarmad, M., Lee, H.J., & Kim, Y.M. (2019). RI-gan-net: A reinforcement learning agent controlled GAN network for real-time point cloud shape completion. *CVPR Cvp* (5898–5907).
- Sauder, J., & Sievers, B. (2019). Self-supervised deep learning on point clouds by reconstructing space. *NeurIPS* (12942–12952).
- Sharma, A., Grau, O., & Fritz, M. (2016). Vconv-dae: Deep volumetric shape learning without object labels. *ECCV* (236–250).
- Sharma, C., & Kaul, M. (2020). Self-supervised few-shot learning on point clouds. *NeurIPS337212–7221*
- Shi, Y., Xu, M., Yuan, S., & Fang, Y. (2020). Unsupervised deep shape descriptor with point distribution learning. *CVPR* (9353–9362).
- Sun, Y., Wang, Y., Liu, Z., Siegel, J., & Sarma, S. (2020). Point-grow: Autoregressively learned point cloud generation with self-attention. *WACV* (61–70).
- Van der Maaten, L., & Hinton, G. (2008). *Visualizing data using t-SNE*. *JMLR*.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., & Polosukhin, I. (2017). Attention is all you need. *Neurips30*
- Wang, H., Liu, Q., Yue, X., Lasenby, J., & Kusner, M. J. (2020). *Unsupervised point cloud pre-training via view-point occlusion, completion*. ICCV.
- Wang, P. S., Yang, Y. Q., Zou, Q. F., Wu, Z., Liu, Y., & Tong, X. (2021). Unsupervised 3D learning for shape analysis via multiresolution instance discrimination. *AAAI* (35, 2773–2781).
- Wang, Y., & Bue, A. D. (2020). *Where to explore next? ExHistCNN for history-aware autonomous 3D exploration*. *ECCV*.
- Wang, Y., Carletti, M., Setti, F., Cristani, M., & Bue, A. D. (2019). *Active 3d classification of multiple objects in cluttered scenes*. *ICCVW*.
- Wang, Y., Sun, Y., Liu, Z., Sarma, S. E., Bronstein, M. M., & Solomon, J. M. (2019). Dynamic graph CNN for learning on point. *ACM TOG* 3851–12,
- Wang, Z., Yu, X., Rao, Y., Zhou, J., & Lu, J. (2022). P2p: Tuning pre-trained image models for point cloud analysis with point-to-pixel prompting. *arXiv preprint arXiv:2208.02812*.
- Wiles, O., Gkioxari, G., Szeliski, R., & Johnson, J. (2020). Synsin: End-to-end view synthesis from a single image. *CVPR* (7467–7477).
- Wiles, O., & Zisserman, A. (2019). Learning to predict 3D surfaces of sculptures from single and multiple views. *IJCV* 127111780–1800
- Xie, S., Gu, J., Guo, D., Qi, C. R., Guibas, L., & Litany, O. (2020). *Pointcontrast: Unsupervised pre-training for 3D point cloud understanding*. *ECCV*.
- Xing, J., Luan, F., Yan, L. Q., Hu, X., Qian, H., & Xu, K. (2022). Differentiable rendering using RGBXY derivatives and optimal transport. *ACM TOG*, 41(6), 1–13.
- Xu, Q., Wang, W., Ceylan, D., Mech, R., & Neumann, U. (2019). Disn: Deep implicit surface network for high-quality single-view 3D reconstruction. *Neurips* 32
- Yamada, R., & Ogata, T. (2022). Point cloud pre-training with natural 3d structures. *CVPR* (21283–21293).
- Yan, X., Gao, J., Zheng, C., Zheng, C., Zhang, R., Cui, S., & Li, Z. (2022). 2dpass: 2d priors assisted semantic segmentation on lidar point clouds. *ECCV* (677–695).
- Yang, Y., Feng, C., Shen, Y., & Tian, D. (2018). Foldingnet: Point cloud auto-encoder via deep grid deformation. *CVPR* (206–215).
- Yi, L., Kim, V. G., Ceylan, D., Shen, I. C., Yan, M., Su, H., & Guibas, L. (2016). A scalable active framework for region annotation in 3D shape collections. *ACM TOG*
- Yin, J., Zhou, D., Zhang, L., Fang, J., Xu, C. Z., Shen, J., & Wang, W. (2022). Proposalcontrast: Unsupervised pre-training for lidar-based 3D object. *ECCV* (17–33).
- Yu, X., Tang, L., Rao, Y., Huang, T., Zhou, J., & Lu, J. (2022). Pointbert: Pre-training 3d point cloud transformers with masked point modeling. *CVPR* (19313–19322).
- Zhang, R., Guo, Z., Gao, P., Fang, R., Zhao, B., Wang, D., & Li, H. (2022). Point-M2AE: Multi-scale masked autoencoders for hierarchical point cloud pre-training. *arXiv preprint arXiv:2205.14401*
- Zhang, Z., & Misra, I. (2021). Self-supervised pretraining of 3d features on any point-cloud. *ICCV* (10252–10263).
- Zhao, H., Jiang, L., Jia, J., Torr, P. H., & Koltun, V. (2021). Point transformer. *ICCV* (16259–16268).



- Zhou, Y., Wang, Y., Poiesi, F., Qin, Q., & Wan, Y. (2022). Loop closure detection using local 3D deep descriptors. *IEEE RAL*
- Ziegler, A., & Asano, Y.M. (2022). Self-supervised learning of object parts for semantic segmentation Self-supervised learning of object parts for semantic segmentation. *CVPR* (14502–14511).

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.