

FoldingNet: Interpretable Unsupervised Learning on 3D Point Clouds

Yang, Y.; Feng, C.; Shen, Y.; Tian, D.

TR2017-193 December 2017

Abstract

Recent deep networks that directly handle points in a point set, e.g., PointNet, have been state-of-the-art for supervised semantic learning tasks on point clouds such as classification and segmentation. In this work, a novel end-to-end deep auto-encoder is proposed to address unsupervised learning challenges on point clouds. On the encoder side, a graph-based enhancement is enforced to promote local structures on top of PointNet. Then, a novel folding based approach is proposed in the decoder, which folds a 2D grid onto the underlying 3D object surface of a point cloud. The proposed decoder only uses about 7% parameters of a decoder with fully-connected neural networks, yet leads to a more discriminative representation that achieves higher linear SVM classification accuracy than the benchmark. In addition, the proposed decoder structure is shown, in theory, to be a generic architecture that is able to reconstruct an arbitrary point cloud from a 2D grid. Finally, this folding-based decoder is interpretable since the reconstruction could be viewed as a fine granular warping from the 2D grid to the point cloud surface.

arXiv

This work may not be copied or reproduced in whole or in part for any commercial purpose. Permission to copy in whole or in part without payment of fee is granted for nonprofit educational and research purposes provided that all such whole or partial copies include the following: a notice that such copying is by permission of Mitsubishi Electric Research Laboratories, Inc.; an acknowledgment of the authors and individual contributions to the work; and all applicable portions of the copyright notice. Copying, reproduction, or republishing for any other purpose shall require a license with payment of fee to Mitsubishi Electric Research Laboratories, Inc. All rights reserved.

FoldingNet: Interpretable Unsupervised Learning on 3D Point Clouds

Yaoqing Yang*[†]

yyaoqing@andrew.cmu.edu

Chen Feng[‡]

cfeng@merl.com

Yiru Shen[§]

yirus@g.clemson.edu

Dong Tian[‡]

tian@merl.com

[†]Carnegie Mellon University

[‡]Mitsubishi Electric Research Laboratories (MERL)

[§]Clemson University

Abstract

Recent deep networks that directly handle points in a point set, e.g., PointNet, have been state-of-the-art for supervised semantic learning tasks on point clouds such as classification and segmentation. In this work, a novel end-to-end deep auto-encoder is proposed to address unsupervised learning challenges on point clouds. On the encoder side, a graph-based enhancement is enforced to promote local structures on top of PointNet. Then, a novel folding-based approach is proposed in the decoder, which folds a 2D grid onto the underlying 3D object surface of a point cloud. The proposed decoder only uses about 7% parameters of a decoder with fully-connected neural networks, yet leads to a more discriminative representation that achieves higher linear SVM classification accuracy than the benchmark. In addition, the proposed decoder structure is shown, in theory, to be a generic architecture that is able to reconstruct an arbitrary point cloud from a 2D grid. Finally, this folding-based decoder is interpretable since the reconstruction could be viewed as a fine granular warping from the 2D grid to the point cloud surface.

1. Introduction

3D point cloud processing and understanding are usually deemed more challenging than 2D images mainly due to a fact that point cloud samples live on an irregular structure while 2D image samples (pixels) rely on a 2D grid in the image plane with a regular spacing. Point cloud geometry is typically represented by a set of sparse 3D points. Such a data format makes it difficult to apply traditional deep learning framework on point clouds. E.g. for each sample, traditional convolutional neural network (CNN) requires its neighboring samples to appear at some fixed spatial orientations and distances so as to facilitate the convolution. Unfortunately, point cloud samples typically do not follow such constraints. One way to alleviate the problem is to voxelize a point cloud to mimic the image representation and then to operate on voxels. The downside of this approach is that voxelization has to either sacrifice the rep-

*This work is supported by MERL.


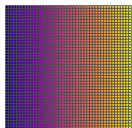
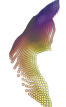
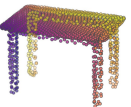

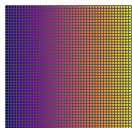

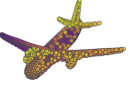

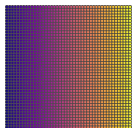



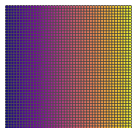
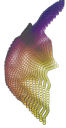


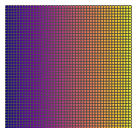


Input	2D grid	1st folding	2nd folding
			
			
			
			
			

Table 1. **Illustration of the two-step-folding decoding.** Column one contains the original point cloud samples from the ShapeNet dataset [54]. Column two illustrates the 2D grid points to be folded during decoding. Column three contains the output after one folding operation. Column four contains the output after two folding operations. This output is also the reconstructed point cloud. We use a color gradient to illustrate the correspondence between the 2D grid in column two and the reconstructed point clouds after folding operations in the last two columns. Best viewed in color.

resentation accuracy or incurs huge redundancies, that may pose an unnecessary cost in the subsequent processing, either at a compromised performance or an rapidly increased processing complexity. Related prior-arts will be reviewed in Section 1.1.

In this work, we focus on the emerging field of unsupervised learning for point clouds. We propose an auto-encoder (AE) that is referenced as FoldingNet. The output from the bottleneck layer in the auto-encoder is called

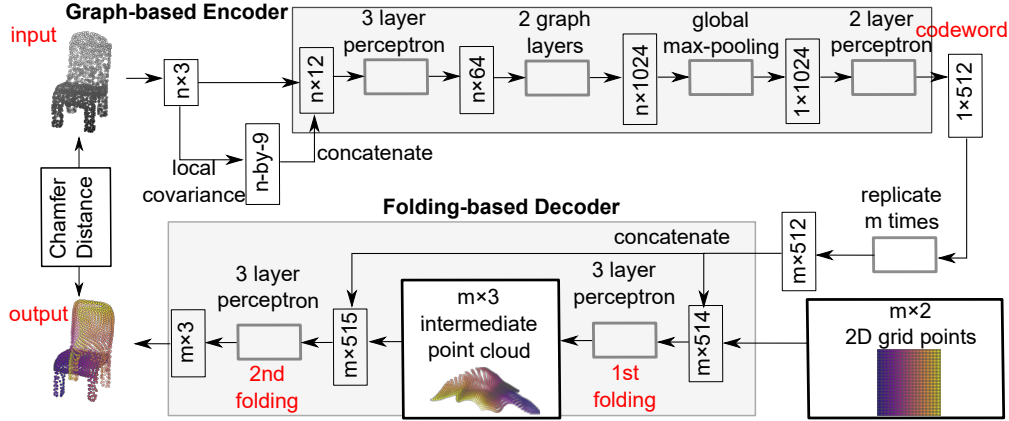


Figure 1. **FoldingNet Architecture.** The graph-layers are the graph-based max-pooling layers mentioned in (2) in Section 2.1. The 1st and the 2nd folding are both implemented by concatenating the codeword to the feature vectors followed by a 3-layer perceptron. Each perceptron independently applies to the feature vector of a single point as in [39], i.e., applies to the rows of the m -by- k matrix.

a codeword that can be used as a high-dimensional embedding of an input point cloud. We are going to show that a 2D grid structure is not only a sampling structure for imaging, but can indeed be used to construct a point cloud through the proposed *folding* operation. This is based on the observation that the 3D point clouds of our interest are obtained from object surfaces: either discretized from boundary representations in CAD and computer graphics, or sampled from line-of-sight sensors like LIDAR. Intuitively, any 3D object surface could be transformed to a 2D plane through certain operations like cutting, squeezing, and stretching. The inverse procedure is to glue those 2D point samples back onto an object surface via certain folding operations, which are initialized as 2D grid samples. As illustrated in Table 1, to reconstruct a point cloud, successive folding operations are joined to reproduce the surface structure. The points are colorized to show the correspondence between the initial 2D grid samples and the reconstructed 3D point samples. Using the folding-based method, the challenges from the irregular structure of point clouds are well addressed by directly introducing such an implicit 2D grid constrain in the decoder, which avoids the costly 3D voxelization in other works [53]. It will be demonstrated later that the folding operations can build an arbitrary surface provided a proper codeword.

Despite being strongly expressive in reconstructing point clouds, the folding operation is simple: it is started by augmenting the 2D grid points with the codeword obtained from the encoder, which is then processed through a 3-layer perceptron. The proposed decoder is simply a concatenation of two folding operations. This design makes the proposed decoder much smaller in parameter size than the fully-connected decoder proposed in the recent work [1]. In Section 4.3, we show that the number of parameters of our folding-based decoder is about 7% of the fully connected decoder in [1]. Although the proposed decoder has a

simple structure, we theoretically show in Theorem 3.2 that the folding-based decoding structure is universal in that one folding operation that uses only a 2-layer perceptron can already reproduce arbitrary point-cloud structure. Therefore, it is not surprising that our FoldingNet auto-encoder exploiting two consecutive folding operations can produce elaborate structures. Notice that when data are from volumetric format instead of 2D surfaces, 3D grid may perform better.

To show the efficiency of FoldingNet auto-encoder for unsupervised representation learning, we follow the experimental settings in [1] and test the transfer classification accuracy from ShapeNet dataset [7] to ModelNet dataset [54]. The FoldingNet auto-encoder is trained using ShapeNet dataset, and tested out by extracting codewords from ModelNet dataset. Then, we train a linear SVM classifier to test the discrimination effectiveness of the extracted codewords. The transfer classification accuracy is 88.4% on the ModelNet dataset with 40 shape categories. This classification accuracy is even close to the state-of-the-art supervised training result [39]. To achieve the best classification performance and least reconstruction loss, we use a graph-based encoder structure that is different from [39]. This graph-based encoder is based on the idea of local feature pooling operations and is able to retrieve and propagate local structural information along the graph structure.

Interpretability is an important aspect that people care about when using deep neural networks. In the proposed FoldingNet, the interpretability comes from two aspects. First, the proposed folding architecture is designed with a clear geometric interpretation: we want to impose a “virtual force” to deform/cut/stretch a 2D grid lattice onto a 3D object surface, while such a deformation force should be influenced or regulated by interconnections induced by the lattice neighborhood. Second, the intermediate steps of the folding process and the training process can be illustrated

by network outputs, and hence the gradual change of the folding forces can be visualized.

Now we summarize our contributions in this work:

- We train an end-to-end unsupervised deep auto-encoder that consumes point clouds directly without preprocessing.
- We propose a new decoding operation called folding and theoretically show it is universal in point cloud reconstruction.
- We show by experiments on major datasets that folding can achieve higher classification accuracy than other unsupervised methods.
- We show that the folding-based decoder is interpretable by visualizing the training process and the “folding” process on different point models.

1.1. Related works

Applications of learning on point clouds include shape completion and recognition [54], unmanned autonomous vehicles [34], 3D object detection, recognition and classification [9, 31, 38, 39, 45, 46, 50], contour detection [20], layout inference [16], scene labeling [29], category discovery [57], point classification, dense labeling and segmentation [3, 10, 13, 19, 21, 24, 26, 35, 39, 51, 52, 55],

Most deep neural networks designed for 3D point clouds are based on the idea of partitioning the 3D space into regular voxels and extending 2D CNNs to voxels, such as [4, 11, 35], including the the work on 3D generative adversarial network [53]. The main problem of voxel-based networks is the fast growth of neural-network size with the increasing spatial resolution. Some other options include octree-based [42] and kd-tree-based [28] neural networks. Recently, it is shown that neural networks based on purely 3D point representations [1, 39–41] work quite efficiently for point clouds. The point-based neural networks can reduce the overhead of converting point clouds into other data formats (such as octrees and voxels), and in the meantime avoid the information loss due to the conversion.

The only work that we are aware of on end-to-end deep auto-encoder that directly work on point clouds is [1]. The AE designed in [1] is for the purpose of extracting features for generative networks. To encode, it sorts the 3D points using the lexicographic order and applies a 1D CNN on the point sequence. To decode, it applies a three-layer fully connected network. This simple structure turns out to outperform all existing unsupervised works on representation extraction of point clouds in terms of the transfer classification accuracy from the ShapeNet dataset to the ModelNet dataset [1]. Our method, which has a graph-based encoder and a folding-based decoder, outperforms this method in transfer classification accuracy on the ModelNet40 dataset [1]. Moreover, compared to [1], our AE design is much more interpretable: the encoder learns the local shape in-

formation and combines information by max-pooling on a nearest-neighbor graph, and the decoder learns a *force* to fold a two-dimensional grid twice in order to warp the grid into the shape of the point cloud, using the information obtained by the encoder.

It is hard for purely point-based neural networks to extract local neighborhood structure around points, i.e., features of neighboring points instead of individual ones. Some attempts for this are made in [1, 40]. In this work, we exploit local neighborhood features using a graph-based framework. Deep learning on graph-structured data is not a new idea. There are tremendous amount of works on applying deep learning onto irregular data such as graphs and point sets [2, 5, 6, 12, 14, 15, 22, 23, 27, 30, 33, 36, 37, 41, 44, 49, 56]. Although using graphs as a processing framework for deep learning on point clouds is a natural idea, only several seminal works made attempts in this direction [5, 36, 44]. These works try to generalize the convolution operations from 2D images to graphs. However, since it is hard to define convolution operations on graphs, we use a simple graph-based neural network layer that is different from previous works: we construct the K-nearest neighbor graph (k-NNG) and repeatedly conduct the max-pooling operations in each node’s neighborhood. It generalizes the global max-pooling operation proposed in [39] in that the max-pooling is only applied to each local neighborhood to generate local data signatures. Compared to the above graph based convolution networks, our design is simpler and computationally efficient as in [39]. K-NNGs are also used in other applications of point clouds without the deep learning framework such as surface detection, 3D object recognition, 3D object segmentation and compression [18, 47, 48].

1.2. Preliminaries and Notation

We will often denote the point set by S . We use bold lower-case letters to represent vectors, such as \mathbf{x} , and use bold upper-case letters to represent matrices, such as \mathbf{A} . The codeword is always represented by θ . We call a matrix m -by- n or $m \times n$ if it has m rows and n columns.

2. FoldingNet Auto-encoder on Point Clouds

Now we propose the FoldingNet deep auto-encoder. The structure of the auto-encoder is shown in Figure 1. The input to the encoder is an n -by-3 matrix. Each row of the matrix is composed of the 3D position (x, y, z) . The output is an m -by-3 matrix, representing the reconstructed point positions. The number of reconstructed points m is not necessarily the same as n . Suppose the input contains the point set S and the reconstructed point set is the set \hat{S} . Then, the reconstruction error for \hat{S} is computed using a layer defined

as the (extended) Chamfer distance,

$$d_{CH}(S, \hat{S}) = \max \left\{ \frac{1}{|S|} \sum_{\mathbf{x} \in S} \min_{\hat{\mathbf{x}} \in \hat{S}} \|\mathbf{x} - \hat{\mathbf{x}}\|_2, \frac{1}{|\hat{S}|} \sum_{\hat{\mathbf{x}} \in \hat{S}} \min_{\mathbf{x} \in S} \|\hat{\mathbf{x}} - \mathbf{x}\|_2 \right\}. \quad (1)$$

The term $\min_{\hat{\mathbf{x}} \in \hat{S}} \|\mathbf{x} - \hat{\mathbf{x}}\|_2$ enforces that any 3D point \mathbf{x} in the original point cloud has a matching 3D point $\hat{\mathbf{x}}$ in the reconstructed point cloud, and the term $\min_{\mathbf{x} \in S} \|\hat{\mathbf{x}} - \mathbf{x}\|_2$ enforces the matching vice versa. The max operation enforces that the distance from S to \hat{S} and the distance vice versa have to be small simultaneously. The encoder computes a representation (codeword) of each input point cloud and the decoder reconstructs the point cloud using this codeword. In our experiments, the codeword length is set as 512 in accordance with [1].

2.1. Graph-based Encoder Architecture

The encoder is a concatenation of multi-layer perceptrons and graph-based max-pooling layers. The graph is the K -nearest-neighbor graph (K-NNG) constructed from the 3D positions of the nodes in the input point cloud. In our experiments, we choose $K = 16$. First, for each single point v , we compute its local covariance matrix of size 3-by-3 and vectorize it to size 1-by-9. The local covariance of v is computed along the xyz directions using the 3D positions of the points that are one-hop neighbors of v (including v) in the K-NNG. We concatenate the matrix of point positions with size n -by-3 and the local covariances for all points of size n -by-9 into a matrix of size n -by-12 and input them to a 3-layer perceptron. The perceptron is applied in parallel to each row of the input matrix of size n -by-12. It can be viewed as a per-point function on each 3D point. The output of the perceptron is fed to two consecutive graph layers, where each layer applies max-pooling to the neighborhood of each node. More specifically, suppose the K-NN graph has adjacency matrix \mathbf{A} and the input matrix to the graph layer is \mathbf{X} . Then, the output matrix is

$$\mathbf{Y} = \mathbf{A}_{\max}(\mathbf{X})\mathbf{K}, \quad (2)$$

where \mathbf{K} is a feature mapping matrix, and the (i, j) -th entry of the matrix $\mathbf{A}_{\max}(\mathbf{X})$ is

$$(\mathbf{A}_{\max}(\mathbf{X}))_{ij} = \text{ReLU}\left(\max_{k \in \mathcal{N}(i)} x_{kj}\right). \quad (3)$$

The local max-pooling operation $\max_{k \in \mathcal{N}(i)}$ in (3) essentially computes a local signature based on the graph structure. This signature can represent the (aggregated) topology information of the local neighborhood. Through concatenations of the graph-based max-pooling layers, the network propagates the topology information into larger areas.

2.2. Folding-based Decoder Architecture

The main purpose of this paper is to design a decoder that can be used to reconstruct a point cloud from the information obtained by the encoder.

The proposed decoder uses two consecutive 3-layer perceptrons to warp a 2-dimensional grid into the shape of the input point cloud. The input codeword is obtained from the graph-based encoder. Before we feed the codeword into the decoder, we replicate it m times and concatenate the m -by-512 matrix with an m -by-2 matrix that contains the m grid points on a square centered at the origin. The result of the concatenation is a matrix of size m -by-514. The matrix is processed row-wise by a 3-layer perceptron and the output is a matrix of size m -by-3. After that, we again concatenate the replicated codewords to the m -by-3 output and feed it into a 3-layer perceptron. This output is the reconstructed point cloud. The parameter n is set as per the input point cloud size, e.g. $n = 2048$ in our experiments, which is the same as [1]. We choose m grid points in a square, so m is chosen as 2025 which is the closest square number to 2048.

Notice that the concatenation of the codewords to the 2-dimensional grids, followed by a 3-layer perceptron essentially implements a *folding* operation. To see this, denote the input 2D grid points by matrix \mathbf{U} . Each row of \mathbf{U} is a two dimensional grid point. Denote the i -th row of \mathbf{U} by \mathbf{u}_i and the codeword output from the encoder by $\boldsymbol{\theta}$. Then, after concatenation, the i -th row of the input matrix to the 3-layer perceptron is $[\mathbf{u}_i, \boldsymbol{\theta}]$. Since the 3-layer perceptron is applied in parallel to each row of the input matrix, the i -th row of the output matrix can be written as $f([\mathbf{u}_i, \boldsymbol{\theta}])$, where f indicates the function conducted by the 3-layer perceptron. This function can be viewed as a parameterized high-dimensional function with the codeword $\boldsymbol{\theta}$ being a parameter to guide the structure of the function (the folding operation). Since multilayer perceptrons are good at approximating non-linear functions, they can perform elaborate folding operations on the 2D grids. The high-dimensional codeword essentially stores the *force* that is needed to do the folding, which makes the folding operation more diverse.

The proposed decoder has two successive folding operations. The first one folds the 2D grid to 3D space, and the second one folds inside the 3D space. We show the outputs after these two folding operations in Table 1. From column C and column D in Table 1, we can see that each folding operation conducts a relatively simple operation, and the composition of the two folding operations can produce quite elaborate surface shapes. Although the first folding seems simpler than the second one, together they lead to substantial changes in the final output. More successive folding operations can be applied if more elaborate surface shapes are required.


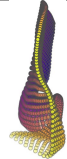

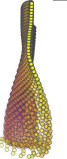


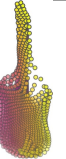


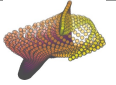
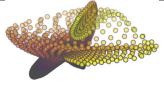
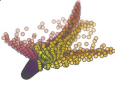
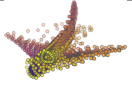
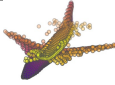
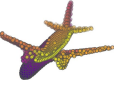
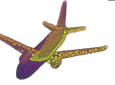
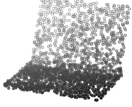
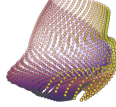
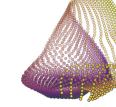
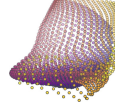
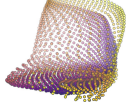
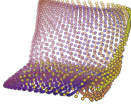
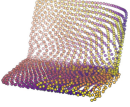
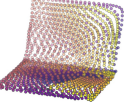

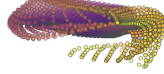
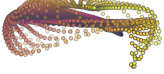
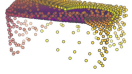
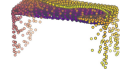
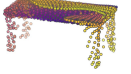
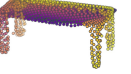
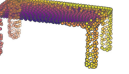

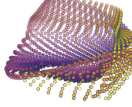
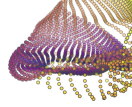
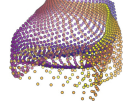
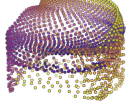
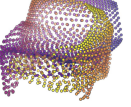
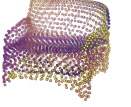
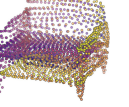

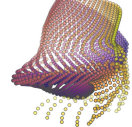
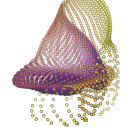
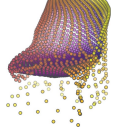
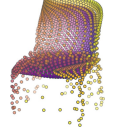
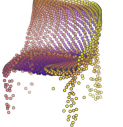
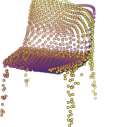
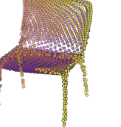
Input	5K iters	10K iters	20K iters	40K iters	100K iters	500K iters	4M iters
							
							
							
							
							
							

Table 2. Illustration of the training process. Random 2D manifolds gradually transform into the surface of point clouds.

3. Theoretical Analysis

First, we state that the graph-based encoder is permutation-invariant.

Theorem 3.1. The proposed encoder structure is permutation invariant, i.e., if the rows of the input point cloud matrix are permuted, the codeword remains unchanged.

Proof. See Supplementary Section 5. \square

Then, we state a theorem about the universality of the proposed folding-based decoder. It shows the existence of a folding-based decoder such that by changing the codeword θ , the output can be an arbitrary point cloud.

Theorem 3.2. There exists a 2-layer perceptron that can reconstruct arbitrary point clouds from a 2-dimensional grid using the *folding* operation.

More specifically, suppose the input is a matrix \mathbf{U} of size m -by-2 such that each row of \mathbf{U} is the 2D position of a point on a 2-dimensional grid of size m . Then, there exists an explicit construction of a 2-layer perceptron (with hand-crafted coefficients) such that for any arbitrary 3D point cloud matrix \mathbf{S} of size m -by-3 (where each row of \mathbf{S} is the (x, y, z) position of a point in the point cloud), there exists a codeword vector θ such that if we concatenate θ to each row of \mathbf{U} and apply the 2-layer perceptron in parallel to each row of the matrix after concatenation, we obtain the point cloud matrix \mathbf{S} from the output of the perceptron.

Proof in sketch. The full proof is in Supplementary section 6. In the proof we show the existence by explicitly constructing a 2-layer perceptron that satisfies the stated properties. The main idea is to show that in the worst case, the points in the 2D grid functions as a selective logic gate to map the 2D points in the 2D grid to the corresponding 3D points in the point cloud. \square

Notice that the above proof is just an existence-based one to show that our decoder structure is universal. It *does not* indicate what happens in reality inside the FoldingNet auto-encoder. The theoretically constructed decoder requires $3m$ hidden units while in reality, the size of the decoder that we use is much smaller. Moreover, the construction in Theorem 3.2 leads to lossless reconstruction of the point cloud, while the FoldingNet auto-encoder only achieves lossy reconstruction. However, the above theorem can indeed guarantee that the proposed decoding operation (i.e., concatenating the codewords to the 2-dimensional grid points and processing each row using a perceptron) is legitimate because in the worst case there exists a folding-based neural network with hand-crafted edge weights that can reconstruct arbitrary point clouds. In reality, a good parameterization of the proposed decoder with suitable training leads to better performance.

4. Experimental Results

4.1. Visualization of the Training Process

It is still not straightforward to tell how the decoder folds the 2D grid into the surface of a 3D point cloud. Therefore, we include an illustration on the training process to show how a random 2D manifold obtained by random folding gradually turns into a point cloud. The auto-encoder is a single FoldingNet trained using the ShapeNet part dataset [55] which contains 16 categories of the ShapeNet dataset. We train the FoldingNet using ADAM with an initial learning rate of 0.0001 and batch size of 1 for 4×10^6 iterations, which is approximately 330 epochs. The reconstructed point clouds of several models after different number of training iterations are reported in Table 2. From the training process, we can see that a random 2D manifold can be warped, cut, squeezed, stretched, and attached to form the point cloud surface in various ways.

4.2. Transfer Classification Accuracy

In this section, we show the efficiency of FoldingNet in representation learning and feature extraction from 3D point clouds. In particular, we follow the routine from [1, 53] to train a linear SVM classifier on the ModelNet dataset [54] using the codewords (latent representations) obtained from the auto-encoder, while training the auto-encoder from the ShapeNet dataset [7]. The train/test splits of the ModelNet dataset in our experiment is the same as in [39, 53]. The point-cloud-format of the ShapeNet dataset is obtained by sampling random points on the triangles from the mesh models in the dataset. It contains 57447 models from 55 categories of man-made objects. The ModelNet datasets are the same one used in [39], and the MN40/MN10 datasets respectively contain 9843/3991 models for training and 2468/909 models for testing. Each point cloud in the selected datasets contains 2048 points with (x, y, z) positions that are normalized into the 2-dimensional unit sphere as in [39].

The codewords obtained from the FoldingNet auto-encoder is of length 512, which is the same as in [1] and smaller than 7168 in [54]. When training the auto-encoder, we used ADAM with an initial learning rate of 0.0001 and batch size of 1. We train the auto-encoder for 1.6×10^7 iterations, which is approximately 278 epochs on the ShapeNet dataset. Similar to [1, 39], when training the AE, we apply random rotations to each point cloud. Unlike the random rotations in [1, 39], we apply the rotation that is one of the 24 axis-aligned rotations in the right-handed system. When training the linear SVM from the codewords obtained by the AE, we do not apply random rotations. We report our results in Table 3. The results of [8, 17, 25, 43] are according to the report in [1, 53]. Since the training of the AE and the training of the SVM are based on different datasets, the experi-

Chamfer distance v.s. classification accuracy on ModelNet40

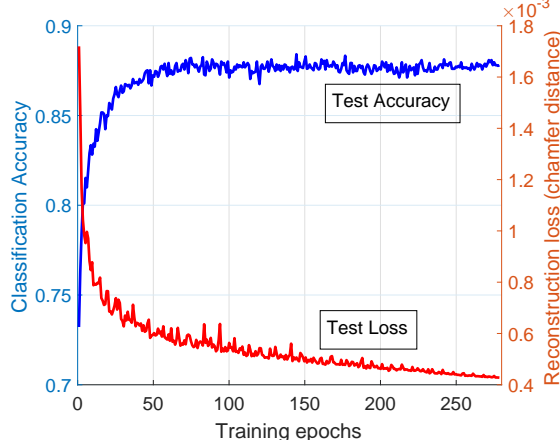


Figure 2. Linear SVM classification accuracy v.s. reconstruction loss on ModelNet40 dataset. The auto-encoder is trained using data from the ShapeNet dataset.

ment shows the transfer robustness of the FoldingNet. We also include a figure (see Figure 2) to show how the reconstruction loss decreases and the linear SVM classification accuracy increases during training. From Table 3, we can see that FoldingNet outperforms all other methods on the MN40 dataset. On the MN10 dataset, the auto-encoder proposed in [1] performs slightly better. However, the point-cloud format of the ModelNet10 dataset used in [1] is not public, so the point-cloud sampling protocol of ours may be different from the one in [1]. Therefore, it is not conclusive that the method in [1] is better than ours on MN10 dataset.

Method	MN40	MN10
SPH [25]	68.2%	79.8%
LFD [8]	75.5%	79.9%
T-L Network [17]	74.4%	-
VConv-DAE [43]	75.5%	80.5%
3D-Gan [53]	83.3%	91.0%
Latent-Gan [1]	85.7%	95.3%
FoldingNet (ours)	88.4%	94.4%

Table 3. The comparison on classification accuracy between FoldingNet and other unsupervised methods. All the methods train a linear SVM on the high-dimensional representations obtained from unsupervised training.

4.3. Effectiveness of the Folding-Based Decoder

In this section, we show that the folding-based decoder performs better in extracting features than the fully-connected decoder proposed in [1] in terms of classification accuracy and reconstruction loss. We use the ModelNet40 dataset to train two deep auto-encoders. The first auto-encoder uses the folding-based decoder that has the same structure as in Section 2.2, and the second auto-encoder uses a fully-connected three-layer perceptron as proposed

in [1]. For the fully-connected decoder, the number of inputs and number of outputs in the three layers are respectively $\{512,1024\}$, $\{1024,2048\}$, $\{2048,2048 \times 3\}$, which are the same as in [1]. The output is a 2048-by-3 matrix that contains the three-dimensional points in the output point cloud. The encoders of the two auto-encoders are both the graph-based encoder mentioned in Section 2.1. When training the AE, we used ADAM with an initial rate of 0.0001 and batch size of 1. We train the auto-encoder for 4×10^6 iterations, which is approximately 406 epochs on the ModelNet dataset.

After training, we use the encoder to process all data in the ModelNet40 dataset to obtain a codeword for each point cloud. Then, similar to Section 4.2, we train a linear SVM using these codewords and report the classification accuracy to see if the codewords are already linearly separable after encoding. The results are shown in Figure 3. During the training process, the reconstruction loss (measured in Chamfer distance) keeps decreasing, which means the reconstructed point cloud is more and more similar to the input point cloud. At the same time, the classification accuracy of the linear SVM trained on the codewords gets higher and higher accuracy, which means the codeword representations become more and more linearly separable.

From the figure, we can see that the folding decoder almost always has a higher accuracy and lower reconstruction loss. Compared to the fully-connected decoder that relies on the unnatural “1D order” of the reconstructed 3D points in 3D space, the proposed decoder relies on the folding of an inherently 2D manifold corresponding to the point cloud inside the 3D space. As we mentioned earlier, this folding operation is more natural than the fully-connected decoder. Moreover, the number of parameters in the fully-connected decoder is 1.52×10^7 , while the number of parameters in our folding decoder is 1.05×10^6 , which is about 7% of the fully-connected decoder.

One may wonder if uniformly random sampled 2D points on a plane can perform better than the 2D grid points in reconstructing point clouds. From our experimental observation, 2D grid points indeed provide reduced reconstruction loss than random points. Notice that our graph-based max-pooling encoder can be viewed as a generalized version of the max-pooling neural network PointNet [39]. The main difference is that the pooling operation in our encoder is done in local neighborhood instead of globally (see Section 2.1). In supplementary section 7, we show that the graph-based encoder architecture is better than an encoder architecture without the graph-pooling layers mentioned in Section 2.1 in terms of robustness towards random disturbance in point positions.

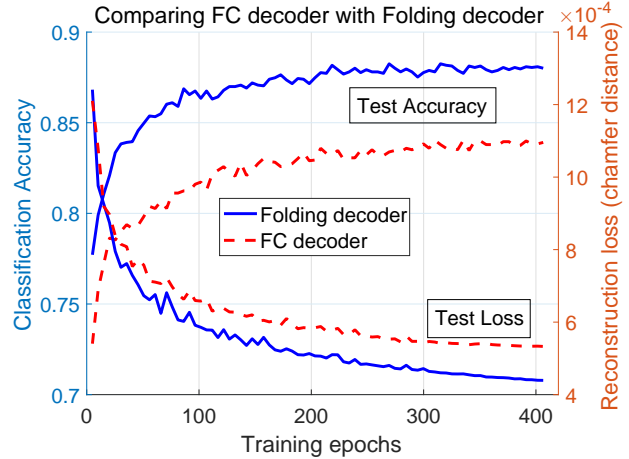


Figure 3. Comparison between the fully-connected (FC) decoder in [1] and the folding decoder on ModelNet40.

4.4. Semi-supervised Learning: What Happens when Labeled Data are Rare

One of the main motivations to study unsupervised classification problems is that the number of labeled data is usually much smaller compared to the number of unlabeled data. In Section 4.2, the experiment is very close to this setting: the number of data in the ShapeNet dataset is large, which is more than 5.74×10^4 , while the number of data in the labeled ModelNet dataset is small, which is around 1.23×10^4 . Since obtaining human-labeled data is usually hard, we would like to test how the performance of FoldingNet degrades when the number of labeled data is small. We still use the ShapeNet dataset to train the FoldingNet auto-encoder. Then, we train the linear SVM using only $a\%$ of the overall training data in the ModelNet dataset, where a can be 1, 2, 5, 7.5, 10, 15, and 20. The test data for the linear SVM are always all the data in the test data partition of the ModelNet dataset. If the codewords obtained by the auto-encoder are already linearly separable, the required number of labeled data to train a linear SVM should be small. To prove this intuitive statement, we report the experiment results in Figure 4. We can see that even if only 1% of the labeled training data are available (98 labeled training data, which is about 1~3 labeled data per class), the test accuracy is still more than 55%. When 20% of the training data are available, the test classification accuracy is already close to 85%, higher than most methods listed in Table 3.

4.5. Illustration of Point Cloud Clustering

Here, we provide an illustration of clustering 3D point clouds using the codewords obtained from FoldingNet. We use the ShapeNet dataset to train the AE and obtain codewords for the ModelNet10 dataset, as we detailed in section 4.2. Then, we use T-SNE [32] to obtain an embedding

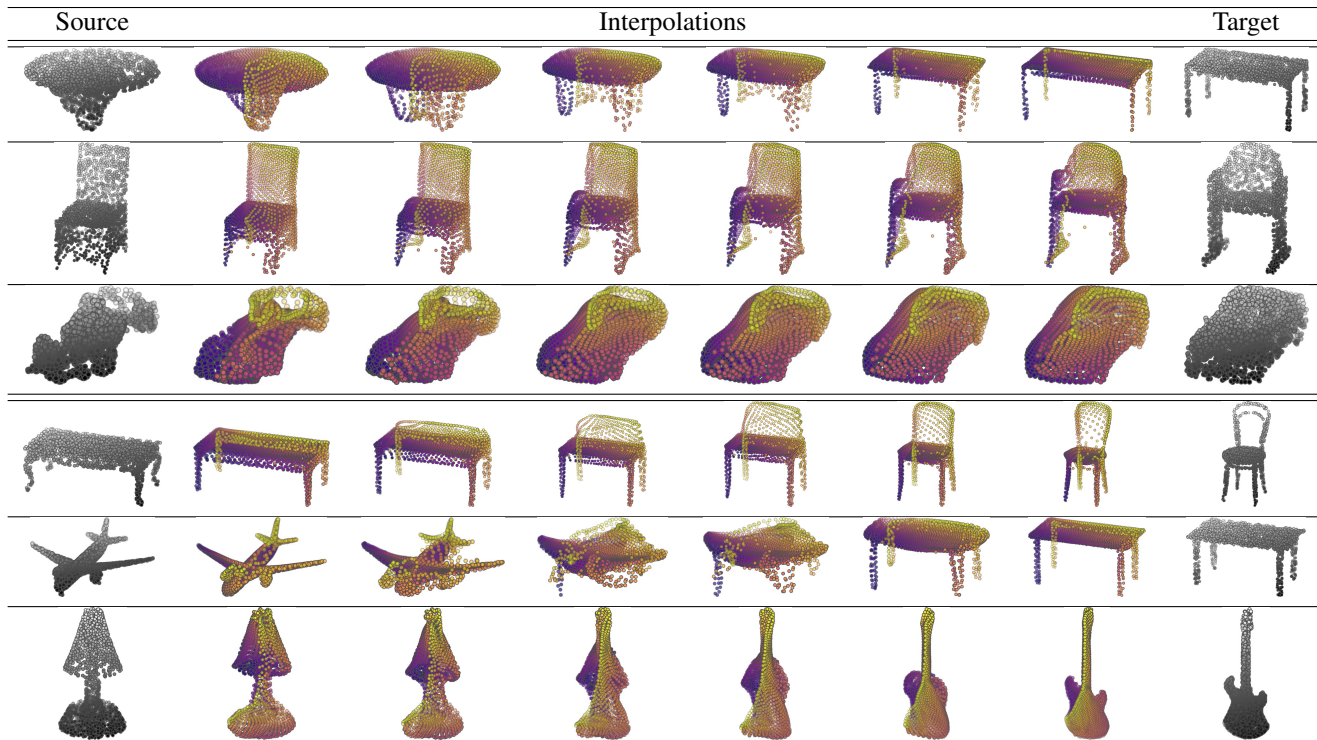


Table 4. Illustration of interpolation between different models. The first three rows contain intra-class interpolations and the last three rows contain inter-class interpolations.

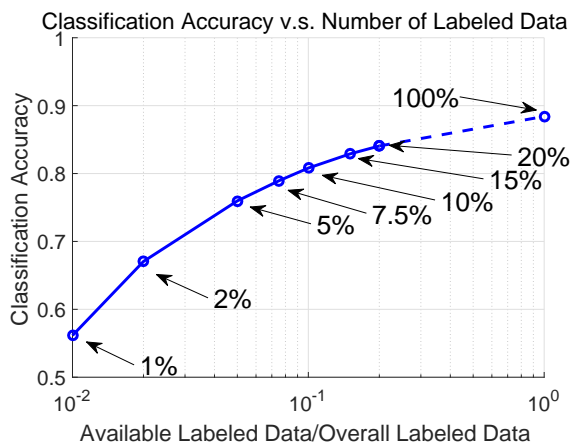


Figure 4. Linear SVM classification accuracy v.s. percentage of available labeled training data in ModelNet40 dataset.

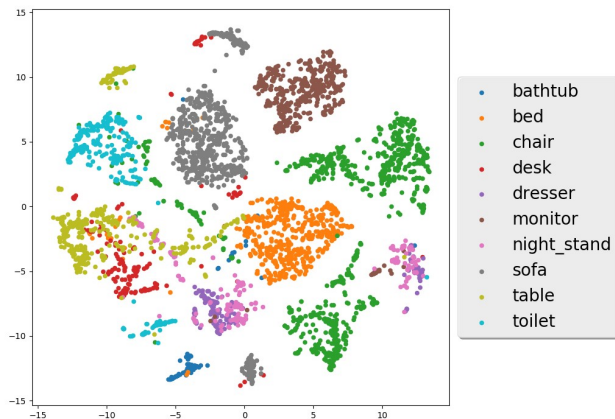


Figure 5. The T-SNE clustering visualization of the codewords obtained from FoldingNet auto-encoder.

of the high-dimensional codewords in \mathbb{R}^2 . The parameter “perplexity” in T-SNE is set as 50. We show the embedding result in Figure 5. From the figure, we see that most classes are easily separable except {dresser (violet) v.s. night stand (pink)} and {desk (red) v.s. table (yellow)}. The authors have visually checked these two pairs of classes, and found that many pairs cannot be easily distinguished even by human. In Table 5, we list the most common mistakes made in classifying the ModelNet10 dataset.

4.6. Model Interpolation

A common method to experimentally prove that the codewords have extracted the natural representations of the images is to see if the auto-encoder can automatically interpolate between two models in the dataset. In Table 4, we show both inter-class and intra-class interpolations between several classes. The auto-encoder is a single FoldingNet trained using the ShapeNet part dataset.

Item 1	Item 2	Number of mistakes
dresser	night stand	19
table	desk	15
bed	bath tub	3
night stand	table	3

Table 5. The first six types of mistakes made in the classification of ModelNet10 dataset. Their images are shown in the supplementary material.

References

- [1] P. Achlioptas, O. Diamanti, I. Mitliagkas, and L. Guibas. Latent-space gans for 3d point clouds. *34th International Conference on Machine Learning, Implicit Models Workshop*, 2017. [2](#), [3](#), [4](#), [6](#), [7](#)
- [2] J. Atwood and D. Towsley. Diffusion-convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1993–2001, 2016. [3](#)
- [3] A. Boulch, B. L. Saux, and N. Audebert. Unstructured point cloud semantic labeling using deep segmentation networks. In *Eurographics Workshop on 3D Object Retrieval*, volume 2, 2017. [3](#)
- [4] A. Brock, T. Lim, J. M. Ritchie, and N. Weston. Generative and discriminative voxel modeling with convolutional neural networks. *Advances in Neural Information Processing Systems, Workshop on 3D learning*, 2017. [3](#)
- [5] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017. [3](#)
- [6] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. Spectral networks and locally connected networks on graphs. *International Conference on Learning Representations (ICLR)*, 2014. [3](#)
- [7] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, et al. Shapenet: An information-rich 3d model repository. *CoRR*, 2015. [2](#), [6](#)
- [8] D.-Y. Chen, X.-P. Tian, Y.-T. Shen, and M. Ouhyoung. On visual similarity based 3D model retrieval. In *Computer graphics forum*, volume 22, pages 223–232. Wiley Online Library, 2003. [6](#)
- [9] X. Chen, K. Kundu, Y. Zhu, A. G. Berneshawi, H. Ma, S. Fidler, and R. Urtasun. 3D object proposals for accurate object class detection. In *Advances in Neural Information Processing Systems*, pages 424–432, 2015. [3](#)
- [10] S. Christoph Stein, M. Schoeler, J. Papon, and F. Worgotter. Object partitioning using local convexity. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 304–311, 2014. [3](#)
- [11] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner. Scannet: Richly-annotated 3D reconstructions of indoor scenes. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017. [3](#)
- [12] M. Defferrard, X. Bresson, and P. Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852, 2016. [3](#)
- [13] D. Dohan, B. Matejek, and T. Funkhouser. Learning hierarchical semantic segmentations of LIDAR data. In *International Conference on 3D Vision (3DV)*, pages 273–281. IEEE, 2015. [3](#)
- [14] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pages 2224–2232, 2015. [3](#)
- [15] M. Edwards and X. Xie. Graph based convolutional neural network. *CoRR*, 2016. [3](#)
- [16] A. Geiger and C. Wang. Joint 3D object and layout inference from a single RGB-D image. In *German Conference on Pattern Recognition*, pages 183–195. Springer, 2015. [3](#)
- [17] R. Girdhar, D. F. Fouhey, M. Rodriguez, and A. Gupta. Learning a predictable and generative vector representation for objects. In *European Conference on Computer Vision*, pages 484–499. Springer, 2016. [6](#)
- [18] A. Golovinskiy, V. G. Kim, and T. Funkhouser. Shape-based recognition of 3d point clouds in urban environments. In *12th International Conference on Computer Vision*, pages 2154–2161. IEEE, 2009. [3](#)
- [19] T. Hackel, N. Savinov, L. Ladicky, J. D. Wegner, K. Schindler, and M. Pollefeys. Semantic3D. net: A new large-scale point cloud classification benchmark. *ISPRS Annals*, 2017. [3](#)
- [20] T. Hackel, J. D. Wegner, and K. Schindler. Contour detection in unstructured 3d point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1610–1618, 2016. [3](#)
- [21] T. Hackel, J. D. Wegner, and K. Schindler. Fast semantic segmentation of 3D point clouds with strongly varying density. *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences*, 3(3), 2016. [3](#)
- [22] Y. Hechtlinger, P. Chakravarti, and J. Qin. A generalization of convolutional neural networks to graph-structured data. *arXiv preprint arXiv:1704.08165*, 2017. [3](#)
- [23] M. Henaff, J. Bruna, and Y. LeCun. Deep convolutional networks on graph-structured data. *arXiv:1506.05163*, 2015. [3](#)
- [24] J. Huang and S. You. Point cloud labeling using 3D convolutional neural network. In *23rd International Conference on Pattern Recognition (ICPR)*, pages 2670–2675. IEEE, 2016. [3](#)
- [25] M. Kazhdan, T. Funkhouser, and S. Rusinkiewicz. Rotation invariant spherical harmonic representation of 3d shape descriptors. In *Symposium on geometry processing*, volume 6, pages 156–164, 2003. [6](#)
- [26] B.-S. Kim, P. Kohli, and S. Savarese. 3D scene understanding by voxel-CRF. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1425–1432, 2013. [3](#)
- [27] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *International Conference on Learning Representations (ICLR)*, 2017. [3](#)
- [28] R. Klokov and V. Lempitsky. Escape from cells: Deep Kd-networks for the recognition of 3D point cloud models. *In-*

- ternational Conference on Computer Vision (ICCV), 2017. 3
- [29] K. Lai, L. Bo, and D. Fox. Unsupervised feature learning for 3D scene labeling. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3050–3057. IEEE, 2014. 3
- [30] R. Levie, F. Monti, X. Bresson, and M. M. Bronstein. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. *arXiv:1705.07664*, 2017. 3
- [31] Y. Li, S. Pirk, H. Su, C. R. Qi, and L. J. Guibas. Fpnn: Field probing neural networks for 3D data. In *Advances in Neural Information Processing Systems*, pages 307–315, 2016. 3
- [32] L. v. d. Maaten and G. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008. 7
- [33] J. Masci, D. Boscaini, M. Bronstein, and P. Vandergheynst. Geodesic convolutional neural networks on riemannian manifolds. In *Proceedings of the IEEE International conference on computer vision workshops*, pages 37–45, 2015. 3
- [34] D. Maturana and S. Scherer. 3D convolutional neural networks for landing zone detection from LIDAR. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3471–3478. IEEE, 2015. 3
- [35] D. Maturana and S. Scherer. Voxnet: A 3D convolutional neural network for real-time object recognition. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 922–928. IEEE, 2015. 3
- [36] F. Monti, D. Boscaini, J. Masci, E. Rodolà, J. Svoboda, and M. M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model CNNs. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017. 3
- [37] M. Niepert, M. Ahmed, and K. Kutzkov. Learning convolutional neural networks for graphs. In *International Conference on Machine Learning*, pages 2014–2023, 2016. 3
- [38] G. Pang and U. Neumann. Fast and robust multi-view 3D object recognition in point clouds. In *International Conference on 3D Vision (3DV)*, pages 171–179. IEEE, 2015. 3
- [39] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3D classification and segmentation. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017. 2, 3, 6, 7, 12
- [40] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in Neural Information Processing Systems*, 2017. 3
- [41] S. Ravanbakhsh, J. Schneider, and B. Póczos. Deep learning with sets and point clouds. *International Conference on Learning Representations (ICLR), workshop track*, 2017. 3
- [42] G. Riegler, A. O. Ulusoy, and A. Geiger. Octnet: Learning deep 3D representations at high resolutions. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017. 3
- [43] A. Sharma, O. Grau, and M. Fritz. Vconv-DAE: Deep volumetric shape learning without object labels. In *Computer Vision-ECCV 2016 Workshops*, pages 236–250. Springer, 2016. 6
- [44] M. Simonovsky and N. Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017. 3
- [45] R. Socher, B. Huval, B. Bath, C. D. Manning, and A. Y. Ng. Convolutional-recursive deep learning for 3D object classification. In *Advances in Neural Information Processing Systems*, pages 656–664, 2012. 3
- [46] S. Song and J. Xiao. Deep sliding shapes for amodal 3D object detection in RGB-D images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 808–816, 2016. 3
- [47] J. Strom, A. Richardson, and E. Olson. Graph-based segmentation for colored 3d laser point clouds. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2131–2136. IEEE, 2010. 3
- [48] D. Thanou, P. A. Chou, and P. Frossard. Graph-based compression of dynamic 3d point cloud sequences. *IEEE Transactions on Image Processing*, 25(4):1765–1778, 2016. 3
- [49] J.-C. Vialatte, V. Gripon, and G. Mercier. Generalizing the convolution operator to extend cnns to irregular domains. *arXiv preprint arXiv:1606.01166*, 2016. 3
- [50] E. Wahl, U. Hillenbrand, and G. Hirzinger. Surflet-pair-relation histograms: a statistical 3D-shape representation for rapid classification. In *Fourth International Conference on 3-D Digital Imaging and Modeling*, pages 474–481. IEEE, 2003. 3
- [51] P. Wang, Y. Gan, P. Shui, F. Yu, Y. Zhang, S. Chen, and Z. Sun. 3d shape segmentation via shape fully convolutional networks. *Computers & Graphics*, 2017. 3
- [52] Y. Wang, R. Ji, and S.-F. Chang. Label propagation from ImageNet to 3D point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3135–3142, 2013. 3
- [53] J. Wu, C. Zhang, T. Xue, B. Freeman, and J. Tenenbaum. Learning a probabilistic latent space of object shapes via 3D generative-adversarial modeling. In *Advances in Neural Information Processing Systems*, pages 82–90, 2016. 2, 3, 6
- [54] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3D Shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1912–1920, 2015. 1, 2, 3, 6
- [55] L. Yi, V. G. Kim, D. Ceylan, I. Shen, M. Yan, H. Su, A. Lu, Q. Huang, A. Sheffer, L. Guibas, et al. A scalable active framework for region annotation in 3d shape collections. *ACM Transactions on Graphics (TOG)*, 35(6):210, 2016. 3, 6
- [56] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Póczos, R. Salakhutdinov, and A. Smola. Deep sets. *Advances in Neural Information Processing Systems*, 2017. 3
- [57] Q. Zhang, X. Song, X. Shao, H. Zhao, and R. Shibasaki. Unsupervised 3D category discovery and point labeling from a large urban environment. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2685–2692. IEEE, 2013. 3

5. Supplementary: Proof of Theorem 3.1

Denote the input n -by-12 matrix by \mathbf{L} . Denote by $\boldsymbol{\theta}$ the codeword obtained by the encoder. Now we prove if the input is \mathbf{PL} where \mathbf{P} is an n -by- n permutation matrix, the codeword obtained from the encoder is still $\boldsymbol{\theta}$.

The first part of the encoder is a per-point function, i.e., the 3-layer perceptron is applied to each row of the input matrix \mathbf{L} . Denote the function by f_1 . Then, it is obvious that $f_1(\mathbf{PL}) = \mathbf{P}f_1(\mathbf{L})$. The second part computes (2). Now we prove that for (2),

$$\mathbf{PY} = \mathbf{A}_{\max}(\mathbf{PX})\mathbf{K}. \quad (4)$$

Since $\mathbf{Y} = \mathbf{A}_{\max}(\mathbf{X})\mathbf{K}$, we only need to prove

$$\mathbf{A}_{\max}(\mathbf{PX}) = \mathbf{PA}_{\max}(\mathbf{X}). \quad (5)$$

Suppose the permutation operation \mathbf{P} makes the i -th row of \mathbf{PX} equal to $\mathbf{x}_{\pi(i)}$, where $\pi(\cdot)$ is a permutation function on the set of row indexes $\{1, 2, \dots, n\}$. Then, from (3), the (i, j) -th entry of the matrix $\mathbf{A}_{\max}(\mathbf{PX})$ is

$$(\mathbf{A}_{\max}(\mathbf{PX}))_{ij} = \text{ReLU}\left(\max_{k \in \mathcal{N}(\pi(i))} x_{kj}\right). \quad (6)$$

In the meantime, the $(\pi(i), j)$ -th entry of $\mathbf{A}_{\max}(\mathbf{X})$ is

$$(\mathbf{A}_{\max}(\mathbf{X}))_{\pi(i)j} = \text{ReLU}\left(\max_{k \in \mathcal{N}(\pi(i))} x_{kj}\right). \quad (7)$$

Since the right hand side of (6) and (7) are the same, we know that the matrix $\mathbf{A}_{\max}(\mathbf{PX})$ can be obtained by changing the i -th row of $\mathbf{A}_{\max}(\mathbf{X})$ to the $\pi(i)$ -th row, which means $\mathbf{A}_{\max}(\mathbf{PX}) = \mathbf{PA}_{\max}(\mathbf{X})$. Thus, we have proved that for the second part of the encoder, permuting the input rows is equivalent to permuting the output rows, i.e., (4) holds.

Therefore, if we permute the input to the encoder, the output of the graph layers also permute. Then, we apply global max-pooling to the output of the graph layers. It is obvious that the result remains the same if the rows of the input to the global max-pooling layer (or the output of the graph layers) permutes. The conclusion of Theorem 3.1 is hence proved.

6. Supplementary: Proof of Theorem 3.2

We prove the existence-based Theorem 3.2 by explicitly constructing a 2-layer perceptron and a codeword vector $\boldsymbol{\theta}$ that satisfy the stated properties.

The codeword is simply chosen as the vectorized form of the point cloud matrix \mathbf{S} . In particular, For a matrix \mathbf{S} of size m -by-3, if $\mathbf{S} = [s_{jk}], j = 1, 2, \dots, m$ and $k = 1, 2, 3$, the codeword vector $\boldsymbol{\theta}$ is chosen to be $\boldsymbol{\theta} = [s_{11}, s_{12}, s_{13}, s_{21}, s_{22}, s_{23}, \dots, s_{m1}, s_{m2}, s_{m3}]$. Then, the i -th row after concatenation is $\mathbf{v}_i =$

$[x_i, y_i, s_{11}, s_{12}, s_{13}, s_{21}, s_{22}, s_{23}, \dots, s_{m1}, s_{m2}, s_{m3}]$, where $[x_i, y_i]$ is the position of the i -th 2D grid point. Suppose the 2D grid points have an interval 2δ , i.e., the distance between any two points in the 2D grid is at least 2δ . Further assume these m grid points can all be written as $[x_i, y_i] = [(2\beta_i + 1)\delta, (2\gamma_i + 1)\delta]$, where β_i and γ_i are two integers whose absolute values are smaller than a positive constant M . One example of a set of 4-by-4 grid points is

$$\begin{Bmatrix} [-3\delta, -3\delta], & [-3\delta, -1\delta], & [-3\delta, 1\delta], & [-3\delta, 3\delta], \\ [-1\delta, -3\delta], & [-1\delta, -1\delta], & [-1\delta, 1\delta], & [-1\delta, 3\delta], \\ [1\delta, -3\delta], & [1\delta, -1\delta], & [1\delta, 1\delta], & [1\delta, 3\delta], \\ [3\delta, -3\delta], & [3\delta, -1\delta], & [3\delta, 1\delta], & [3\delta, 3\delta]. \end{Bmatrix} \quad (8)$$

In this case, the choice of M is 4. Also assume that the output point cloud is bounded inside 3-dimensional box of length 2 centered at the origin, i.e., $|s_{ij}| \leq 1$.

Now, we construct a 2-layer perceptron f that takes the rows \mathbf{v}_i as inputs and provides the outputs $f(\mathbf{v}_i) = [s_{i1}, s_{i2}, s_{i3}]$, for $i = 1, 2, \dots, m$. The input layer takes the vector input \mathbf{v}_i which has $3m + 2$ scalars. The hidden layer has $3m$ neurons. The output layer provides three scalar outputs $[s_{i1}, s_{i2}, s_{i3}]$. The $3m$ neurons in the hidden layer are partitioned into m groups of 3 neurons. The k -th neuron ($k = 1, 2, 3$) in the j -th group ($j = 1, 2, 3, \dots, m$) is only connected to three inputs x_i, y_i and $[s_{j,k}]$, and it computes a linear combination of x_i, y_i and $s_{j,k}$ with weights

$$\begin{aligned} \alpha_{j1} &= u^2 x_j, \\ \alpha_{j2} &= u y_j, \\ \alpha_{j3} &= 1, \end{aligned} \quad (9)$$

and bias

$$b = -u^2 x_j^2 - u y_j^2 \quad (10)$$

where u is a positive constant to be specified later. Suppose the linear combination output is $y_{j,k}$. The linear combination is followed by a nonlinear activation function¹ that computes the following

$$z_{j,k} = \begin{cases} y_{j,k}, & \text{if } |y_{j,k}| < c, \\ 0, & \text{if } |y_{j,k}| \geq c, \end{cases} \quad (11)$$

where c is a constant to be specified later. The outputs of the activation functions are linearly combined to produce the final output. There are three neurons in the output layer. The k -th neuron ($k=1,2,3$) computes

$$w_k = \sum_{j=1}^m z_{j,k}. \quad (12)$$

¹It is not hard to prove that this function can be obtained by concatenating ReLU functions with appropriate bias terms. We specifically avoid using the ReLU function in order not to hinder the main intuition. In all of our experiments, we use ReLU activation functions.

We assume the parameters (δ, u, c, M) satisfy

$$u > 0, c > 0, \delta > 0, M > 0, \quad (13)$$

$$u\delta^2 > c + 1, \quad (14)$$

$$u > 8M^2 + 4M + 1, \quad (15)$$

$$c > 1. \quad (16)$$

Now we prove that for this perceptron, the final output $[w_1, w_2, w_3]$ is indeed $[s_{i1}, s_{i2}, s_{i3}]$ when the input to the perceptron is \mathbf{v}_i . For the i -th input $\mathbf{v}_i = [x_i, y_i, s_{11}, s_{12}, s_{13}, s_{21}, s_{22}, s_{23}, \dots, s_{m1}, s_{m2}, s_{m3}]$, the k -th neuron in the j -th group in the hidden layer computes the following linear combination

$$\begin{aligned} y_{j,k} &= \alpha_{j1}x_i + \alpha_{j2}y_i + \alpha_{j3}s_{j,k} + b \\ &= u^2x_jx_i + uy_jy_i + s_{j,k} - u^2x_j^2 - uy_j^2 \\ &= u^2x_j(x_i - x_j) + uy_j(y_i - y_j) + s_{j,k}. \end{aligned} \quad (17)$$

Notice that we have assumed $[x_i, y_i] = [(2\beta_i + 1)\delta, (2\gamma_i + 1)\delta]$, $\forall i$. So we have

$$\begin{aligned} y_{j,k} &= u^2x_j(x_i - x_j) + uy_j(y_i - y_j) + s_{j,k} \\ &= 2u^2\delta^2(2\beta_j + 1)(\beta_i - \beta_j) + 2u\delta^2(2\gamma_j + 1)(\gamma_i - \gamma_j) + s_{j,k} \\ &= u^2\delta^2m_1 + u\delta^2m_2 + s_{j,k}, \end{aligned} \quad (18)$$

where the two integer constants $m_1 = 2(2\beta_j + 1)(\beta_i - \beta_j)$ and $m_2 = 2(2\gamma_j + 1)(\gamma_i - \gamma_j)$, and $m_1 = 0$ only if $x_i = x_j$ and $m_2 = 0$ only if $y_i = y_j$. Since the absolute values of $\beta_i, \beta_j, \gamma_i$ and γ_j are all smaller than M , we have

$$|m_1| \leq 2|2\beta_j + 1| \cdot |\beta_i - \beta_j| < 2(2M + 1) \cdot 2M = 8M^2 + 4M. \quad (19)$$

Similarly, we have

$$|m_2| \leq 2|2\gamma_j + 1| \cdot |\gamma_i - \gamma_j| < 2(2M + 1) \cdot 2M = 8M^2 + 4M. \quad (20)$$

Now we consider 3 possible cases:

- $|m_1| \geq 1$: In this case,

$$\begin{aligned} |y_{j,k}| &= |u^2\delta^2m_1 + u\delta^2m_2 + s_{j,k}| \\ &> u^2\delta^2|m_1| - u\delta^2|m_2| - |s_{j,k}| \\ &> u^2\delta^2 - u\delta^2(8M^2 + 4M) - 1 \\ &= u\delta^2[u - (8M^2 + 4M)] - 1 \\ &\stackrel{(a)}{>} (c + 1) \cdot 1 - 1 = c, \end{aligned} \quad (21)$$

where step (a) follows from the assumption (14).

- $m_1 = 0$ but $|m_2| \geq 1$: In this case,

$$\begin{aligned} |y_{j,k}| &= |u\delta^2m_2 + s_{j,k}| \\ &\geq u\delta^2|m_2| - |s_{j,k}| \\ &\geq u\delta^2 \stackrel{(a)}{\geq} c + 1 > c, \end{aligned} \quad (22)$$

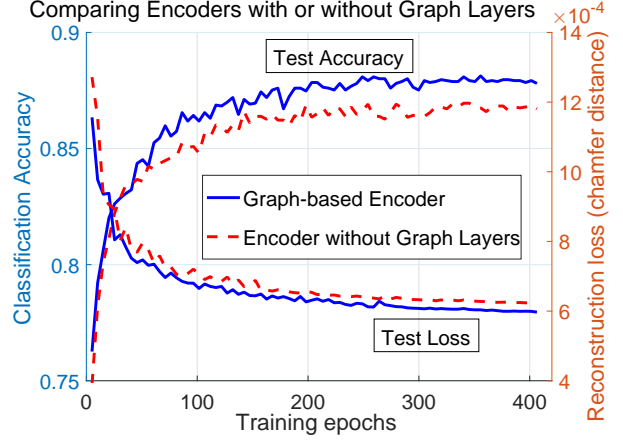


Figure 6. Comparison between the graph-based encoder in Section 2.1 and the encoder from which the graph-based max-pooling layers are removed. The encoder with no graph-based layers is similar to the one proposed in [39] which is for a different goal (supervised learning).

where step (a) follows from assumption (15).

- $m_1 = m_2 = 0$. In this case,

$$|y_{j,k}| = |s_{j,k}| \leq 1 \stackrel{(a)}{<} c, \quad (23)$$

where step (a) follows from assumption (16).

Notice that the first two cases are equivalent to $i \neq j$ and the last case is equivalent to $i = j$. Thus, from (11), we have

$$z_{j,k} = \begin{cases} s_{j,k}, & \text{if } j = i, \\ 0, & \text{if } j \neq i. \end{cases} \quad (24)$$

Thus, from (12), the final output is

$$w_k = \sum_{j=1}^m z_{j,k} = s_{i,k}, k = 1, 2, 3, \quad (25)$$

which means the output is indeed $[s_{i,1}, s_{i,2}, s_{i,3}]$ when the input is \mathbf{v}_i . This concludes the proof.

7. Supplementary: Robustness of the graph-based encoder

Here, we use one experiment to show that the graph-pooling layers are useful in maintaining the good performance of the FoldingNet when the data is subject to random noise. The following experiment compares FoldingNet with a deep auto-encoder that has the same folding-based decoder architecture but a different encoder architecture in which the graph-based max-pooling layers are removed. The setting of the experiment is the same as in Section 4.3 except that 5 percents of the points in each point cloud in

the ModelNet40 dataset are randomly shifted to other positions (but still within the bounding box of the original point cloud). We use this noisy data to see how the performances degrade for the graph-based encoder and the encoder without graph-based max-pooling layers. The results are reported in Figure 6. We can see that when the graph-based max-pooling layers are removed, the performance degrades by approximately 2 percents when noise is injected to the dataset. However, the classification accuracy of FoldingNet does not change much (when compared with Figure 3 in Section 4.3). Thus, it can be seen that the graph-based encoder can make FoldingNet more robust.

8. Supplementary: More Details on the Linear SVM Experiment on ModelNet10

The classification accuracy obtained in Section 4.2 on MN10 dataset is 94.4%. We stated in Section 4.5 that many pairs which are wrongly classified are actually hard to distinguish even by human. In the table on next page, we list all the incorrectly classified models and their point cloud representations. A phrase like “table → desk” means the point cloud has label “table” but it is wrongly classified as “desk” by the linear SVM.

