6. 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),

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

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

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

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 $\{1, 2, \ldots, n\}$. Then, from (3), the (i,j)-th entry of the matrix $\mathbf{A}_{\max}(\mathbf{PX})$ is

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

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

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

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) permute. The conclusion of Theorem 3.1 is hence proved.

7. 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 $\mathbf{S} = [s_{jk}], j = 1, 2, \ldots m$ and $k = 1, 2, 3$, the codeword vector $\boldsymbol{\theta}$ is chosen to be $\boldsymbol{\theta} = [s_{11}, s_{12}, s_{13}, s_{21}, s_{22}, s_{23}, \ldots, s_{m1}, s_{m2}, s_{m3}]$. Then, the i-th row after concatenation is $\mathbf{v}_i = [x_i, y_i, s_{11}, s_{12}, s_{13}, s_{21}, s_{22}, s_{23}, \ldots, s_{m1}, s_{m2}, s_{m3}]$, 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

$$\begin{array}{cccc} \{[-3\delta, -3\delta], & [-3\delta, -1\delta], & [-3\delta, 1\delta], & [-3\delta, 3\delta], \\ [-1\delta, -3\delta], & [-1\delta, -1\delta], & [-1\delta, 1\delta], & [-1\delta, 3\delta], \\ [1\delta, -3\delta], & [1\delta, -1\delta], & [1\delta, 1\delta], & [1\delta, 3\delta], \\ [3\delta, -3\delta], & [3\delta, -1\delta], & [3\delta, 1\delta], & [3\delta, 3\delta]\}. \end{array} \tag{8}$$

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 $i = 1, 2, \ldots, m$. The input layer takes the vector intput $\mathbf{v}_i$ 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 ($j = 1, 2, 3, \ldots, m$) 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

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

and bias

$$b = -u^2 x_j^2 - u y_j^2 \tag{10}$$

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 function[1] that computes the following

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

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

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

[1]It 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

$$u > 0, c > 0, \delta > 0, M > 0, \tag{13}$$
$$u\delta^2 > c + 1, \tag{14}$$
$$u > 8M^2 + 4M + 1, \tag{15}$$
$$c > 1. \tag{16}$$

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 $\mathbf{v}_i = [x_i, y_i, s_{11}, s_{12}, s_{13}, s_{21}, s_{22}, s_{23}, \ldots, s_{m1}, s_{m2}, s_{m3}]$, the k-th neuron in the j-th group in the hidden layer computes the following linear combination

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

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

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

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 β_i, β_j, γ_i and γ_j are all smaller than M, we have

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

Similarly, we have

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

Now we consider 3 possible cases:

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

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

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

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

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

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

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

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

  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

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

Thus, from (12), the final output is

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

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.

8. Supplementary: Decoder Variations

The current decoder design has two consecutive folding operations that apply on a 2D grid. Therefore, one may wonder if the performance of FoldingNet can be improved if we utilize (1) more folding operations or (2) the same number of folding operations on regular grids of different dimensions. In this section, we report the results for these different settings. The experimental settings are the same with Section 4.6. The experiment results are shown in Table 6. As one can see from line 1 and line 2, increasing the number of folding operations does not significantly increase the performance. Comparing line 1 and line 3, one can see that a 2D grid is better than a 1D grid for both classification and reconstruction. From line 1 and line 4, one can see that a 3D grid only brings a marginal improvement. As we discussed in the introduction, this is because the intrinsic dimensionality of data in the ShapeNet and ModelNet datasets is 2, as they are sampled from object surfaces. If point clouds are intrinsically volumetric, we believe using a 3D grid in the decoder is more suitable. In addition, we also tried to generate the fixed grid by uniformly random

Grid Setting	#Folds	Test Cls. Acc.	Test Loss
regular 2D	2	88.25%	0.0296
regular 2D	3	88.41%	0.0290
regular 1D	2	86.71%	0.0355
regular 3D	2	88.41%	0.0284
uniform 2D	2	87.12%	0.0321

Table 6. Comparison between different FoldingNet decoders. "Uniform": the grid is uniformly random sampled. "Regular": the grid is regularly sampled with fixed spacings.

sampling in the square. However, it leads to slightly worse results. We believe it is caused by the local density variation introduced by the random sampling.

9. Supplementary: Folding by Deconvolution

The folding operation in Definition 1 is essentially a per-point 2D-to-3D function from a 2D grid to a 3D surface. A natural question to ask is whether introducing explicit correlations in the functions imposed on neighboring grid points can help improve the performance. We noted that there is a closely related work on reconstructing 3D point sets using side information from images [17]. The point reconstruction network in [17] uses deconvolution to fuse information on the regular grid structure imposed by the image, which is similar to the idea above. Here, we compare a deconvolution network with FoldingNet on the reconstruction performance. The feature sizes of the deconvolution network (C×H×W) are $512\times1\times1\rightarrow256\times3\times3\rightarrow128\times5\times5\rightarrow64\times15\times15\rightarrow3\times45\times45$ with kernel sizes 3, 3, 5, 5. The comparison is shown in Table 7. We conjecture that deconvolution goes beyond point-wise operations, thus imposes a stronger constraint on the smoothness of the reconstructed surface. Thus, its reconstruction is worse (although with comparable classification accuracy). On the other hand, the use of grids with point-wise MLP in FoldingNet only impose an implicit constraint, thus leading to better reconstructions.

	Cl. Acc.	Tst. Loss	# Params.
FoldingNet	88.41%	0.0296	1.0×10^6
Deconv	88.86%	0.0319	1.7×10^6

Table 7. Comparison of two different implementations of the folding operation.

10. 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 4.6 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

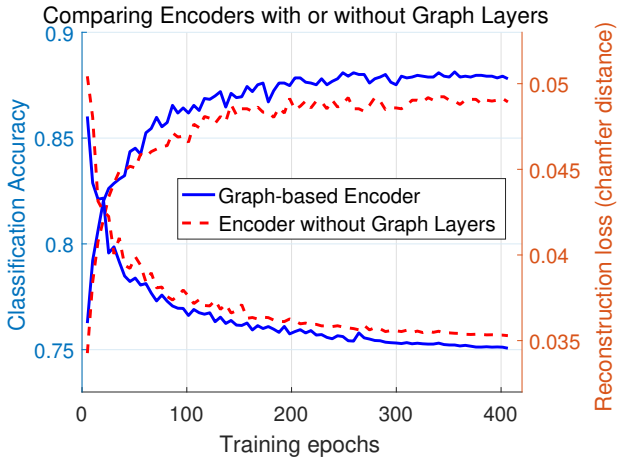
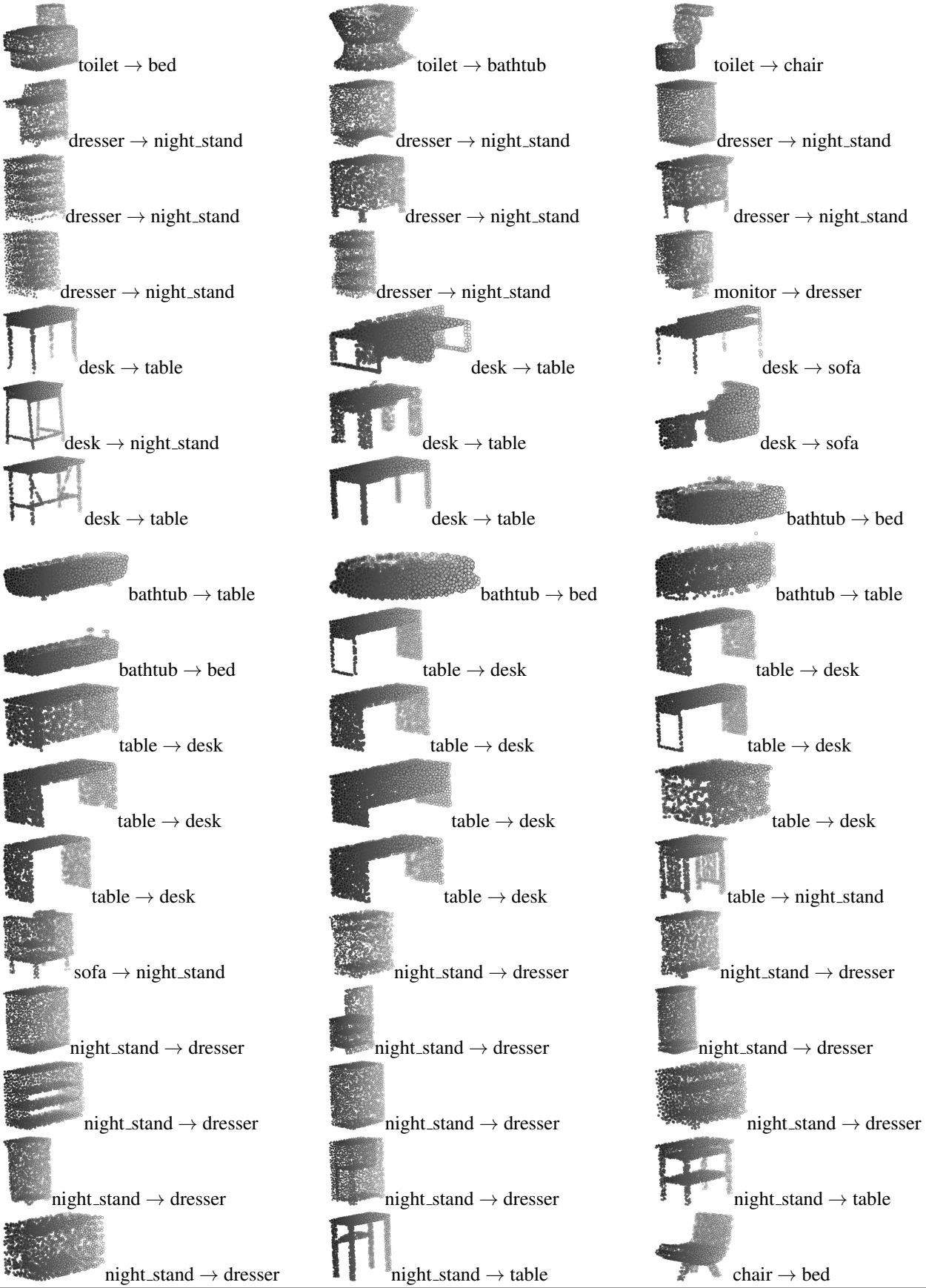


Figure 6. Comparison between the graph-based encoder in Section 2.1 and the encoder from which the graph-based max-pooling layers are removed. The encoder with no graph-based layers is similar to the one proposed in [41] which is for a different goal (supervised learning).

max-pooling layers are removed, the performance degrades by approximately 2 percents when noise is injected into the dataset. However, the classification accuracy of FoldingNet does not change much (when compared with Figure 5 in Section 4.6). Thus, it can be seen that the graph-based encoder can make FoldingNet more robust.

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

The classification accuracy obtained in Section 4.4 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 a human. In the table on the 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.


toilet → bed
toilet → bathtub
toilet → chair
dresser → night_stand
dresser → night_stand
dresser → night_stand
dresser → night_stand
dresser → night_stand
dresser → night_stand
dresser → night_stand
dresser → night_stand
monitor → dresser
desk → table
desk → table
desk → sofa
desk → night_stand
desk → table
desk → sofa
desk → table
desk → table
bathtub → bed
bathtub → table
bathtub → bed
bathtub → table
bathtub → bed
table → desk
table → desk
table → desk
table → desk
table → desk
table → desk
table → desk
table → desk
table → desk
table → desk
table → desk
table → night_stand
sofa → night_stand
night_stand → dresser
night_stand → dresser
night_stand → dresser
night_stand → dresser
night_stand → dresser
night_stand → dresser
night_stand → dresser
night_stand → dresser
night_stand → dresser
night_stand → dresser
night_stand → table
night_stand → dresser
night_stand → table
chair → bed