# TearingNet: Point Cloud Autoencoder to Learn Topology-Friendly Representations

Jiahao Pang  Duanshun Li*  Dong Tian
InterDigital, Princeton, NJ, USA
jiahao.pang@interdigital.com, duanshun@ualberta.ca, dong.tian@interdigital.com

## Abstract

*Topology matters. Despite the recent success of point cloud processing with geometric deep learning, it remains arduous to capture the complex topologies of point cloud data with a learning model. Given a point cloud dataset containing objects with various genera, or scenes with multiple objects, we propose an autoencoder, TearingNet, which tackles the challenging task of representing the point clouds using a fixed-length descriptor. Unlike existing works directly deforming predefined primitives of genus zero (e.g., a 2D square patch) to an object-level point cloud, our TearingNet is characterized by a proposed Tearing network module and a Folding network module interacting with each other iteratively. Particularly, the Tearing network module learns the point cloud topology explicitly. By breaking the edges of a primitive graph, it tears the graph into patches or with holes to emulate the topology of a target point cloud, leading to faithful reconstructions. Experimentation shows the superiority of our proposal in terms of reconstructing point clouds as well as generating more topology-friendly representations than benchmarks.*

## 1. Introduction

Based on a point cloud sampled from the surface of an object (or a scene), humans are able to perceive its underlying shape. Via properly capturing the *topology* behind the point set, human understanding is robust to variations in scales and viewpoints. Intuitively, topology reflects how the points are put together to form an object. Moreover, topology is an intrinsic property of Riemannian manifolds that are usually used to model 3D shapes in geometric learning [2, 23]. Hence, it is important to seek topology-aware representations for point clouds in machine learning.

As an unsupervised learning architecture, *autoencoder* (AE) [25] is popularly investigated to learn latent representations with unlabeled point clouds. It tries to approximate an *identity* function that is non-trivially constrained by outputting a compact representation from its encoder network. The decoder network attempts to reconstruct the point cloud from the compact representation. The compact representation typically takes the shape of a fixed-length codeword which characterizes geometric properties of point clouds. Therefore, it not only preserves the ability for reconstruction [6] but is also valuable for high-level tasks such as classification [40, 42, 11].

With ample topological structures in the real world, unfortunately, it is non-trivial to produce topology-friendly representations that count for object point clouds with holes (*i.e.*, varying genera) or scene point clouds with a varying

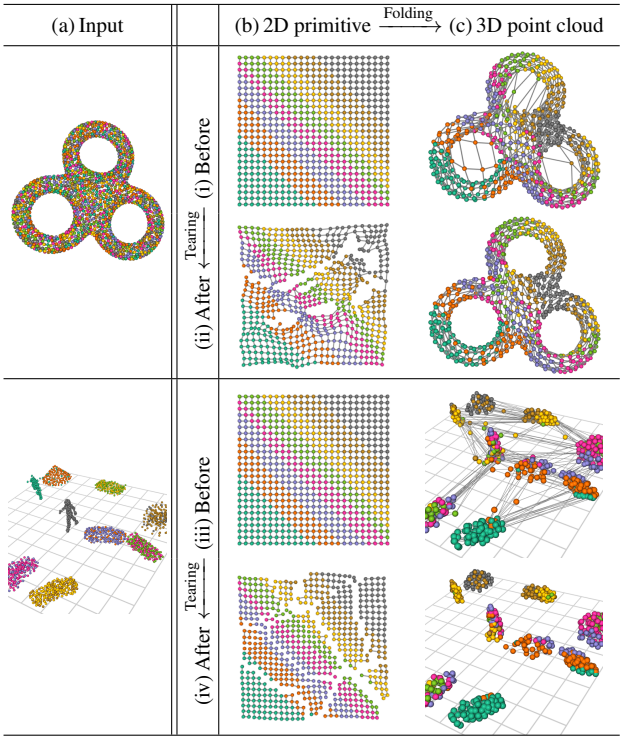



Figure 1. TearingNet for point cloud reconstruction. By *Tearing* and *Folding*, we achieve high-quality reconstructions (c-ii) and (c-iv). Edges of the primitive graphs are also drawn in (b) and (c).

*Work done while the author was an intern at InterDigital.

number of objects (*i.e.*, varying zeroth Betti number [33]). For example, to represent a multi-object scene, a descriptor not only needs to delineate the shapes of individual objects but also the spatial relationship between them. In fact, existing works, including LatentGAN [1], FoldingNet [40], AtlasNet [14], 3D Point Capsule Network [42] and GraphTER [11], all target to reconstruct point clouds with simple topologies, *e.g.*, object point clouds. Some other works, such as AtlasNetV2 [8] and DeepSDF [27], even train class-specific networks to alleviate the effects of ample topologies across classes.

To tackle this challenge, we propose a new autoencoder for point cloud, entitled *TearingNet*. Similar to FoldingNet [40], AtlasNet [14] and 3D Point Capsule Network [42], *etc.*, we reconstruct point clouds by deforming genus zero 2D primitives (*e.g.*, a regular 2D patch). Differently, we novelly *tear* the primitive with holes or into several parts to match its topology to the target point clouds, *e.g.*, Figure 1b-ii and Figure 1b-iv. Our architecture couples a proposed Tearing network and a Folding network, letting them interact with each other. Especially, by running through a trial folding in the first place, our proposal enables the Tearing network to tear a graph defining on the 2D primitive, letting it count for the empty space in object holes and boundaries. The torn primitive graph, with desired topology similar to the target point cloud, is fed to the Folding network again. In this way, the Tearing network and the Folding network run alternatively to decode an accurate reconstruction.

We verified that the TearingNet generates topology-friendly representations. The superiority of the obtained representations is demonstrated in different down-stream tasks, including shape reconstruction, object counting, and object detection. We also examine why the learned representations are topology-aware by analyzing the feature space.

The contributions of our work can be summarized below:

(i) We propose a novel autoencoder—TearingNet—to generate topology-friendly representations for complex 3D point clouds. Our TearingNet includes a Folding network (F-Net) and a Tearing network (T-Net) "talking" to each other by a feedback loop, which gradually improves the reconstructions.

(ii) Our TearingNet endeavors to explicitly learn the target point cloud topology. By tearing a local graph built upon the 2D primitive, we update its topology to match with the ground-truth point cloud. With the notion of graph tearing, our proposal is generalized as a Graph-Conditioned AutoEncoder (GCAE) which discovers and utilizes topology iteratively.

(iii) We unroll the proposed TearingNet then apply its produced representations to several tasks that are sensitive to topology, *e.g.*, reconstruction, and object counting, which verify our superiority. We further analyze the structure of the feature space to understand how the topology-friendliness is achieved.

Our paper is organized as follows. Section 2 reviews related work. In Section 3, we present the methodology of the TearingNet under the notion of graph tearing. Section 4 unrolls the TearingNet architecture then illustrates its individual components. Experimentation is presented in Section 5 and conclusions are provided in Section 6.

## 2. Related Work

Recently, geometric deep learning has shown great potential in various point cloud applications [15]. Compared to deep learning on regularly structured data such as image and video, point cloud learning is, however, more challenging as the points are unorganized and irregularly sampled over the object/scene surface.

**Non-native learning of point cloud**: Conventionally, point clouds are preprocessed, *e.g.*, either voxelized [24, 19] or projected into multiview images [34], so as to carry over deep learning frameworks justified in the image domain. After format conversion, for example, the conventional convolutional neural network (CNN) could be applied on 3D voxels or 2D pixels [7, 31]. Obviously, voxelization exhibits a tradeoff between accuracy and data volume; while multiview projection is a balance between accuracy/occlusion and data volume. Such compromises occur even before the data is fed into deep neural networks. Octree-like approaches [36] demonstrate limited adaptivity on such tradeoffs. Fortunately, emerging techniques for native learning on point clouds relieve the frustration from the front.

**Point cloud encoders**: In [29], Qi *et al.*propose *PointNet*, which directly operates on input points and generates a latent codeword depicting the object shape. The latent code is point permutation invariant through a pooling operation. Once equipped with object-level or part-level labels, PointNet could serve for supervised tasks like classification or segmentation. Qi *et al.* [29] also show that, PointNet is a *universal approximator* of any set functions. In other words, it is highly flexible network whose behavior heavily depends on the overall network design. PointNet++ [30] recursively applies PointNet in a hierarchical manner so as to capture local structures and enhance the ability to recognize fine-grained patterns. With similar motivations, PointCNN [22] utilizes a hierarchical convolution and Dynamic Graph CNN (DGCNN) [37] employs an edge-convolution over graphs. In brief, advanced feature extractors for point clouds often exploit local topology information.

**Point cloud decoders**: As opposed to the advanced feature extractors, designs of current point cloud generators, *e.g.*, the generator in a Generative Adversarial Network (GAN) [13] and the decoder in an autoencoder (AE) [25], appear to be more preliminary without taking the advantage

of topology. For example, topology is never counted by the fully-connected decoder of LatentGAN [1]. Exploiting the fact that point clouds are sampled from 2D surfaces/manifolds, the pioneering works FoldingNet [40] and AtlasNet [14] propose to fold/deform 2D primitives to reconstruct 3D point clouds. They for the first time embed topology explicitly with 2D patches, such as square patches, of genus zero in their decoders.

FoldingNet adopts a PointNet-like [29] encoder to produce latent representations. Like the PointNet encoder, FoldingNet decoder is a point-wise network shared among points. Given a predefined 2D primitive (*e.g.*, 2D square patch or a sphere), the FoldingNet decoder takes a 2D coordinate from the primitive and latent codeword as input, then maps the 2D point to a 3D coordinate. The set of 3D points mapped by the FoldingNet decoder constitute the reconstructed point cloud. Unfortunately, by a direct mapping from the regular 2D samples to 3D points, FoldingNet fails to represent point clouds with complex topologies even if the network is scaled up [40]. AtlasNet [14] and AtlasNetV2 [8] naively duplicate primitive-decoder pairs to comply with different shapes; while 3D Point Capsule Network [42] learns multiple latent capsules to describe individual object parts. However, these approaches do not scale well for point clouds with complex topologies (Section 5.3). In [5], a fully-connected graph is advanced as a companion to the FoldingNet decoder aiming to approximate point cloud topology by a graph. Its main weakness is in the misaligned topology from graphs to point clouds as it allows connections between distant point pairs.

Recent works, such as DeepSDF [27] and DISN [39], propose to use signed distance functions (SDF) as implicit representations of 3D shapes. However, learning of such representations requires additional knowledge of object surface that is difficult to acquire in practice. For instance, the widely used LiDAR sensors only provide *sparse* and *incomplete* point clouds [12].

Motivated by the limitations in the related work, we propose an autoencoder—*TearingNet*. Intuitively, our proposal "tears" the 2D primitive into pieces or with holes so as to align its topology to the target 3D point clouds. It effectively drives the latent representation to be aware of the topology of the point clouds. To the best of our knowledge, TearingNet is the *first* autoencoder that is able to use a fixed-length latent representation to reconstruct multi-object point clouds.

# 3. Topology Update with the TearingNet

## 3.1. Overview

A block diagram of the TearingNet is shown in Figure 2. The Encoder network (referred to as E-Net, "E" in Figure 2) first generates a latent representation $\mathbf{c}$ from an input 3D point cloud. It is then passed to the TearingNet decoder, which consists of two sub-networks. On top of the FoldingNet [40] decoder, referred to as *Folding network* (F-Net) hereinafter, a novel *Tearing network* (T-Net) is proposed to couple with the F-Net by a feedback loop. Given a codeword $\mathbf{c}$, the TearingNet decoder runs the F-Net and the T-Net ("F" and "T" in Figure 2) iteratively. In a nutshell, the F-Net takes as input a certain topology (represented by $\widehat{\mathbf{U}}$) and "embeds" it to a 3D point cloud $\widehat{\mathbf{X}}$, then the T-Net considers $\widehat{\mathbf{X}}$ to "correct" the topology with a feedback connection.

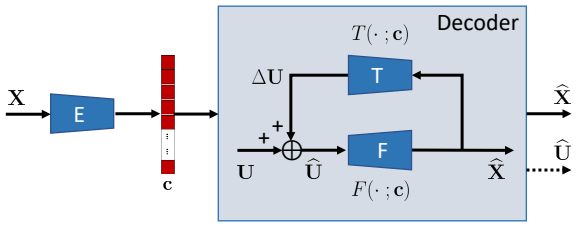



Figure 2. Block diagram of the TearingNet/GCAE, featured by the interaction between the F-Net and the T-Net. "E", "F", "T" represent the E-Net, F-Net and T-Net, respectively.

For an input 3D point cloud $\mathbf{X} = \{\mathbf{x}_i\}_{i=1}^n$ composed of $n$ points $\mathbf{x}_i = (x_i, y_i, z_i)$, the encoder first generates a vector $\mathbf{c} \in \mathbb{R}^d$. As an auxiliary input, a 2D point set $\mathbf{U} = \{\mathbf{u}_i\}_{i=1}^m$ samples $m$ points $\mathbf{u}_i = (u_i, v_i)$ on a 2D region. Similar to FoldingNet [40], it contains regularly sampled 2D-grid locations in the square region $[-1, 1] \times [-1, 1]$ (the points in Figure 1b-i and Figure 1b-iii). This point set $\mathbf{U}$ brings in a *primitive* shape for reconstruction, which embodies a *genus zero* topology. For convenience, the 2D point set is also referred to as *2D grid* in the sequel. With the 2D grid $\widehat{\mathbf{U}}$ (or $\mathbf{U}$ at the first iteration), the Folding network maps each of its 2D point $\widehat{\mathbf{u}}_i$ to a 3D coordinate $\widehat{\mathbf{x}}_i$, aiming at reconstructing a 3D point cloud $\widehat{\mathbf{X}}$ following the topology of $\widehat{\mathbf{U}}$. The Tearing network especially takes $\widehat{\mathbf{X}}$ and modifies each 2D point in $\widehat{\mathbf{U}}$ individually, leading to a new 2D grid representing an updated topology (*e.g.*, Figure 1b-ii and Figure 1b-iv). Hence, the Folding network and the Tearing network interact with each other for high-quality reconstructions.

## 3.2. Tearing as Breaking Graph Edges

With the Tearing network that "stretches" the 2D grid, the overall TearingNet still admits a continuous mapping from the 2D space to 3D. It appears to be *conflict* with the notion of tearing, which implies introducing discontinuities. We fill this gap by viewing tearing as collectively breaking graph edges that connect neighboring points on the 2D grid.

**2D grid as a local graph**: The 2D grid $\mathbf{U}$ in our work (as well as FoldingNet [40] and AtlasNet [14]) essentially approximates a simple Riemannian manifold—a genus zero square patch, denoted as $\mathcal{M}$—with $m$ points regularly sampled on it. From [18, 35], *etc.*, to represent/approximate a Riemannian manifold (in the continuous domain) with its sampled points (in the discrete domain) means to construct a *local graph* connecting the nearby points. In other words,

the set $\mathbf{U}$ implies a primitive graph (denoted as $\mathcal{G}$) associated with the manifold $\mathcal{M}$. This graph $\mathcal{G}$, according to [18], have $m$ vertices, with each represents one point in $\mathbf{U}$. For any two points $\mathbf{u}_i$ and $\mathbf{u}_j$ from $\mathbf{U}$, the graph weight between them is given by a truncated Gaussian kernel:

$$w_{ij} = \begin{cases} \exp\left(-\frac{\|\mathbf{u}_i - \mathbf{u}_j\|_2^2}{2\epsilon^2}\right) & \text{if } \|\mathbf{u}_i - \mathbf{u}_j\|_2 \leq r, \\ 0 & \text{otherwise,} \end{cases} \tag{1}$$

where $\epsilon > 0$ is a parameter controlling the sensitivity of the graph weight, and $r > 0$ is a threshold. Hence, the primitive graph $\mathcal{G}$ is an $r$-neighborhood graph, *i.e.*, there is no edge between two points with a distance greater than $r$.

**Graph tearing**: Suppose the 2D grid $\mathbf{U}$ has $N^2$ points with a dimension $N \times N$. According to [17, 18], as $r$ in Eq. (1) becomes smaller and $N$ becomes larger (*i.e.*, 2D grid $\mathbf{U}$ sampling $\mathcal{M}$ gets denser), the graph $\mathcal{G}$ better approximates $\mathcal{M}$. Given a certain dimension $N$, we also let $r$ be small, which takes a value just equal to or slightly larger than $2/N$—the horizontal/vertical spacing of neighboring points.[1] Then from Eq. (1), each point $\mathbf{u}_i$ has four edges connecting to its neighbors—the top, left, bottom, and right points—on the 2D grid $\mathbf{U}$. Hence, before feeding to the Tearing network, $\mathcal{G}$ defaults back to a simple *2D grid graph*, *e.g.*, Figure 1b-i.

Running the Tearing network updates the 2D grid $\mathbf{U}$ as well as the graph. Particularly, a 2D point $\mathbf{u}_i$ is moved to another location $\widehat{\mathbf{u}}_i$. Thus, for any two neighboring points in the 2D grid that are pulled apart by the Tearing network, the graph edge between them is naturally *broken*, resulting in an updated graph (denoted at $\widehat{\mathcal{G}}$). In our proposal, the breakings of all the edges collectively achieve *tearing in the graph domain* and update the underlying topology:

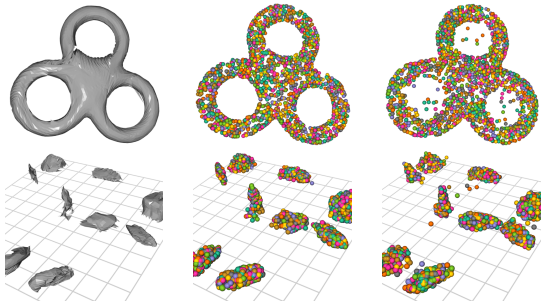
(i) When localized edges within the 2D grid are removed by the Tearing network, a gap/seam is introduced to the graph topology. This is to reconstruct a target point cloud with holes (*i.e.*, its genus number $g > 0$), see Figure 1c-ii.

(ii) When all the edges connecting two groups of points in the 2D grid are removed by the Tearing network, the graph is *torn* apart into disconnected sub-graphs. This benefits the reconstruction when the two groups correspond to two distinct objects. Hence, a point cloud with multiple objects (*i.e.*, its zeroth Betti number $b_0 > 1$) is reconstructed, see Figure 1c-iv.

Note that in practice, both of these two cases may happen on the same point cloud.

**Torn graph as a free mesh**: The torn graph $\widehat{\mathcal{G}}$—as a side output—naturally represents a mesh over the reconstructed point cloud. In fact, each elementary square on the 2D grid corresponds to a quadrilateral face of the 3D mesh, where a face is pruned if it has any edges removed by the Tearing network. Then the remaining faces together constitute a 3D mesh. See Figure 3a for an example.

[1] An even smaller $r$ results in the trivial case which no graph edges exist.



(a) Induced mesh (b) TearingNet (c) FoldingNet

Figure 3. The torn graphs induce 3D meshes (a) and bring better resampled point clouds (b) than that of the FoldingNet (c). Each row in this figure is associated to an example in Figure 1.

The torn graph also enables us to resample the input 3D point cloud by resampling 2D points in manifold $\mathcal{M}$ while avoiding the "ghost" 3D points between different objects or within object holes. This is achieved by removing outliers sampled on the pruned faces (Figure 3b). In contrast, it is inevitable for the simple FoldingNet to introduce undesired points in the resampled point cloud (Figure 3c).

### 3.3. Graph-Conditioned AutoEncoder

The insights of graph tearing motivate us to generalize the architecture of Figure 2 and call it a *Graph-Conditioned AutoEncoder* (GCAE), which we believe useful for processing data where topology matters, *e.g.*, image, video, or any graph signals. It promotes an explicit way to *discover* and *utilize* topology within an autoencoder. Particularly, it is equipped with a graph $\widehat{\mathcal{G}}$ whose topology evolves by iterating F-Net and T-Net. F-Net *embeds* the graph to a reconstruction; while T-Net attempts to *decodes* a graph (in a residual form) from a reconstruction, which may tear a graph into patches or with holes to achieve desired reconstructions.

## 4. The TearingNet Architecture

In this section, we unroll the TearingNet (Figure 2) then elaborate on its components, especially the Tearing network.

### 4.1. Unrolling the TearingNet

As a concrete example, Figure 4 shows an *unrolled* version of the TearingNet where a T-Net is wedged into two iterations of F-Net. We adopt the PointNet architecture [29] as our encoder due to its flexibility (as discussed in Section 2): it has the potential to generate topology-friendly representations given it is trained properly. With the latent code $\mathbf{c}$ (a 512-dimension vector in our work) and the initial

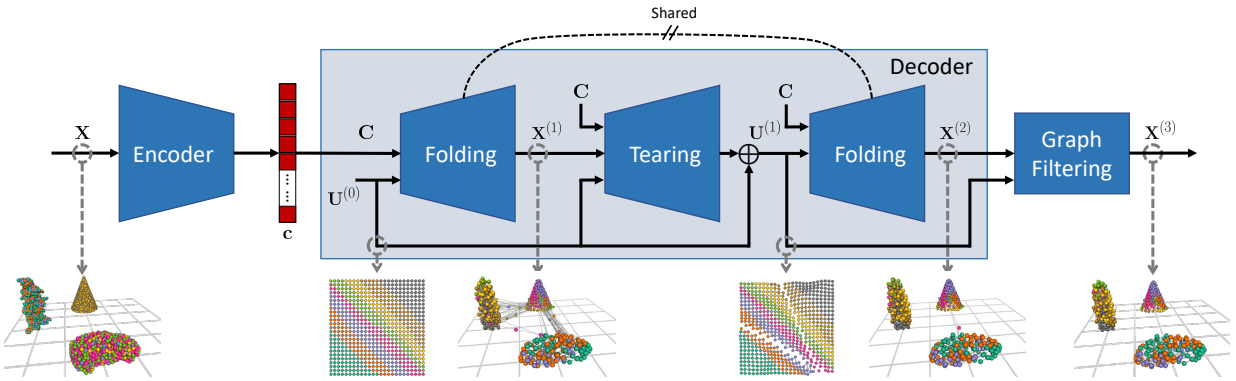



Figure 4. Block diagram of an unrolled TearingNet which has a T-Net wedged in-between two iterations of F-Net.

point set $\mathbf{U}^{(0)}$ (the regular 2D grid) as inputs, the TearingNet decoder (in Figure 4) runs F-Net, T-Net and F-Net sequentially. Specifically, an input point $\mathbf{u}_i^{(0)} \in \mathbf{U}^{(0)}$ is mapped to a 3D point $\mathbf{x}_i^{(2)}$:

$$\begin{aligned} & \mathbf{x}_i^{(1)} = F\left(\mathbf{u}_i^{(0)}; \mathbf{c}\right) \\ \rightarrow\ & \mathbf{u}_i^{(1)} = T\left(\mathbf{u}_i^{(0)}, \mathbf{x}_i^{(1)}; \mathbf{c}\right) + \mathbf{u}_i^{(0)} \\ \rightarrow\ & \mathbf{x}_i^{(2)} = F\left(\mathbf{u}_i^{(1)}; \mathbf{c}\right), \end{aligned} \tag{2}$$

where $F$ and $T$ denote the *point-wise* networks of F-Net and the T-Net, respectively. We see that, the F-Net first endeavors a trial folding to produce a preliminary 3D point cloud $\mathbf{X}^{(1)}$. The T-Net takes $\mathbf{X}^{(1)}$ and generates the 2D point set $\mathbf{U}^{(1)}$. It is then supplied to the second iteration of F-Net for an improved 3D point cloud $\mathbf{X}^{(2)}$.

With the updated 2D point set $\mathbf{U}^{(1)}$ and the second Folding network output $\mathbf{X}^{(2)}$, one may optionally append a *graph filtering* module at the end to further enhance the reconstruction. Note that all reconstructions $\mathbf{X}^{(\cdot)}$ contain $m$ points, the same as that of the 2D pointsets $\mathbf{U}^{(\cdot)}$. By iterating the F-Net twice (as Figure 4), the TearingNet achieves a good tradeoff between computation and reconstruction quality. Hence, we focus on this configuration for experimentation.

### 4.2. The Tearing Network

As a core ingredient, the Tearing network (or T-Net) is introduced to explicitly learn the topology by tearing the primitive graph, which boosts the reconstruction accuracy, and ultimately enhances the representability of the codeword. In our design, the Tearing network learns point-wise modifications to the 2D point set $\mathbf{U}^{(0)}$ and computes $\mathbf{U}^{(1)}$ with a residual connection [16], see Figure 4. The 2D points are then moved around like flocks depending on the topology chart they belong to.

Similar to PointNet [29] and the Folding network [40], the Tearing network consists of shared point-wise MLP layers. Its architecture, as well as its model scale, are similar to that of the Folding network. Given the 2D coordinate $\mathbf{u}_i^{(0)} \in \mathbb{R}^2$, it is first concatenated with the associated 3D coordinate $\mathbf{x}_i^{(1)} \in \mathbb{R}^3$ as well as the codeword $\mathbf{c} \in \mathbb{R}^{512}$, to form a 517-dimensional vector. This vector is then fed to the point-wise network $T$ with two stages of MLP layers, and produces a translation vector on the 2D plane. The detailed architecture of the Tearing network is provided in the supplementary material.

To demonstrate the effectiveness of the Tearing network, we train the whole TearingNet to over-fit the *Torus* dataset introduced in [5] which contains 300 torus-shape point clouds with genus number ranging from 1 to 3. Figure 1c-i and Figure 1c-ii show a genus-3 torus before and after the T-Net respectively, where we see that the 2D grid is "torn" with holes to accommodate the topology of the input torus.

### 4.3. Graph Filtering for Enhancement

Equipped with the torn graph, we propose an optional graph filter appended at the end of the TearingNet (see Figure 4) to improve the reconstruction point cloud [5]. We first compute the unnormalized graph Laplacian matrix $\mathbf{L} \in \mathbb{R}^{m \times m}$ of the graph $\widehat{\mathcal{G}}$. Then we run the following graph filter [32] to obtain the final output $\mathbf{X}^{(3)}$:

$$\mathbf{X}^{(3)} = (\mathbf{I} - \lambda \mathbf{L}) \cdot \mathbf{X}^{(2)} \tag{3}$$

where the point clouds $\mathbf{X}^{(\cdot)}$ are viewed as $m \times 3$ matrices and the filtering parameter $\lambda = 0.5$.

For better filtering, we incorporate the second folding output $\mathbf{X}^{(2)}$ when computing the edge weights—we let $\mathbf{p}_i = \left[\mathbf{u}_i^{(1)^{\mathrm{T}}}\ \mathbf{x}_i^{(2)^{\mathrm{T}}}\right]^{\mathrm{T}}$, and compute $\|\mathbf{p}_i - \mathbf{p}_j\|_2^2$ instead of $\|\mathbf{u}_i - \mathbf{u}_j\|_2^2$ in Eq. (1). Additionally, rather than thresholding with distance $r$, we equivalently remove an edge if its weight is too smaller (*i.e.*, less than $10^{-12}$). We see that our graph filter is a lightweight and differentiable signal processing module for enhancement with little overhead. Thus, it is included in the end-to-end training of the unrolled TearingNet.

## 5. Experimentation

In this section, the training of the TearingNet, alongside with other experimental settings, are first introduced. We then perform the evaluation on three tasks: reconstruction, object counting, and object detection.

### 5.1. Training of the TearingNet

Instead of training the entire TearingNet directly, we first pre-train the Encoder network (E-Net) and the Folding network (F-Net). They are trained together under the FoldingNet [40] autoencoder architecture without the Tearing network. After that, we load the pre-trained E-Net and F-Net then train the overall TearingNet autoencoder as shown in Figure 4. This step specifically lets the Tearing network learn to tear the 2D grid/primitive graph and update the topology. In this step, a smaller learning rate is adopted. Details of the training strategy are provided in the supplementary material.

Similar to [6, 40], we train the overall TearingNet (and other methods) with the *augmented Chamfer distance*. Given an original and a reconstructed point clouds being $\mathbf{X}$ and $\widehat{\mathbf{X}}$, respectively, the augmented Chamfer distance is written as:

$$d_{\mathbf{X},\widehat{\mathbf{X}}} = \max\left\{\frac{1}{n}\sum_{\mathbf{x}\in\mathbf{X}}\min_{\widehat{\mathbf{x}}\in\widehat{\mathbf{X}}}\|\mathbf{x}-\widehat{\mathbf{x}}\|_2, \frac{1}{m}\sum_{\widehat{\mathbf{x}}\in\widehat{\mathbf{X}}}\min_{\mathbf{x}\in\mathbf{X}}\|\mathbf{x}-\widehat{\mathbf{x}}\|_2\right\}. \tag{4}$$

Eq. (4) essentially gives the Hausdorff distance between two point clouds [3]. As analyzed in [6], it is more robust to ill cases compared to the original Chamfer distance [10].

### 5.2. Experimental Setup

**Datasets**: We verify our work with both single- and multi-object point cloud datasets, with a focus on the latter ones. Particularly, we adopt ShapeNet [4] and Torus [5] datasets to experiment with the single-object scenarios; while we collect objects from off-the-shelf point cloud datasets to synthesize multi-object point cloud datasets. To assemble a point cloud with $k$ objects, a $K \times K$ square-shaped "playground" with $K^2$ grids is defined to host objects. Then randomly picked $k \leq K^2$ objects are normalized and randomly placed on the grids of the playground.

We first generate multi-object datasets with objects from KITTI 3D Object Detection [12]. In total, 10165 objects from KITTI with labels `Pedestrian`, `Cyclist`, `Car`, `Van` and `Truck` are "cropped" using annotated bounding boxes.

Four datasets are created with playground dimensions $K \in \{3, 4, 5, 6\}$, and the resulting KITTI multi-object datasets are called KIMO-$K$ respectively. Each KIMO-$K$ dataset is composed of $K \times 10000$ and $K \times 2000$ point clouds for training and testing, where each point cloud has up to $K^2$ objects. All generated point clouds have 2048 points, and each object in a point cloud occupies roughly the same number of points. The KIMO-$K$ datasets are challenging, since they are composed of real LiDAR scans that are sparse and incomplete (*e.g.*, ground-truths of the "K" rows in Table 1). For better visualization, we similarly generate datasets that we call CAD model multi-object (CAMO) which are composed of point clouds sampled from CAD models in ModelNet40 [38] and ShapeNet [4]. More details are discussed in the supplementary material.

**Implementation details**: The 2D grid $\mathbf{U}$ is defined to be $45 \times 45$, and the codeword $\mathbf{c}$ to be 512-dimension. Adam optimizer [20] is applied for training with a batch size 32. The constant $\epsilon$ in (1) is set to be 0.02. We pre-train E-Net and F-Net for 640 epochs with a learning rate of $2 \times 10^{-4}$, then train TearingNet end-to-end for another 480 epochs with a smaller learning rate $10^{-6}$. Isolated points with no edges are removed from the reconstructions. Our experiments are implemented with the PyTorch framework [28].

**Benchmarks**: We compare the TearingNet with several methods: (i) LatentGAN [1] composed of the fully-connected layers, which exploits a much larger model than ours; (ii) AtlasNet [14] with 3 patches which has the same model size as ours; (iii) FoldingNet [40], and its extension, (iv) Cascaded F-Net, with two FoldingNets cascaded as $F_2(F_1(\mathbf{u}; \mathbf{c}); \mathbf{c})$. Its model size is similar to ours. Note that besides (i), the other methods all reconstruct point clouds via deforming 2D primitive(s).

We also consider several variants of the TearingNet for shape reconstruction: i) TearingNet$_{\text{TF}}$: instead of having a trial folding first, this configuration runs a T-Net directly for topology update without considering $\mathbf{X}^{(1)}$; ii) TearingNet$_{\overline{\text{GF}}}$ which excludes the graph filtering at the end; iii) TearingNet$_3$ which augments the TearingNet by iterating three times, *i.e.*, as $F \rightarrow T \rightarrow F \rightarrow T \rightarrow F$. It is trained via loading the weights from a pre-trained TearingNet followed by a finetuning.

### 5.3. Performance Comparison

**Reconstruction**: We first evaluate the reconstruction quality of the proposed TearingNet. Table 1 visualizes some reconstructed point clouds from several datasets. Compared to the TearingNet, FoldingNet leaves more "ghost" points outside object surfaces, while AtlasNet results in unbalanced point distributions. In contrast, TearingNet produces point clouds that look *clean* and *orderly*, with appearances close to the inputs. The 2D-grids (last column of Table 1) are torn apart to accommodate the corresponding 3D topologies. It demonstrates how object topologies are *discovered* and *utilized* via the TearingNet architecture.

We report the augmented Chamfer Distance (referred to as CD) and the Earth Mover's Distance (EMD) [10] of the competing methods in Table 2. For both metrics, a smaller number indicates more accurate reconstruction. In general, our TearingNet and TearingNet$_3$ outperform the benchmarks, which is even more obvious in terms of EMD.

Table 1. Visual comparisons of point cloud reconstructions. Points are colored according to their indices. S: ShapeNet; T: Torus; C: CAMO-5; K: KIMO-5. Objects in the red boxes are zoomed in and shown in the blue boxes.


Table 2. Evaluation of 3D point cloud reconstruction, in terms of both CD and EMD.

| Metrics | CD ($\times 10^{-2}$) | | | | | | EMD | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Datasets | ShapeNet | Torus | KI.-3 | KI.-4 | KI.-5 | KI.-6 | ShapeNet | Torus | KI.-3 | KI.-4 | KI.-5 | KI.-6 |
| LatentGAN | 2.85 | 2.45 | 7.10 | 11.64 | 17.18 | 19.21 | 0.218 | 0.202 | 1.982 | 3.231 | 3.773 | 4.721 |
| AtlasNet | 2.72 | 2.41 | **4.50** | 6.76 | 9.59 | 11.63 | **0.163** | **0.146** | 1.333 | 2.811 | 3.173 | 4.440 |
| FoldingNet | 2.75 | 1.90 | **4.72** | 6.57 | 9.01 | 11.06 | 0.372 | 0.191 | 1.748 | 2.864 | 3.056 | 4.569 |
| Cascaded F-Net | 2.69 | 1.95 | 4.77 | 6.67 | 9.13 | 10.94 | 0.207 | 0.196 | 1.533 | 2.381 | 2.944 | 4.189 |
| TearingNet$_{\text{TF}}$ | 2.59 | 1.84 | 5.05 | 6.58 | 8.55 | 10.86 | 0.206 | 0.163 | 1.055 | 1.787 | 2.565 | 3.476 |
| TearingNet$_{\overline{\text{GF}}}$ | 2.56 | 1.74 | 4.92 | 6.45 | 8.29 | 10.29 | 0.172 | 0.170 | 0.958 | 1.441 | 1.879 | 2.648 |
| TearingNet (Ours) | **2.54** | **1.72** | 4.78 | **6.43** | **8.29** | **10.23** | 0.174 | 0.156 | **0.940** | **1.438** | **1.872** | **2.614** |
| TearingNet$_3$ (Ours) | **2.53** | **1.73** | 4.74 | **6.42** | **8.24** | **10.15** | **0.169** | **0.143** | **0.941** | **1.361** | **1.867** | **2.315** |

As the topology complexity grows from KIMO-3 to KIMO-6, our method outperforms the competitors more significantly. By comparing our TearingNet to TearingNet$_{\text{TF}}$, we see the effectiveness of inserting a Folding network before the Tearing network. That is because, with the first trial folding result, the Tearing network can better capture the discrepancy between a genus zero topology and the ground-truth topology via back-propagation. From Table 2, we also see that, begin with the TearingNet$_{\overline{\text{GF}}}$, by first incorporating the graph filter (TearingNet), then further iterating the T-Net and the F-Net (TearingNet$_3$), reconstruction qualities continues to improve. We observe similar results on the CAMO datasets.

We also experimented with recent methods AtlasNetV2 [8] and 3D Point Capsule Network [42] and observe good performance on single-object datasets. On ShapeNet, AtlasNetV2 has even achieved a lowest CD of $2.48 \times 10^{-2}$. However, both methods fail to converge on our multi-object datasets. We conjecture that is because both of them are over-optimized for single-object cases, *e.g.*, AtlasNetV2 specifically learns the elementary structure of objects. Hence, the diversified scene configurations in our multi-object datasets "confuse" these two methods.

**Object counting**: In a multi-object scene, adding objects yields a more complex topology. From the multi-object examples in Table 1, we see that the number of torn patches in a 2D-grid approximately equals the number of objects in a scene. It implies that the latent codewords from TearingNet are *aware of the geometric topology*. To further affirm their representativeness of topologies, we next try to "count" the object number directly from the codewords. In practice,

Table 3. Evaluation of object counting and object detection.

| Tasks | Methods | Datasets | | | |
|---|---|---|---|---|---|
| | | KI.-3 | KI.-4 | KI.-5 | KI.-6 |
| Counting (MAE, $\times 10^{-1}$) | LatentGAN | 0.671 | 8.439 | 14.079 | 14.523 |
| | AtlasNet | 0.125 | 2.908 | 6.727 | 8.569 |
| | FoldingNet | 0.204 | 3.031 | 6.340 | 8.527 |
| | Cascaded F-Net | 0.270 | 3.207 | 7.104 | 9.792 |
| | TearingNet$_{\text{TF}}$ | 0.127 | 2.207 | 5.716 | 8.346 |
| | TearingNet$_{\overline{\text{GF}}}$ | 0.123 | 1.805 | 5.105 | 8.044 |
| | TearingNet (Ours) | **0.123** | **1.721** | **5.079** | **8.026** |
| | TearingNet$_3$ (Ours) | **0.121** | **1.740** | **5.050** | **7.992** |
| Detection (Accuracy, %) | LatentGAN | **93.53** | 63.78 | 65.65 | 78.80 |
| | AtlasNet | 88.84 | 73.79 | 73.58 | 83.53 |
| | FoldingNet | 92.71 | 80.12 | 77.10 | 82.92 |
| | Cascaded F-Net | 89.78 | 76.36 | 76.84 | 82.63 |
| | TearingNet$_{\text{TF}}$ | 93.33 | 82.83 | 78.43 | 83.74 |
| | TearingNet$_{\overline{\text{GF}}}$ | 93.44 | 83.42 | 79.70 | 84.55 |
| | TearingNet (Ours) | 93.42 | **83.42** | **79.74** | **84.55** |
| | TearingNet$_3$ (Ours) | **93.46** | **83.44** | **79.72** | **86.52** |

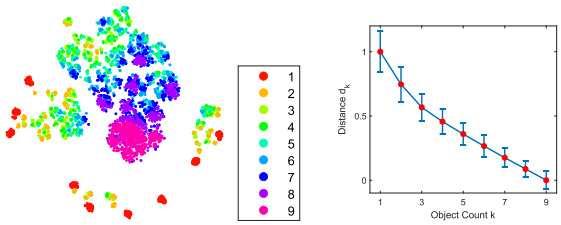
counting is a critical task in applications such as traffic jam detection and crowd analysis [21, 26].

In this task, the TearingNet and other benchmarks trained from the reconstruction experiment are carried over. We again use the challenging KIMO datasets to experiment with this use case. As a preparation, we feed the test dataset to the PointNet encoder to collect codewords. Next, we employ 4-fold cross-validation to train/test an SVM classifier: codewords are equally divided into 4 folds, then *only one* of the four is used to train the SVM together with their count labels while the other three are reserved for counting test. SVM is selected for the test as it would not modify the feature space learned by autoencoders. Consequently, this setting overall requires a small number of labels, because our feature (codeword) learning is achieved in an *unsupervised* manner, while the counting task is learned in a *weakly supervised* manner.

The counting performance is measured by mean absolute error (MAE) between the predicted counting and the ground-truth counting [41]. As shown in the upper-half of Table 3, TearingNet and TearingNet$_3$ consistently produce the smallest MAEs. For example, on KIMO-4, TearingNet brings down MAE by more than 40% comparing to FoldingNet/AtlasNet, showing its strong capability in representing scene topologies.

We further inspect the feature space learned by the TearingNet to understand how it is linked to the topology. We first collect the TearingNet codewords of the KIMO-3 dataset and visualize them using t-SNE, as shown in Figure 5a. Here the points are colored based on their corresponding counting labels. Note that for the $3 \times 3$ playground in KIMO-3, there are 9 and 36 combinations when placing 1 and 2 objects, respectively. Correspondingly, 9 and 36 clusters could be observed in the t-SNE figure. And as there is only 1 possible combination to arrange 9 objects, all points representing 9 objects aggregate to a single cluster.

Finally, the appearance of the t-SNE diagram exhibits a *tree* structure. When inspecting one cluster of a larger counting (*e.g.*, 9, 8, *etc.*), it is always surrounded by several smaller counting clusters (*e.g.*, 8, 7, *etc.*). This observation




(a) t-SNE visualization (b) $d_k$ vs. object count $k$

Figure 5. Analyzing the feature space of the TearingNet.

is actually due to a recursive encapsulation from counting 1 to 9 where counting 9 stays at the center. If we compute an average Euclidean distance $d_k$ from all codewords of counting $k$ to the mean codeword of counting 9, we observe that $d_k$ approximately linearly increases as object counting $k$ decreases (see Figure 5b, where the distances are normalized to $[0, 1]$, the error bars are also shown). It implies that the codewords distribute in a layered manner with respect to counting (*i.e.*, topology) and they are *topology-aware*.

**Object detection**: Having revealed the superiority of our proposal in point reconstruction and topology understanding, we finally devised a last experiment to demonstrate such superiority in low/mid-level tasks can be transferred to high-level understanding tasks. Specially, we consider the pedestrian detection task which is critical under an autonomous driving scenario [9]. Similar to object counting, we train binary SVM classifiers and evaluate their performance using a 4-fold cross-validation strategy. Again, one fold is applied for training and the rest for testing. Detection accuracy is collected in the bottom-half of Table 3. Note that KIMO-3 is the easiest dataset as it contains the least combination possibilities, and thus the simple LatentGAN already provides good accuracy. Compared to the other benchmarks, both TearingNet and TearingNet$_3$ perform comparable on KIMO-3 and significantly better on KIMO-4, -5, and -6. Moreover, for KIMO-4, TearingNet surpasses AtlasNet and FoldingNet by about 10% and 3%, respectively.

## 6. Conclusion

We consider the problem of representing and reconstructing point clouds of ample topologies with an autoencoder, given the latent representations in the form of a fixed-length vector. To tackle this task, we propose a TearingNet architecture which iteratively discovers and utilizes topology with a Tearing network and a Folding network, respectively. The superior capability of our proposal is demonstrated in terms of shape reconstruction and producing topology-friendly representations for point clouds. Essentially, the Tearing network reparameterizes the surface defined by the Folding network according to a learned topology. For future research, we plan to apply the TearingNet for scene point clouds with natural object placement. We are also interested in applying its generalization—the GCAE—to other data modalities where topology matters, such as images and videos.

## References


[1] Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. Learning representations and generative models for 3D point clouds. In *International Conference on Machine Learning*, pages 40–49, 2018. 2, 3, 6

[2] Davide Boscaini, Jonathan Masci, Emanuele Rodolà, and Michael Bronstein. Learning shape correspondence with anisotropic convolutional neural networks. In *Adv. Neural Inform. Process. Syst.*, pages 3189–3197, 2016. 1

[3] Sofien Bouaziz, Andrea Tagliasacchi, Hao Li, and Mark Pauly. Modern techniques and applications for real-time non-rigid registration. In *SIGGRAPH ASIA 2016 Courses*, pages 1–25. Association for Computing Machinery, 2016. 6

[4] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. ShapeNet: An information-rich 3D model repository. *arXiv preprint arXiv:1512.03012*, 2015. 6

[5] Siheng Chen, Chaojing Duan, Yaoqing Yang, Duanshun Li, Chen Feng, and Dong Tian. Deep unsupervised learning of 3D point clouds via graph topology inference and filtering. *IEEE Trans. Image Process.*, 2019. 3, 5, 6

[6] Siheng Chen, Baoan Liu, Chen Feng, Carlos Vallespi-Gonzalez, and Carl Wellington. 3D point cloud processing and learning for autonomous driving. *IEEE Signal Processing Magazine*, 2020. 1, 6

[7] Christopher B Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3D-R2N2: A unified approach for single and multi-view 3D object reconstruction. In *Eur. Conf. Comput. Vis.*, pages 628–644. Springer, 2016. 2

[8] Theo Deprelle, Thibault Groueix, Matthew Fisher, Vladimir Kim, Bryan Russell, and Mathieu Aubry. Learning elementary structures for 3D shape generation and matching. In *Adv. Neural Inform. Process. Syst.*, pages 7433–7443, 2019. 2, 3, 7

[9] Piotr Dollár, Christian Wojek, Bernt Schiele, and Pietro Perona. Pedestrian detection: A benchmark. In *CVPR*, pages 304–311, 2009. 8

[10] Haoqiang Fan, Hao Su, and Leonidas J Guibas. A point set generation network for 3D object reconstruction from a single image. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 605–613, 2017. 6

[11] Xiang Gao, Wei Hu, and Guo-Jun Qi. GraphTER: Unsupervised learning of graph transformation equivariant representations via auto-encoding node-wise transformations. In *IEEE Conf. Comput. Vis. Pattern Recog.*, June 2020. 1, 2

[12] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? The KITTI vision benchmark suite. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 3354–3361, 2012. 3, 6

[13] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Adv. Neural Inform. Process. Syst.*, pages 2672–2680, 2014. 2

[14] Thibault Groueix, Matthew Fisher, Vladimir G Kim, Bryan C Russell, and Mathieu Aubry. A papier-mâché approach to learning 3D surface generation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 216–224, 2018. 2, 3, 6

[15] Yulan Guo, Hanyun Wang, Qingyong Hu, Hao Liu, Li Liu, and Mohammed Bennamoun. Deep learning for 3D point clouds: A survey. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2020. 2

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 770–778, 2016. 5

[17] Matthias Hein. Uniform convergence of adaptive graph-based regularization. In *International Conference on Computational Learning Theory*, pages 50–64. Springer, 2006. 4

[18] Matthias Hein, Jean-Yves Audibert, and Ulrike von Luxburg. Graph Laplacians and their convergence on random neighborhood graphs. *Journal of Machine Learning Research*, 8(Jun):1325–1368, 2007. 3, 4

[19] Anastasia Ioannidou, Elisavet Chatzilari, Spiros Nikolopoulos, and Ioannis Kompatsiaris. Deep learning advances in computer vision with 3D data: A survey. *ACM Computing Surveys (CSUR)*, 50(2):1–38, 2017. 2

[20] Diederick P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Int. Conf. Learn. Represent.*, 2015. 6

[21] Victor Lempitsky and Andrew Zisserman. Learning to count objects in images. In *Adv. Neural Inform. Process. Syst.*, pages 1324–1332, 2010. 8

[22] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. PointCNN: Convolution on X-transformed points. In *Adv. Neural Inform. Process. Syst.*, pages 820–830, 2018. 2

[23] Jonathan Masci, Davide Boscaini, Michael Bronstein, and Pierre Vandergheynst. Geodesic convolutional neural networks on Riemannian manifolds. In *IEEE International Conference on Computer Vision Workshops*, pages 37–45, 2015. 1

[24] 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*, pages 922–928, 2015. 2

[25] Andrew Ng et al. Sparse autoencoder. *CS294A Lecture Notes*, 72(2011):1–19, 2011. 1, 2

[26] Daniel Onoro-Rubio and Roberto J López-Sastre. Towards perspective-free object counting with deep learning. In *Eur. Conf. Comput. Vis.*, pages 615–629, 2016. 8

[27] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. DeepSDF: Learning continuous signed distance functions for shape representation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 165–174, 2019. 2, 3

[28] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *Advances in Neural Information Processing Systems Workshop*, 2017. 6

[29] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. PointNet: Deep learning on point sets for 3D classification and segmentation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 652–660, 2017. 2, 3, 4, 5

[30] Charles R. Qi, Li Yi, Hao Su, and Leonidas J Guibas. PointNet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems*, pages 5099–5108, 2017. 2

[31] Riccardo Roveri, Lukas Rahmann, Cengiz Oztireli, and Markus Gross. A network architecture for point cloud classification via automatic depth images generation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 4176–4184, 2018. 2

[32] David I Shuman, Sunil K Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*, 30(3):83–98, 2013. 5

[33] Edwin H Spanier. *Algebraic topology*. Springer Science & Business Media, 1989. 2

[34] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller. Multi-view convolutional neural networks for 3D shape recognition. In *Int. Conf. Comput. Vis.*, pages 945–953, 2015. 2

[35] Daniel Ting, Ling Huang, and Michael Jordan. An analysis of the convergence of graph Laplacians. In *International Conference on Machine Learning*, page 1079–1086, 2010. 3

[36] Peng-Shuai Wang, Chun-Yu Sun, Yang Liu, and Xin Tong. Adaptive O-CNN: A patch-based deep representation of 3D shapes. *ACM Trans. Graph.*, 37(6):1–11, 2018. 2

[37] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph CNN for learning on point clouds. *ACM Trans. Graph.*, 38(5):1–12, 2019. 2

[38] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3D ShapeNets: A deep representation for volumetric shapes. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 1912–1920, 2015. 6

[39] Qiangeng Xu, Weiyue Wang, Duygu Ceylan, Radomir Mech, and Ulrich Neumann. DISN: Deep implicit surface network for high-quality single-view 3D reconstruction. In *Adv. Neural Inform. Process. Syst.*, pages 492–502, 2019. 3

[40] Yaoqing Yang, Chen Feng, Yiru Shen, and Dong Tian. FoldingNet: Point cloud auto-encoder via deep grid deformation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 206–215, 2018. 1, 2, 3, 5, 6

[41] Cong Zhang, Hongsheng Li, Xiaogang Wang, and Xiaokang Yang. Cross-scene crowd counting via deep convolutional neural networks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 833–841, 2015. 8

[42] Yongheng Zhao, Tolga Birdal, Haowen Deng, and Federico Tombari. 3D point capsule networks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 1009–1018, 2019. 1, 2, 3, 7