Received October 5, 2019, accepted October 20, 2019, date of publication October 28, 2019, date of current version November 7, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2949785

ELF-Nets: Deep Learning on Point Clouds Using Extended Laplacian Filter

SEON-HO LEE, (Student Member, IEEE), HAN-UL KIM, (Student Member, IEEE), AND CHANG-SU KIM, (Senior Member, IEEE)
School of Electrical Engineering, Korea University, Seoul 136-701, South Korea

Corresponding author: Chang-Su Kim (changsukim@korea.ac.kr)

This work was supported in part by the Cross-Ministry Giga KOREA Project Grant funded by the Korean Government (MSIT) through the development of 4D reconstruction and dynamic deformable action model based hyper-realistic service technology under Grant GK18P0200, and in part by the National Research Foundation of Korea (NRF) Grant funded by the Korean Government (MSIP) under Grant NRF-2018R1A2B3003896.

ABSTRACT We propose a deep learning framework for various 3D vision tasks, which takes a point cloud as input. The convolution is a basic operator for feature extraction in deep learning. However, it is not directly applicable to a point cloud, which is an irregular, unordered point set. This makes deep learning on point clouds challenging. To address this issue, we propose the extended Laplacian filter (ELF) for point clouds, which adopts the design principles of discrete Laplacian filters in 2D image processing. In other words, ELF extends the Laplacian filters and has the following two properties: 1) it is a two-state filter using two filter matrices (one for a center point and the other for neighboring points), and 2) it employs a scalar weighting function to predict the relative importance of the neighboring points. Then, we develop ELF-Nets, which consist of ELF convolution layers and fully connected layers. Experimental results demonstrate that the proposed ELF-Nets are capable of recognizing the 3D shape of a point cloud effectively and efficiently. In particular, ELF-Nets provide better or comparable performances than the state-of-the-art techniques in both object classification and part segmentation tasks.

INDEX TERMS Point cloud, convolutional neural network, Laplacian filter, 3D deep learning, object classification, semantic part segmentation.

I. INTRODUCTION

Point clouds can be acquired by employing 3D sensing techniques, such as LiDAR scanning or stereo reconstruction. They are widely used in various 3D vision applications, including self-driving cars [1]–[3], medical image analysis [4], robotics, virtual reality, and geographic information systems [5]. Thus, point cloud processing has drawn increasing interest from both industry and academia. Recently, with the great success of convolutional neural networks (CNNs) in 2D image processing [6]–[11], several attempts [12]–[15] have been made to exploit deep learning for point cloud processing. However, deep learning on point clouds is challenging. Figure 1 illustrates a point cloud, where points are irregularly distributed in a 3D space. Due to the irregularity, it is difficult to apply conventional CNN techniques for 2D regular pixels, including convolution and pooling, directly to point clouds.

The associate editor coordinating the review of this manuscript and approving it for publication was Abdel-Hamid Soliman.

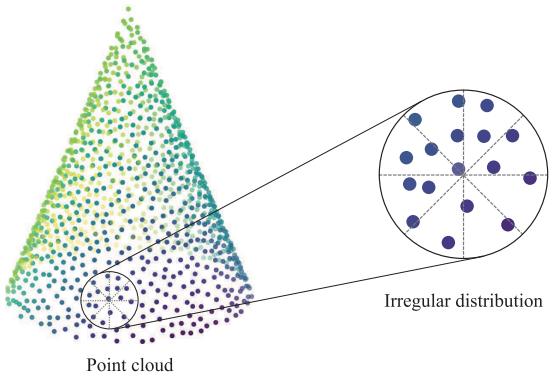

FIGURE 1. An example of a 3D point cloud: Each point is associated with optional normal and color vectors. In this paper, only position and (optional) normal vectors are used for object classification and semantic part segmentation, whereas color vectors are not used.

To overcome this issue, early methods [12], [16]–[18] convert a point cloud into 3D voxels in a regular grid and use 3D CNNs to extract features. However, due to high

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see http://creativecommons.org/licenses/by/4.0/

computational and memory costs, voxel representation is often limited to low levels of resolution, yielding quantization artifacts and making it difficult to handle fine details of 3D shapes. For efficient computation, octrees [19] and k-d trees [20] can be used to skip convolution operations on empty voxels. However, the conversion to the voxel format inevitably incurs an information loss or an overhead.

Recently, several deep neural networks [13]–[15], [21], [21]–[25] have been developed to process point cloud data directly. As a pioneering algorithm, PointNet [13] adopts multi-layer perceptron to extract deep features of points. However, it does not consider the spatial structure of points. A key to the great success of CNNs is that they use convolution layers to exploit spatial correlation. Hence, the methods in [14], [15], and [21]–[24] attempt to use local correlation of points in a 3D space. Especially, Xu *et al.* [15] developed a modified convolution operation on point clouds.

In this paper, we propose a novel point processing operation, called ELF convolution, which extends the conventional 2D convolution. In particular, ELF is an extended version of 2D discrete Laplacian filters [26]. While Xu *et al.* [15] also proposed a 3D operation based on the 2D convolution, their operation is based on the Taylor expansion and demands significantly more parameters than the proposed ELF convolution. Also, we propose ELF-Nets using ELF convolution layers, which perform 3D object classification and semantic part segmentation, respectively. Given a point cloud, the proposed algorithm extracts pointwise feature vectors using ELF convolution layers. Then, the global feature vector is obtained through the global max pooling of those pointwise feature vectors. For 3D object classification, the global feature vector is propagated to fully connected layers to predict the category of the input point cloud. On the other hand, for semantic part segmentation, not only the global feature vector but also the pointwise feature vectors are used to predict the segmentation label of each point.

We evaluate the performance of the proposed ELF-Nets on popular benchmark datasets: ModelNet40 [17], SHREC15 [27], and ShapeNet [28]. Experimental results demonstrate that ELF-Nets provide better or comparable performances than the state-of-the-art techniques in [12]–[17], [20], [21], [25], and [29]–[31] on the three datasets.

To summarize, this work has the following main contributions.

- We propose the novel operation, called ELF convolution, to process unordered points in a 3D space.
- We develop two ELF-Nets for 3D vision tasks: one for 3D object classification and the other for semantic part segmentation. The proposed ELF-Nets effectively capture both global and local structures of a point cloud, by stacking a series of ELF convolution layers.
- The proposed ELF-Nets yield better or comparable results than the state-of-the-art algorithms [12]–[17], [20], [21], [25], [29]–[31], [31] on ModelNet40, SHREC15, and ShapeNet.

The rest of this paper is organized as follows: Section II reviews related work. Section III describes the proposed algorithm, and Section IV assesses its performance comparatively. Finally, Section V concludes this work.

II. RELATED WORK

While deep learning has been popular and successfully employed in a wide variety of 2D vision tasks, relatively little effort has been made for its application to 3D tasks. This is mainly because, unlike 2D images, typical 3D data formats, such as point clouds or meshes, do not have regular structures. Deep learning techniques, developed for regular signals, cannot be used directly for irregular 3D data. Hence, some algorithms represent 3D objects in the regular voxel domain or project them onto multi-view 2D images, before processing them using deep learning techniques. Recently, several deep learning techniques have been proposed to directly process irregular 3D data, such as point clouds or graphs. In this section, we review these regular and irregular domain approaches briefly.

A. REGULAR DOMAIN APPROACHES

1) MULTI-VIEW ALGORITHMS

This approach projects a 3D object onto multiple 2D views and processes them using CNNs. In [12], [32], and [33], a point cloud is represented by a collection of 2D images from different viewpoints. Early algorithms [12], [32] extract features from projected images using CNNs and aggregate them via pooling. Yu *et al.* [33] utilize local features obtained from image patches. Feng *et al.* [34] divide multi-view images into several groups with different weights to obtain a more discriminative descriptor for a 3D shape. Kanezaki *et al.* [35] perform viewpoint estimation and 3D object classification jointly, which facilitates the extraction of view-specific features.

These multi-view algorithms can utilize conventional 2D CNNs and yield promising results in 3D object classification. They, however, cannot be extended straightforwardly to other 3D vision tasks, requiring per-point prediction, such as part segmentation and scene segmentation.

2) VOXEL ALGORITHMS

In the voxel representation, a 3D space is divided into regular cubes, each of which is assigned a binary variable to indicate whether the cubic voxel is occupied or empty. Wang and Posner [36] extract hand-crafted geometry features from each voxel and feed them into an SVM classifier. Also, in [16], [17], and [37], deep voxel features are extracted by applying 3D CNNs. However, as computational and memory requirements increase cubically with the resolution of voxels, only low levels of resolution are supported in practice, making it hard to exploit detailed shape information. To address this issue, Li *et al.* [18] use a probing filter to extract features efficiently. Alternatively, to improve computational and memory efficiency, [19], [20] exploit the sparsity of occupied voxels in a 3D space, by adopting tree data structures.

B. IRREGULAR DOMAIN APPROACHES

1) POINT CLOUD INPUT

Recently, several deep neural networks have been developed, which directly accept point clouds as input. Qi *et al.* [13] proposed PointNet, which exploits multi-layer perceptron (MLP) to extract the feature vector of each point and then aggregates those vectors via global max pooling. However, since PointNet processes each point independently, it cannot use the structural information in the neighborhood of each point. To overcome this limitation, Qi *et al.* [14] developed PointNet++ to capture local structures in a point cloud. It selects a subset of points based on the farthest point sampling, constructs local regions by finding neighboring points around each selected point, and uses PointNet to extract local features from each local region. Also, to capture local geometric information, Shen *et al.* [38] proposed the kernel correlation and the graph pooling. Li *et al.* [21] algorithm adopts the self organizing map [39] to model spatial distribution of local points and extracts hierarchical features of both individual points and nodes of the self organizing map. Huang *et al.* [25] algorithm slices points along the x, y, and z axes and uses recurrent neural networks (RNNs) to obtain features from the spatial slices.

Basic operations for processing point clouds have been also proposed. Su *et al.* [24] project data to a predefined regular domain and obtain features by applying a bilateral filter to the projected data. Xu *et al.* [15] SpiderConv is a modified version of the well-known convolution. It exploits the Taylor expansion to approximate a convolutional filter as a parameterized function of input points. Hua *et al.* [40] pointwise convolution allocates points to 3D grid cells, represents each grid cell with the average of points within the cell, and performs the 3D convolution on the regular grid. In this work, we develop ELF convolution, which generalizes the Laplacian filter to process irregular point data.

2) GRAPH INPUT

Graphs are widely used to represent data in an irregular domain. After Bruna *et al.* [41] introduced a neural network architecture on graphs, various techniques have been developed for deep learning on graphs [30], [42]–[46]. Wang *et al.* [45] combine PointNet++ [14] with a graph convolutional network. They group neighboring points as in [14], build a graph representing points in each neighborhood, and then apply the graph convolution to extract features. [46] and [30] proposed generalized versions of the convolution, which are defined on graphs. Specifically, Monti *et al.* [46] defined the convolution as a Gaussian mixture model in a pseudo coordinate system computed from the graph structure. Also, Simonovsky and Komodakis [30] defined their generalized convolution on a local graph as filtering, whose weights are conditioned on edge labels in the neighborhood of a vertex.

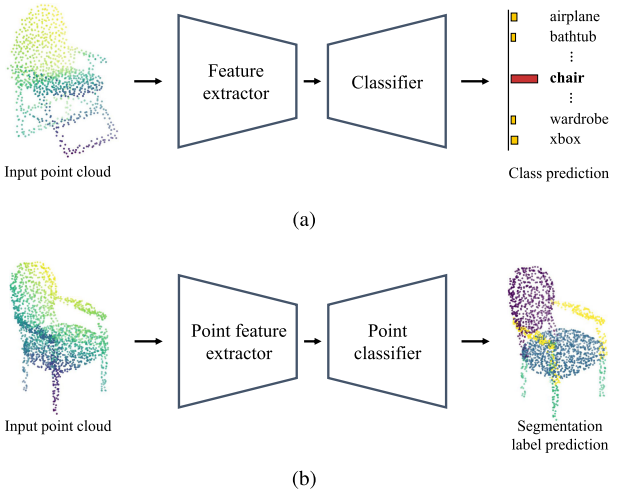

FIGURE 2. An overview of the proposed ELF-Nets for (a) 3D object classification and (b) semantic part segmentation: Given a point cloud, the ELF-Net for classification first extracts its features and then classifies it into one of pre-defined object categories. Also, the ELF-Net for part segmentation assigns each point a semantic label. In this example, the chair is divided into 'legs,' 'seat,' 'arms,' and 'back,' which are colored in green, blue, yellow, and purple, respectively.

III. PROPOSED ALGORITHM

We develop two deep learning networks, called ELF-Nets, for 3D object classification and part segmentation, respectively. Figure 2 is an overview of these ELF-Nets. Given a point cloud, the proposed ELF-Nets extract discriminative features, which represent geometrical shapes of points faithfully, and use those features for classification or part segmentation. For the effective feature extraction, we propose the ELF convolution as a basic operator, by extending discrete Laplacian filters in 2D image processing. Thus, the feature extractors of ELF-Nets include multiple ELF convolution layers.

Let us first propose the ELF convolution and then describe the detailed structures of the ELF-Nets. Also, we compare the ELF convolution with SpiderConv [15], which is another operation for point cloud data based on the convolution.

A. ELF CONVOLUTION

To develop a feature extraction operation for point cloud data, we mimic the convolution, which is widely used in CNNs [6]–[10] and has had remarkable success in deep learning tasks for 2D images. To explain the motivation behind the design of the ELF convolution, we first describe the 2D convolution briefly.

1) CONVOLUTION ON REGULAR GRID

A color image can be represented as a function on a regular grid $\mathbf{f} : \mathbb{Z}^2 \rightarrow \mathbb{R}^3$. Figure 3 illustrates the convolution on an image, in which $\mathbf{p}$ is a center pixel and $\mathcal{N}$ denotes its neighborhood, *i.e.* the set of neighboring pixels. Here, $\mathbf{q}$ is a relative position vector between the center pixel position and a neighboring pixel position. For instance, $\mathbf{q}_1 = (1, 1)$

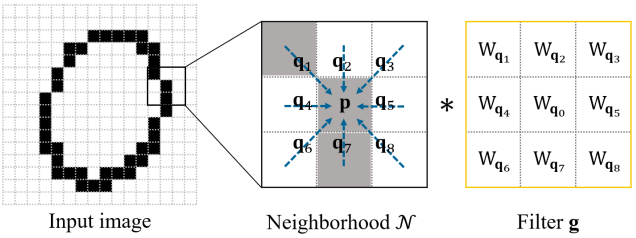

FIGURE 3. Convolution on a 2D image: Relative position vectors are depicted by blue dashed arrows. Here, $\mathbf{q}_1 = (1, 1)$, $\mathbf{q}_2 = (1, 0), \ldots, \mathbf{q}_8 = (-1, -1)$, and $\mathbf{q}_0 = (0, 0)$.

represents the relative position between a center point and its upper left neighbor. The convolution returns the sum of products of convolutional filter coefficients and image pixels in the neighborhood $\mathcal{N}$ as follows.

$$(\mathbf{f} * \mathbf{g})(\mathbf{p}) = \sum_{\mathbf{p}-\mathbf{q}_i \in \mathcal{N}} \mathbf{g}^{\mathrm{T}}(\mathbf{q}_i)\mathbf{f}(\mathbf{p} - \mathbf{q}_i) \tag{1}$$

where $\mathbf{g} : \mathbb{Z}^2 \to \mathbb{R}^{3\times d}$ is a convolutional filter. Hence, the convolution output $\mathbf{f}*\mathbf{g}$ is a d-channel feature vector. Note that, in a CNN, for each $\mathbf{q}_i$,

$$\mathbf{g}(\mathbf{q}_i) = W_{\mathbf{q}_i} \tag{2}$$

which is a $3 \times d$ matrix composed of trainable parameters. This convolutional filter $\mathbf{g}$, which consists of matrices $W_{\mathbf{q}_i}$ in (2), is optimized by the backpropagation algorithm [47].

2) CONVOLUTION ON POINT CLOUD

A point cloud is a set of unordered points in the 3D Euclidean space, where each point $\mathbf{p}$ is a vector of (x, y, z) coordinates. The point $\mathbf{p}$ has its feature vector $\mathbf{f}(\mathbf{p})$, representing its properties, *e.g.*, color or normal. For $\mathbf{p}$, we define its neighborhood as

$$\mathcal{N} = \{\mathbf{p} - \mathbf{q}_i\}_{i=0,\ldots,k-1}, \tag{3}$$

which includes the k nearest neighbors. In (3), we have

$$0 = \|\mathbf{q}_0\| \leq \cdots \leq \|\mathbf{q}_{k-1}\|. \tag{4}$$

Note that $\mathbf{q}_0 = (0, 0, 0)$ and a point is regarded as the nearest neighbor to the point itself. Then, we can use (1) to model the convolution on point clouds as follows.

$$\begin{aligned}(\mathbf{f} * \mathbf{g})(\mathbf{p}) &= \sum_{\mathbf{p}-\mathbf{q}_i \in \mathcal{N}} \mathbf{g}^{\mathrm{T}}(\mathbf{q}_i)\mathbf{f}(\mathbf{p} - \mathbf{q}_i) \\ &= \sum_{i=0}^{k-1} \mathbf{g}^{\mathrm{T}}(\mathbf{q}_i)\mathbf{f}(\mathbf{p} - \mathbf{q}_i).\end{aligned} \tag{5}$$

In this case, $\mathbf{f}$ is a c-channel input feature, and $\mathbf{g}$ is a filter to yield a d-channel output feature.

Different from the regular convolution, points in $\mathcal{N}$ are irregularly distributed. Thus, the translation invariance in the regular convolution does not hold anymore. Also, in the regular convolution in (1), the neighboring pixel $\mathbf{f}(\mathbf{p} - \mathbf{q})$ is multiplied by the filter coefficient matrix $\mathbf{g}(\mathbf{q}) = W_{\mathbf{q}}$ in (2). This is possible because $\mathbf{q}$ is defined on a discrete regular grid within a finite rectangle. In contrast, in the case of point cloud data, the relative position $\mathbf{q}_i$ is an arbitrary vector with continuous components. Thus, there are infinitely many possibilities for $\mathbf{q}_i$, and it is infeasible to define and train a matrix for each possible $\mathbf{q}_i$. Hence, the filter $\mathbf{g}$ should be designed by taking this irregularity of point clouds into consideration.

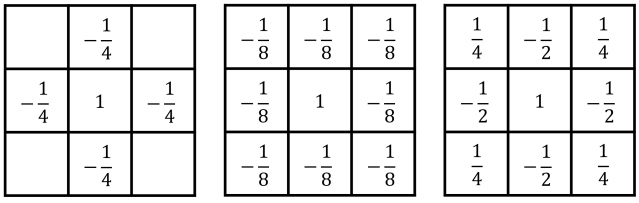

FIGURE 4. Three examples of discrete Laplacian filters [26].

3) EXTENDED LAPLACIAN FILTER

To alleviate the aforementioned problem, we design the filter $\mathbf{g}$ by extending discrete Laplacian filters, which are widely used in 2D image processing [26]. Figure 4 shows three such Laplacian filters, which approximate the second-order derivative in the discrete domain. Note that the output of the Laplace filters quantifies how different the center pixel is from its local neighbors. Contrary to smoothing filters (*e.g.* Gaussian or averaging filters), the Laplacian filters are sensitive to local variations in a signal and thus can be used to extract local features effectively. Also, as discussed below, the Laplacian filters use only a few parameters. Therefore, they can be easily extended to learnable filters for point cloud processing, which require only a moderate number of parameters.

In Figure 4, even though exact filter coefficients are different in the three cases, they have the common design principles.

1) They use a two-state variable: 1 for the center pixel and -1 for the neighboring pixels.

2) They assign weights to the neighboring pixels, and the total sum of the weights is 1. For example, in Figure 4(b), the weight for each 8-neighbor is $\frac{1}{8}$. Also, in Figure 4(c), the weight for each side-sharing neighbor is $\frac{1}{2}$ and that for each corner-sharing neighbor is $-\frac{1}{4}$.

Then, each Laplacian filter coefficient is determined by multiplying the state variable with a weight. Also, note that they are all isotropic filters. In other words, the filter coefficient for pixel $\mathbf{p} - \mathbf{q}$ is dependent only on the distance $\|\mathbf{q}\|$ from the center pixel $\mathbf{p}$.

Following these design principles, we introduce a novel filter $\mathbf{g}$, called ELF, which is given by

$$\mathbf{g}(\mathbf{q}_i) = s_{\mathcal{N}}(i)\mathbf{h}(\mathbf{q}_i) \tag{6}$$

where $s_{\mathcal{N}}$ is a scalar weighting function and $\mathbf{h}$ is a two-state filter. An overview of ELF is shown in the upper part of Figure 5.

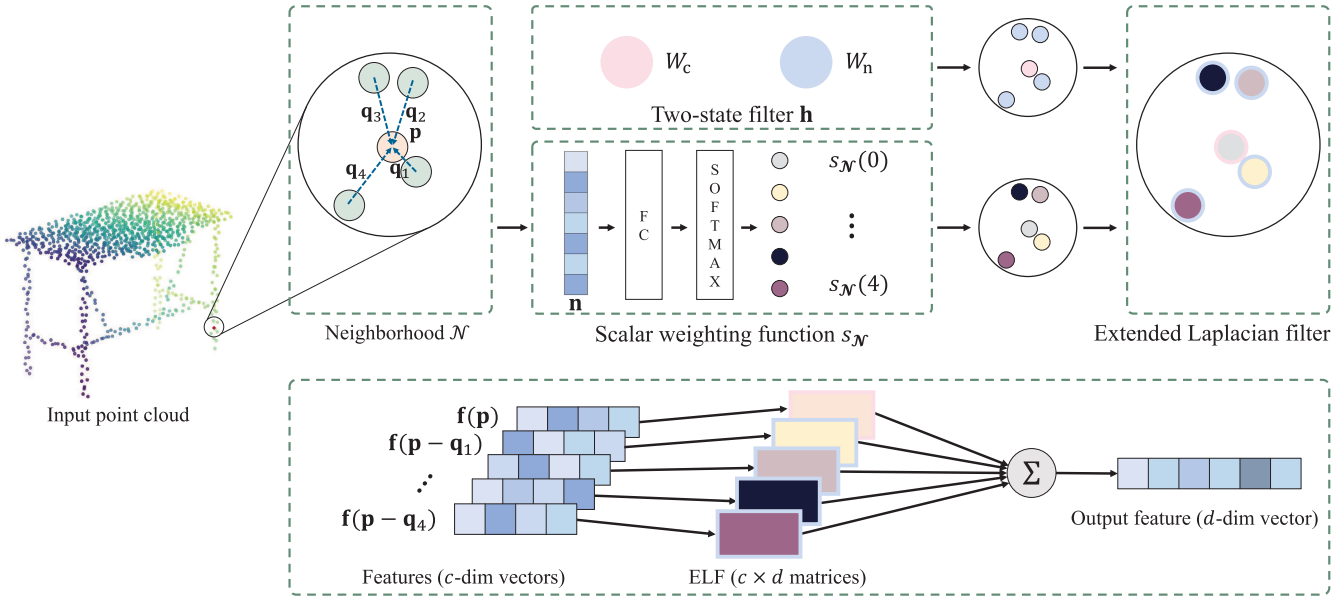

FIGURE 5. Illustration of the ELF convolution: (1) The upper part shows how to determine ELF coefficients. In this example, the neighborhood size k is 5, and $\mathcal{N} = \{\mathbf{p} = \mathbf{p} - \mathbf{q}_0, \mathbf{p} - \mathbf{q}_1, \mathbf{p} - \mathbf{q}_2, \mathbf{p} - \mathbf{q}_3, \mathbf{p} - \mathbf{q}_4\}$. The two-state filter $\mathbf{h}$ yields W_c for the center pixel $\mathbf{p}$ and W_n for the other four points, while the scalar weighting function $s_{\mathcal{N}}$ represents the relative importance of the points. By multiplying the two-state filter output with the scalar weights, the ELF coefficients for those five points are obtained. (2) The lower part summarizes the computation flow of the ELF convolution. By multiplying the c-dimensional input features with the corresponding ELF coefficients and then summing up the products, we obtain the d-dimensional output feature.

First, the two-state filter $\mathbf{h}$ is given by

$$\mathbf{h}(\mathbf{q}_i) = \begin{cases} W_c & \text{if } i = 0, \\ W_n & \text{otherwise.} \end{cases} \tag{7}$$

where W_c and W_n are $c \times d$ trainable matrices for the center point $\mathbf{p}$ and its other neighbors, respectively. Since we use only two states (*i.e.* only two matrices) to define the filter $\mathbf{h}$ in (7), it can be easily trained by the backpropagation in an end-to-end manner. As illustrated in Figure 5, $\mathbf{h}(\mathbf{q}_0) = W_c$ is used for the center pixel, while $\mathbf{h}(\mathbf{q}_1) = \cdots = \mathbf{h}(\mathbf{q}_{k-1}) = W_n$ is for the other neighbors. This two-state filter follows Principle **1** and is similar to the Laplacian filters in Figure 4(a) and (b), where the center pixel is multiplied by a positive number and the neighboring pixels are by a negative number.

Notice that, in Figure 4(c), neighboring pixels are multiplied by either $-\frac{1}{2}$ or $\frac{1}{4}$ depending on their distances from a center pixel. These coefficients can be considered as weighted values of -1, such that the total sum of all such weights for the neighboring pixels is 1. Similarly, following Principle **2**, the two-state filter $\mathbf{h}$ in (6) is modulated by the scalar weighting function $s_{\mathcal{N}}$. It is determined from the distribution of the points in the neighborhood, as shown in Figure 5, and represents the relative importance of each neighbor. First, for the center pixel,

$$s_{\mathcal{N}}(0) = 1. \tag{8}$$

Also, let $\mathbf{n} = [\mathbf{p}^{\mathrm{T}}, \mathbf{q}_1^{\mathrm{T}}, \ldots, \mathbf{q}_{k-1}^{\mathrm{T}}]^{\mathrm{T}}$ represent the neighborhood $\mathcal{N}$. Then, for the other neighbors, we have

$$s_{\mathcal{N}}(i) = \frac{e^{\mathbf{w}_i^{\mathrm{T}}\mathbf{n}}}{\sum_{j=1}^{k-1} e^{\mathbf{w}_j^{\mathrm{T}}\mathbf{n}}} \quad \text{for } 1 \le i \le k-1 \tag{9}$$

where $\mathbf{w}_i$, $1 \le i \le k-1$, are trainable vectors. Note that the weights are normalized so that

$$\sum_{i=1}^{k-1} s_{\mathcal{N}}(i) = 1. \tag{10}$$

Finally, by combining (5)∼(9), we have

$$(\mathbf{f} \star \mathbf{g})(\mathbf{p}) = W_c^{\mathrm{T}}\mathbf{f}(\mathbf{p}) + \sum_{i=1}^{k-1} s_{\mathcal{N}}(i) W_n^{\mathrm{T}}\mathbf{f}(\mathbf{p} - \mathbf{q}_i) \tag{11}$$

where $\star$ denotes the ELF convolution operator. The computation in (11) is illustrated in the lower part of Figure 5. Similarly to the Laplacian filtering, the center pixel is multiplied by W_c, while the neighbors are multiplied by weighted W_n.

All computations in (11), including matrix multiplication and softmax, are differentiable. Hence, ELF is easy to implement, and its parameters W_c, W_n, and $\mathbf{w}_i$ can be trained by the backpropagation algorithm. To summarize, we adopt the design principles of 2D Laplacian filters and extend them to design a trainable filter, called ELF, for unordered point cloud data. ELF can be plugged into existing point processing networks, such as PointNet [13] and PointNet++ [14], to improve their performances.

B. ELF-NETS

We develop ELF-Nets based on the ELF convolution to process unordered points for 3D object classification and per point semantic segmentation problems. Figure 6 shows detailed architectures of ELF-Nets.

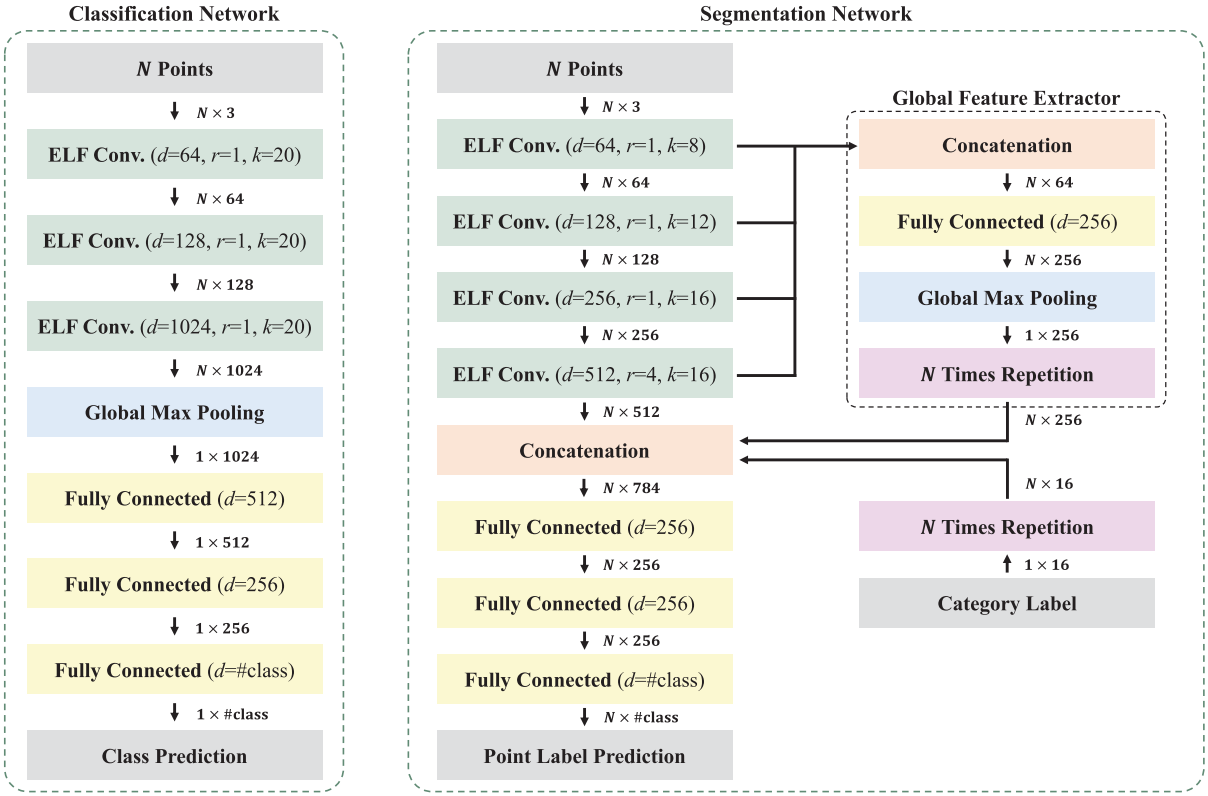

FIGURE 6. The proposed ELF-Nets for classification (left) and part segmentation (right). Input and output data are depicted by gray boxes. Operations are colored to represent their types. For ELF convolution and fully connected layers, d denotes the output feature dimension. Also, in an ELF convolution layer, r is a dilation rate and k is the number of points in the neighborhood. In the segmentation task, the category label for an input point cloud is given as the one-hot vector in $\mathbb{R}^{16}$, since there are 16 object categories in the part segmentation dataset [28].

1) CLASSIFICATION NETWORK

We design the classification network by substituting the multi-layer perceptron layers of PointNet [13] with ELF convolution layers. Given a point cloud, its features are extracted by three ELF layers, aggregated via the global max-pooling, propagated to three fully connected layers, and then categorized into one of pre-defined object classes. Compared to the previous methods in [14]–[16], [20], [29], [40], and [48], the proposed ELF-Net for classification has a simpler architecture but provides better or comparable performance.

2) SEGMENTATION NETWORK

In the case of part segmentation, we employ a similar architecture to [13], [15]. Different from object classification, the segmentation task requires per-point prediction. Thus, each point feature should convey local information, as well as global information, for accurate prediction.

First, we adopt a global feature extractor. It takes the concatenation of the output of the ELF layers as input, fuses the information using a fully connected layer, and then aggregates the pointwise features through channel-wise max pooling.

Next, we concatenate this global feature, the pointwise local features from the fourth ELF convolution layer, and the category information as done in [13] and [15]. Since the objective is to predict part classes, not an object category, the category label for an input point cloud is used to suppress irrelevant predictions. We use three fully connected layers, which take each point feature as input, in order to estimate the part class label of the point.

Also, as [49] adopt the astrous convolution for semantic segmentation to exploit a larger receptive field using the same number of parameters, we implement each ELF convolution layer by collecting the $k \times r$ nearest points and then sampling k points uniformly to build the neighborhood. Here r is a dilation rate.

Both ELF-Nets for classification and segmentation are trained to minimize the standard cross entropy loss.

C. COMPARISON WITH SPIDERCONV

SpiderConv [15] is another operation, based on the convolution, for extracting features from a point cloud. Let us discuss how the proposed ELF is different from SpiderConv. The main difference between SpiderConv and ELF lies in the design of the filter $\mathbf{g}$. In SpiderConv, the filter is defined as

$$\mathbf{g}(\mathbf{q}) = \mathbf{g}^{\text{taylor}}(\mathbf{q})\mathbf{g}^{\text{step}}(\mathbf{q}) \tag{12}$$

TABLE 1. Accuracies on ModelNet40 [17] according to different design choices for ELF.

Design choice	Accuracy (%)
k filters	88.4
One-state filter $\mathbf{h}_1$	89.0
Three-state filter $\mathbf{h}_3$	89.7
w/o $s_{\mathcal{N}}$	90.1
Alternative $s_{\mathcal{N}}$	90.4
Proposed ELF	**91.2**

where

$$\begin{aligned}\mathbf{g}^{\text{taylor}}(x,y,z) = {} & W_0^{\text{t}} + xW_1^{\text{t}} + yW_2^{\text{t}} + zW_3^{\text{t}} \\ & + x^2W_4^{\text{t}} + y^2W_5^{\text{t}} + z^2W_6^{\text{t}} \\ & + xyW_7^{\text{t}} + yzW_8^{\text{t}} + zxW_9^{\text{t}} \\ & + xy^2W_{10}^{\text{t}} + x^2yW_{11}^{\text{t}} + yz^2W_{12}^{\text{t}} \\ & + y^2zW_{13}^{\text{t}} + zx^2W_{14}^{\text{t}} + z^2xW_{15}^{\text{t}} \\ & + x^3W_{16}^{\text{t}} + y^3W_{17}^{\text{t}} + z^3W_{18}^{\text{t}} + xyzW_{19}^{\text{t}} \end{aligned} \tag{13}$$

and

$$\mathbf{g}^{\text{step}}(\mathbf{q}) = W_i^{\text{s}} \quad \text{if } \|\mathbf{q}_i\| \le \|\mathbf{q}\| < \|\mathbf{q}_{i+1}\|. \tag{14}$$

In the Taylor expansion in (13), twenty matrices $W_0^{\text{t}}, \cdots, W_{19}^{\text{t}}$ of size $c \times cT$ are used. In the step function in (14), k matrices $W_0^{\text{s}}, \cdots, W_{k-1}^{\text{s}}$ of size $cT \times d$ are used. Here, T is a hyper-parameter, which is set to 3 for the classification task.

To obtain the filter, SpiderConv employs many trainable matrices and interpolates them using the relative location $\mathbf{q} = (x, y, z)$. In comparison, the proposed ELF convolution uses only two matrices based on the Laplacian filtering. Therefore, for example, when $k = 20$, the input feature dimension $c = 64$, and the output feature dimension $d = 128$, the ELF convolution needs 17,524 parameters, while SpiderConv uses 491,620 parameters. In other words, the parameter requirement of the ELF convolution is only 3.5% of that of SpiderConv. However, as will be shown in Section IV, the ELF convolution provides better or comparable performance than SpiderConv.

IV. EXPERIMENTAL RESULTS

A. CLASSIFICATION ON MODELNET40

ModelNet40 [17] consists of 12,311 mesh models from 40 object categories, which are divided into 9,843 training models and 2,468 test models. As in [13], we uniformly sample 1,024 points from mesh faces and normalize them into a unit sphere. Also, we perform geometric transformation to augment data [13]. More specifically, random horizontal rotation is applied and perturbation noise clipped to 0.05 is added. For training, we use the Adam optimizer [50] with an initial learning rate of 0.001. The training is iterated for 200 epochs. We decrease the learning rate by a factor of 0.5 every 20 epochs. We implement the ELF-Net in TensorFlow [51] and the training takes 10–12 hours using a GTX TITAN X GPU.

TABLE 2. Comparison of accuracies on ModelNet40 [17]. † means our experimental result using the source codes distributed by the authors.

Models	Input	Acc (%).
A. Non Point-based Networks		
Subvolume [12]	Voxels	89.2
OctNet [29]	Hybrid grid octree	86.5
ECC [30]	Graphs	87.4
3D ShapeNets [17]	Voxels	84.7
VoxNet [16]	Voxels	85.9
Kd-Net [20]	1024 points	90.6
B. Point-based Networks		
PointNet w/o T-Net [13]	1024 points	87.1
PointNet [13]	1024 points	89.2
PointNet++ [14]	5000 points + Normal	91.8
PointGrid [52]	1024 points	92.0
SpecGCN [45]	1024 points + Normal	91.8
Pointwise [40]	2048 points	86.1
KCNet [38]	1024 points	91.0
MRTNet [48]	4000 points	91.7
SpiderCNN [15]	1024 points	90.5
SpiderCNN† [15]	1024 points + Normal	91.9
ELF-Net	1024 points	91.2
ELF-Net*	1024 points + Normal	**92.4**

TABLE 3. Accuracies on ModelNet40 [17]. The thee numbers within parentheses mean the hyper-parameter k setting for the first, second and third ELF layers.

k	Accuracy (%)
(12, 12, 12)	90.9
(16, 16, 16)	90.4
(12, 16, 20)	90.7
(8, 16, 32)	89.8
(20, 20, 20)	91.2

To analyze the efficacy of our design choice for ELF, we perform five ablation studies. In Table 1, 'k filters' sorts the k nearest neighbors according to their distances to a center point $\mathbf{p}$. Then, it simply exploits k different filter coefficient matrices according to the sorted indices and the output feature for $\mathbf{p}$ is obtained by

$$\sum_{i=1}^{k} W_i^{\text{T}} \mathbf{f}(\mathbf{p} - \mathbf{q}_i) \tag{15}$$

where W_i is the filter coefficient matrix for the ith neighbor point. Also, instead of using two-state filter function $\mathbf{h}$, we verify alternative options: one-state filter function $\mathbf{h}_1(\mathbf{q}_i) = W$, which yields the same filter coefficient matrix for all points in $\mathcal{N}$, and three-state filter function $\mathbf{h}_3$, which is given by

$$\mathbf{h}_3(\mathbf{q}_i) = \begin{cases} W_{\text{c}} & \text{if } i = 0, \\ W_{\text{n}_1} & \text{if } 1 \le i < \frac{k}{2}, \\ W_{\text{n}_2} & \text{if } \frac{k}{2} \le i < k. \end{cases} \tag{16}$$

Note that the three-state filter function $\mathbf{h}_3$ dichotomizes neighboring points into relatively close ones and the others similarly to Figure 4(c). Furthermore, we confirm the effectiveness of the weighting function $s_{\mathcal{N}}$. In Table 1, we do not

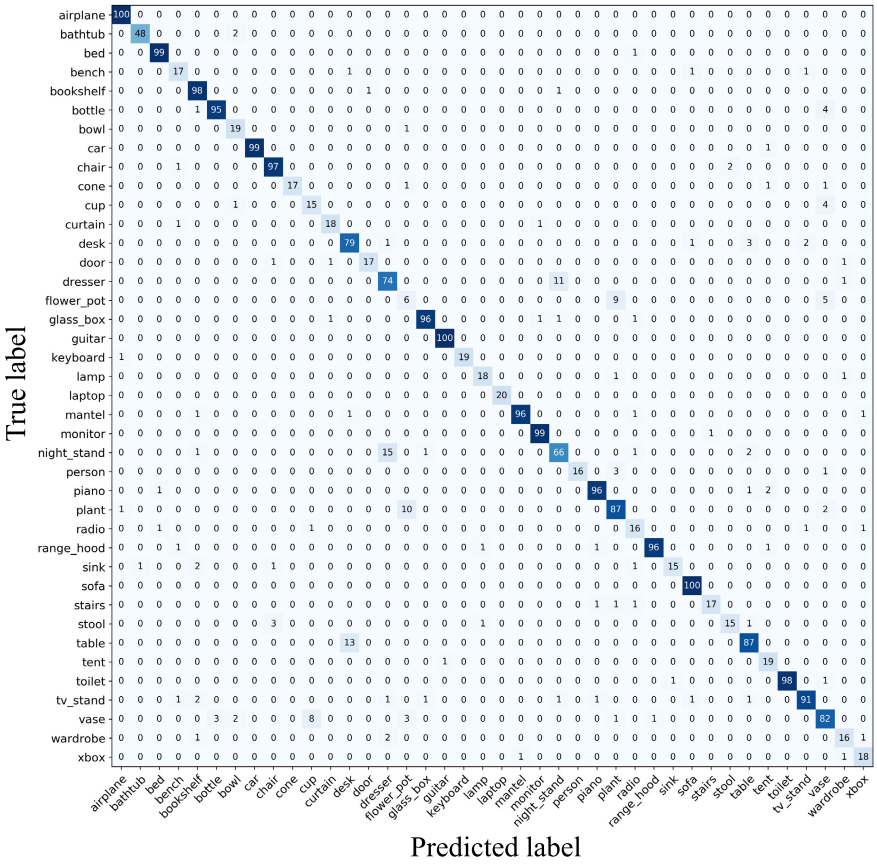

FIGURE 7. The confusion matrix for the classification results on ModelNet40 [17].

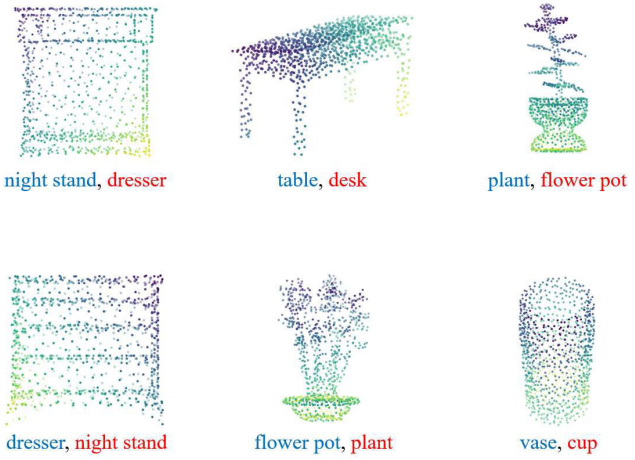

FIGURE 8. Top-6 misclassified cases of the ELF-Net classifier on the ModelNet40 dataset [17]. Ground-truth and predicted classes are reported in blue and red, respectively. Note that even a human being cannot estimate the ground-truth classes reliably.

FIGURE 9. Sample models in the 'cat' category in the SHREC15 dataset [27].

employ $s_{\mathcal{N}}$ for 'w/o $s_{\mathcal{N}}$', *i.e.* we set $\mathbf{g}(\mathbf{q}_i) = \mathbf{h}(\mathbf{q}_i)$. Also, we adopt a different weighting function for 'Alternative $s_{\mathcal{N}}$', which is given by

$$s_{\mathcal{N}}(i) = e^{-\|\mathbf{q}_i\|_2^2} / \sum_{j=1}^{k-1} e^{-\|\mathbf{q}_j\|_2^2}. \tag{17}$$

In other words, we set the weights to be inversely proportional to the distances of neighbors.

We see that the proposed ELF outperforms all the other choices in Table 1, which indicates that our design choice is more effective than the alternative ones. Notably, 'k filters' significantly degrades the performance, even though it uses more coefficient matrices. This is because it considers only the order of neighbor points and thus cannot handle irregular data effectively. In contrast, ELF provides better performance, although it uses only two matrices. As for the alternative choices for $\mathbf{h}$, the one-state filter $\mathbf{h}_1$ performs worse than both ELF and the three-state filter $\mathbf{h}_3$. Thus, it can be concluded that the center point and its other neighbors had better be processed with different coefficient matrices. However, ELF outperforms $\mathbf{h}_3$, since the detailed distinction

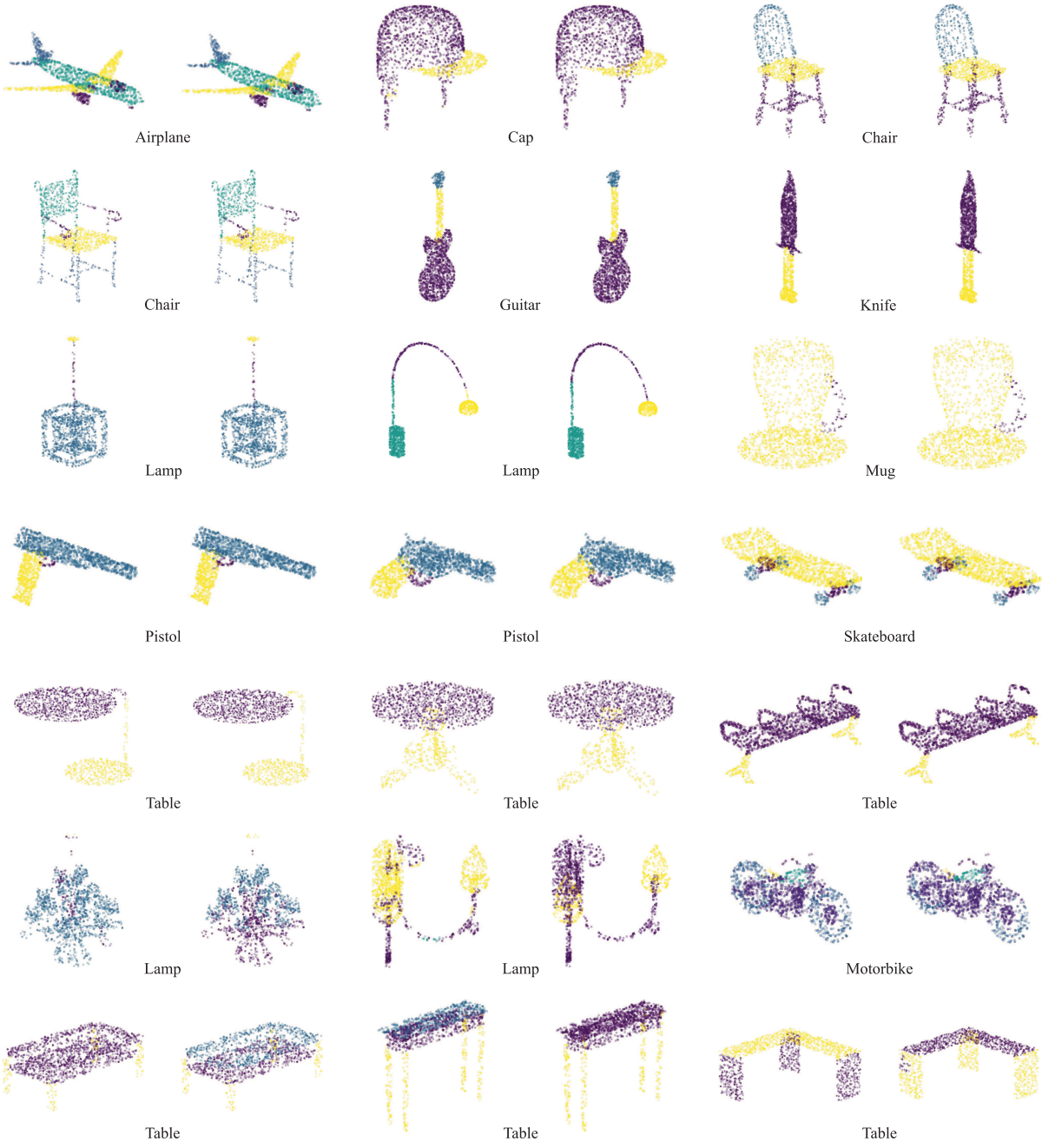

FIGURE 10. Examples of 3D object part segmentation results on ShapeNetPS [28]. For each model, the left shows the ground-truth, while the right is the segmentation result of the proposed ELF-Net. The last two rows present relatively inaccurate segmentation results for more challenging models.

of neighbors of (16) makes the optimization harder. The lower accuracies of both 'w/o $s_{\mathcal{N}}$' and 'Alternative $s_{\mathcal{N}}$' indicate that the proposed weighting function $s_{\mathcal{N}}$ performs well by analyzing the configuration of the neighborhood.

Table 2 compares the proposed ELF-Net with conventional algorithms [12]–[17], [20], [29], [30]. Note that we design the ELF-Net, by replacing the multi-layer perceptron layers in PointNet (without T-Net) [13] with ELF layers. Thus, the performance gap of 4.1% between ELF-Net and 'PointNet w/o T-Net' demonstrates that ELF processes unordered points more effectively than the multi-layer perceptron. The ELF-Net achieves the accuracy of 91.2%, which is comparable to all conventional algorithms taking 1,024 points as input. The performance of the ELF-Net is improved when normal vectors are additionally used. Therefore, ELF-Net* outperforms all conventional algorithms in Table 2.

Table 3 compares the accuracies of the ELF-Net with different hyper-parameter settings of k. We see that ELF with $k = 20$ is a better choice than the other options.

TABLE 4. Classification accuracies on the SHREC15 dataset [27]. † means our experimental result using the source codes distributed by the authors.

Models	Accuracy (%)
PointNet† [13]	65.0
PointNet++ [14]	60.2
SpiderCNN† [15]	85.6
ELF-Net	**87.3**

Figure 7 is the confusion matrix for the classification results on ModelNet40. The diagonal elements are dominant, which indicates that ELF-Net performs classification well regardless of object classes. The most confusion occurs when 'night stand' is misclassified into 'dresser,' 'table' is into 'desk,' and 'dresser' is to 'night stand.' Figure 8 shows examples of top-6 misclassified cases of the proposed ELF-Net classifier. Most of them are very hard, and even a human being cannot tell their ground-truth classes reliably.

B. CLASSIFICATION ON SHREC15

SHREC15 [27] is a dataset for non-rigid 3D shape retrieval. It contains 1,200 watertight triangle mesh models from 50 categories: each category has 24 models. In each category, 6 models are randomly selected to compose a test data set. As in ModelNet40, to generate a point cloud, 1,024 points are uniformly sampled from mesh faces and normalized into a unit sphere. We adopt five-fold cross validation to evaluate the classification performance [14]. Compared to ModelNet40, SHREC15 is more challenging, since 1,140 models are generated by deforming 60 original models. Hence, as shown in Figure 9, it contains models of the same category in different poses. To classify these models correctly, it is necessary to understand not only global shapes but also local details.

We employ the same ELF-Net architecture and data augmentation strategy used in the ModelNet40 experiment. We use the Adam optimizer with an initial learning rate of 0.001. The training is done iteratively for 100 epochs.

Table 4 reports classification accuracies on the SHREC15 dataset. The proposed algorithm outperforms the conventional methods in [13]–[15] by meaningful margins. This indicates that the ELF-Net analyzes complicated geometrical shapes in a 3D space effectively and extracts discriminative features, which are robust against pose variations.

C. PART SEGMENTATION ON SHAPENETPS

The ShapeNet part segmentation dataset (ShapeNetPS) [28] contains 16,881 point clouds from 16 object categories, each annotated with two to six part labels. 14,007 point clouds are for training, and 2,874 for test. Each point cloud includes annotated points, which are assigned part categories, such as chair leg and cup handle. There are 50 different part categories in total. As done in PointNet [13], we employ the intersection over union (IoU) ratio to evaluate part segmentation performance. To train the ELF-Net for classification, we utilize the same training process in Section IV-A, except that 2,048 points are used as input.

Table 5 compares the performance of the proposed ELF-Net with those of the conventional algorithms in [13]–[15], [20], [21], [25], [28], and [31]. Among the conventional algorithms, SpiderCNN [15] and PointNet++ [14] yield the state-of-the-art performances. The ELF-Net yields comparable or better results than these state-of-the-arts. More specifically, in terms of the average IoU, the ELF-Net is comparable to SpiderCNN and better than all other methods. Especially, the ELF-Net achieves the best performances on the five categories of 'chair,' 'knife,' 'laptop,' 'mug,' and 'rocket' and the second best performances on the 'airplane,' 'car,' and 'motorbike' categories. For the 'earphone,' 'motorbike,' and 'rocket' categories, ELF-Net provides relatively low IoUs. This is mainly because there are only few training samples for those categories, which are 49, 125, and 46, respectively. Note that 'rocket' is the most difficult category, on which all algorithms yield poor IoUs. However, the ELF-Net still outperforms the conventional algorithms on this challenging category.

Figure 10 shows some segmentation examples. We see that the ELF-Net provides faithful segmentation results.

D. ELF-NET ANALYSIS

1) ROBUSTNESS TEST

Point clouds in the real world, which are acquired by LiDAR scanners, tend to be noisier than synthetic point clouds in ModelNet40, SHREC15, and ShapeNetPS. Moreover, their densities may vary according to the scanning conditions. Therefore, it is important to evaluate the robustness against various kinds of input corruption for the practical usage.

For the density change, we randomly sample input points as illustrated in Figure 11(a). Figure 12(a) shows accuracies of the proposed ELF-Net and PointNet [13] on ModelNet40 [17] according to the number of input points, where both networks take points and normal vectors as their input. In Figure 12(a), the performance drop of ELF-Net is only 2.8 when the number of points is decreased by 88%. The ELF-Net outperforms PointNet in all cases without exception.

To evaluate the robustness against noise, we generate Gaussian noise and add it to each point independently as shown in Figure 11(b). Figure 12(b) plots the accuracy according to the noise level. The ELF-Net achieves 86.5% even when point cloud are blurred by severe noise with the standard deviation of 0.02. Also, the ELF-Net outperforms PointNet at all noise levels. These results indicate that the proposed ELF-Net is robust to various input corruptions, even though it has a simple structure.

2) FEATURE VISUALIZATION

In Figure 13, we visualize what the filters in the first ELF layer in the ELF-Net for classification captures. We randomly

TABLE 5. Comparison of IoU scores on ShapeNetPS [28]. The best result is boldfaced, while the second best is underlined.

Category	[28]	[38]	[25]	[21]	[13]	[14]	[20]	[31]	[15]	ELF-Net
Airplane	81.0	82.8	82.7	82.8	83.4	82.4	80.1	81.6	**83.5**	83.4
Bag	78.4	81.5	**86.4**	77.8	78.7	79.0	74.6	81.7	81.0	78.2
Cap	77.7	86.4	84.1	**88.0**	82.5	87.7	74.3	81.9	87.2	86.2
Car	75.7	77.6	**78.2**	77.3	74.9	77.3	70.3	75.2	77.5	77.6
Chair	87.9	90.3	90.4	90.6	89.6	**90.8**	88.6	90.2	90.7	**90.8**
Earphone	62.9	**76.8**	69.3	73.5	73.0	71.8	73.5	74.9	**76.8**	71.0
Guitar	92.0	91.0	91.4	90.7	91.5	91.0	90.2	**93.0**	91.1	91.3
Knife	85.4	87.2	87.0	83.9	85.9	85.9	87.2	86.1	87.3	**87.6**
Lamp	82.5	84.5	83.5	82.8	80.8	83.7	81.0	**84.7**	83.3	83.4
Laptop	95.7	95.5	95.4	94.8	95.3	95.3	94.9	95.6	95.8	**95.9**
Motorbike	70.6	69.2	66.0	69.1	65.2	**71.6**	57.4	66.7	70.2	70.6
Mug	91.9	94.4	92.6	94.2	93.0	94.1	86.7	92.7	93.5	**94.5**
Pistol	**85.9**	81.6	81.8	80.9	81.2	81.3	78.1	81.6	82.7	81.4
Rocket	53.1	60.1	56.1	53.1	57.9	58.7	51.8	60.6	59.7	**61.3**
Skateboard	69.8	75.2	75.8	72.9	72.8	76.4	69.9	**82.9**	75.8	76.1
Table	75.3	81.3	82.2	**83.0**	80.6	82.6	80.3	82.1	82.8	82.7
# Best	1	1	2	2	0	2	0	3	2	**5**
Average	81.4	84.7	84.9	84.9	83.7	85.1	82.3	84.7	**85.3**	**85.3**

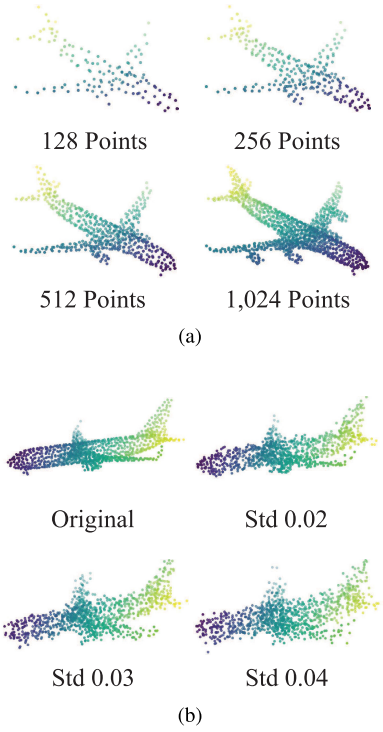

FIGURE 11. Examples of point clouds with different kinds of input corruption: (a) density change (b) Gaussian noise.

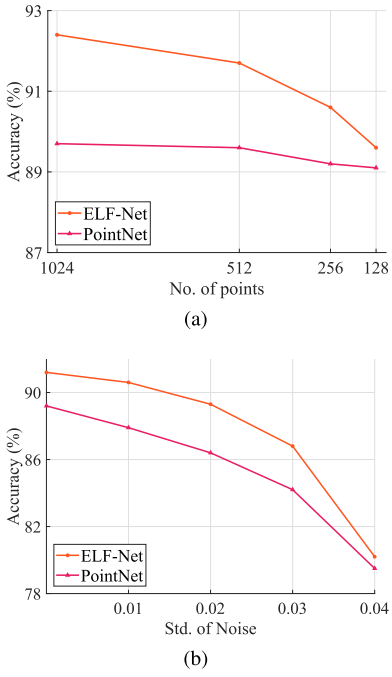

FIGURE 12. Comparison of robustness against (a) density change (b) Gaussian noise.

generate 1,024 points within a unit sphere centered at the origin and find the 150 points with high feature values, which represent the patterns that the filters recognize. We observe that each filter captures different local areas or patterns in the unit sphere.

Figure 14 visualizes important points, which are assigned large weights $s_{\mathcal{N}}$, for some models in ModelNet40 [17]. These points captures object skeletons faithfully. In other words, $s_{\mathcal{N}}$ predicts the relative importance of each point reliably.

3) NETWORK COMPLEXITY

Table 6 reports the complexity of the classification networks. As input, all networks in Table 6 take 1,024 points without normal data. The proposed ELF-Net requires significantly fewer parameters than the conventional networks. Compared to SpiderCNN [15], the ELF-Net uses only 9.5% of parameters. However, as shown in Table 2, the ELF-Net yields 0.7% and 0.5% higher accuracies than SpiderCNN, when 1,024 points and 1,024 points + normal are used as input, respectively. Also, the ELF-Net takes only 7.58ms to classify a point cloud using a GTX TITAN X GPU.

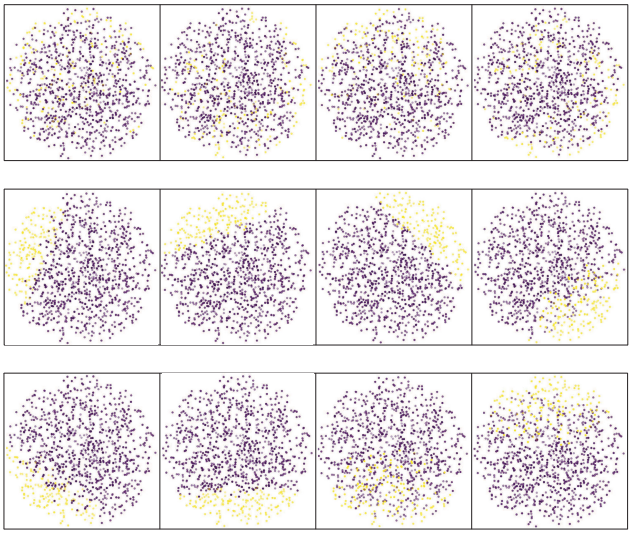

FIGURE 13. Visualization of the patterns captured by the first ELF layer. Points with the top 150 feature values are depicted in yellow. In the second and third rows, we observe that different filters focus on different local regions.

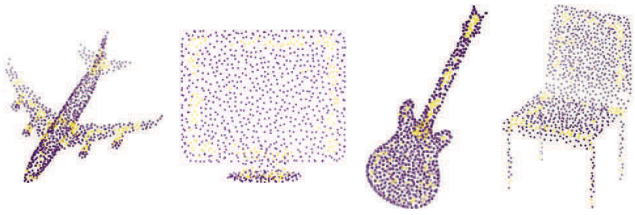

FIGURE 14. Visualization of the impacts of $s_{\mathcal{N}}$. Important points are depicted in yellow.

TABLE 6. Comparison of the numbers of parameters and the floating point operations (FLOP), which are required by the classification networks. M and B stand for million and billion, respectively.

Models	# Parameters	FLOP
Kd-Net (depth 10) [20]	2.0M	
PointNet [13]	3.48M	0.93B
PointNet++ [14]	1.48M	1.69B
SpecGCN [45]	2.05M	
SpiderCNN [15]	5.84M	8.92B
ELF-Net	**0.55M**	5.75B

Table 6 also compares the floating point operations (FLOP), which are carried out in the networks. All these FLOP measurements are done in TensorFlow [51]. The proposed ELF-Net requires a higher FLOP than PointNet or PointNet++, but a lower FLOP than SpiderCNN.

V. CONCLUSION

We proposed ELF-Nets, which are composed of ELF layers, to process and analyze 3D point clouds. We showed that ELF can be interpreted as an extended version of 2D discrete Laplace operators. More specifically, ELF employs two matrices: one for a center point and the other for its neighboring points. The second matrix is scaled using a weighting function, which is determined by the local configuration of the neighborhood. Since it uses only two matrices, ELF is more efficient in terms of memory and computational requirements than the conventional methods. Furthermore, experimental results on popular benchmarks demonstrated that ELF-Nets provide better or comparable performances than the state-of-the-art techniques in both classification and segmentation tasks.

REFERENCES

[1] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3D object detection network for autonomous driving," in *Proc. CVPR*, Jul. 2017, pp. 1907–1915.
[2] M. Menze and A. Geiger, "Object scene flow for autonomous vehicles," in *Proc. CVPR*, Jun. 2015, pp. 3061–3070.
[3] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt, M. Sokolsky, G. Stanek, D. Stavens, A. Teichman, M. Werling, and S. Thrun, "Towards fully autonomous driving: Systems and algorithms," in *Proc. IV*, Jun. 2011, pp. 163–168.
[4] L. C. Hieu, N. Zlatov, J. V. Sloten, E. Bohez, L. Khanh, P.H. Binh, P. Oris, and Y. Toshev, "Medical rapid prototyping applications and methods," *Assem. Automat.*, vol. 25, no. 4, pp. 284–292, 2005.
[5] B. Schwarz, "LIDAR: Mapping the world in 3D," *Nature Photon.*, vol. 4, no. 7, pp. 429–430, 2010.
[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. CVPR*, Jun. 2016, pp. 770–778.
[7] C. Dong, C. C. Loy, K. He, and X. Tang, "Learning a deep convolutional network for image super-resolution," in *Proc. ECCV*, 2014, pp. 184–199.
[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. NIPS*, 2012, pp. 1097–1105.
[9] R. Girshick, "Fast R-CNN," in *Proc. ICCV*, Dec. 2015, pp. 1440–1448.
[10] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Proc. NIPS*, 2015, pp. 91–99.
[11] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, May 2015.
[12] C. R. Qi, H. Su, M. Niessner, A. Dai, M. Yan, and L. J. Guibas, "Volumetric and multi-view CNNs for object classification on 3D data," in *Proc. CVPR*, Jun. 2016, pp. 5648–5656.
[13] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep learning on point sets for 3D classification and segmentation," in *Proc. CVPR*, Jul. 2017, pp. 652–660.
[14] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "PointNet++: Deep hierarchical feature learning on point sets in a metric space," in *Proc. NIPS*, 2017, pp. 5099–5108.
[15] Y. Xu, T. Fan, M. Xu, L. Zeng, and Y. Qiao, "SpiderCNN: Deep learning on point sets with parameterized convolutional filters," in *Proc. ECCV*, Sep. 2018, pp. 87–102.
[16] D. Maturana and S. Scherer, "VoxNet: A 3D convolutional neural network for real-time object recognition," in *Proc. IROS*, Sep./Oct. 2015, pp. 922–928.
[17] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3D ShapeNets: A deep representation for volumetric shapes," in *Proc. CVPR*, Jun. 2015, pp. 1912–1920.
[18] Y. Li, S. Pirk, H. Su, C. R. Qi, and L. J. Guibas, "FPNN: Field probing neural networks for 3D data," in *Proc. NIPS*, 2016, pp. 307–315.
[19] P.-S. Wang, Y. Liu, Y.-X. Guo, C.-Y. Sun, and X. Tong, "O-CNN: Octree-based convolutional neural networks for 3D shape analysis," *ACM Trans. Graph.*, vol. 36, no. 4, p. 72, 2017.
[20] R. Klokov and V. Lempitsky, "Escape from cells: Deep Kd-networks for the recognition of 3D point cloud models," in *Proc. ICCV*, Oct. 2017, pp. 863–872.
[21] J. Li, B. M. Chen, and G. H. Lee, "SO-Net: Self-organizing network for point cloud analysis," in *Proc. CVPR*, Jun. 2018, pp. 9397–9406.
[22] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola, "Deep sets," in *Proc. NIPS*, 2017, pp. 3391–3401.

[23] M. Tatarchenko, J. Park, V. Koltun, and Q.-Y. Zhou, "Tangent convolutions for dense prediction in 3D," in *Proc. CVPR*, Jun. 2018, pp. 3887–3896.
[24] J.-C. Su, M. Gadelha, R. Wang, and S. Maji, "A deeper look at 3D shape classifiers," in *Proc. ECCV*, Sep. 2018, p. 0.
[25] Q. Huang, W. Wang, and U. Neumann, "Recurrent slice networks for 3D segmentation of point clouds," in *Proc. CVPR*, Jun. 2018, pp. 2626–2635.
[26] A. K. Jain, *Fundamentals of Digital Image Processing*. Upper Saddle River, NJ, USA: Prentice-Hall, 1989.
[27] Z. Lian *et al.*, "Non-rigid 3D shape retreival," in *Proc. Eurograph. Workshop DOR*, 2015, pp. 108–120.
[28] L. Yi, V. G. Kim, D. Ceylan, I-C. Shen, M. Yan, H. Su, C. Lu, Q. Huang, A. Sheffer, and L. Guibas, "A scalable active framework for region annotation in 3D shape collections," *ACM Trans. Graph.*, vol. 35, no. 6, p. 210, 2016.
[29] G. Riegler, A. O. Ulusoy, and A. Geiger, "OctNet: Learning deep 3D representations at high resolutions," in *Proc. CVPR*, Jul. 2017, pp. 3577–3586.
[30] M. Simonovsky and N. Komodakis, "Dynamic edge-conditioned filters in convolutional neural networks on graphs," in *Proc. CVPR*, Jul. 2017, pp. 3693–3702.
[31] L. Yi, H. Su, X. Guo, and L. J. Guibas, "SyncSpecCNN: Synchronized spectral CNN for 3D shape segmentation," in *Proc. CVPR*, Jul. 2017, pp. 2282–2290.
[32] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, "Multi-view convolutional neural networks for 3D shape recognition," in *Proc. ICCV*, Dec. 2015, pp. 945–953.
[33] T. Yu, J. Meng, and J. Yuan, "Multi-view harmonized bilinear network for 3D object recognition," in *Proc. CVPR*, Jun. 2018, pp. 186–194.
[34] Y. Feng, Z. Zhang, X. Zhao, R. Ji, and Y. Gao, "GVCNN: Group-view convolutional neural networks for 3D shape recognition," in *Proc. CVPR*, Jun. 2018, pp. 264–272.
[35] A. Kanezaki, Y. Matsushita, and Y. Nishida, "RotationNet: Joint object categorization and pose estimation using multiviews from unsupervised viewpoints," in *Proc. CVPR*, Jun. 2018, pp. 5010–5019.
[36] D. Z. Wang and I. Posner, "Voting for voting in online point cloud object detection," *Robot. Sci. Syst.*, vol. 1, no. 3, p. 15607, 2015.
[37] M. Engelcke, D. Rao, D. Z. Wang, C. H. Tong, and I. Posner, "Vote3Deep: Fast object detection in 3D point clouds using efficient convolutional neural networks," in *Proc. ICRA*, May/Jun. 2017, pp. 1355–1361.
[38] Y. Shen, C. Feng, Y. Yang, and D. Tian, "Mining point cloud local structures by Kernel correlation and graph pooling," in *Proc. CVPR*, Jun. 2018, pp. 4548–4557.
[39] T. Kohonen, "The self-organizing map," *Proc. IEEE*, vol. 78, no. 9, pp. 1464–1480, Sep. 1990.
[40] B.-S. Hua, M.-K. Tran, and S.-K. Yeung, "Pointwise convolutional neural networks," in *Proc. CVPR*, 2018, pp. 984–993.
[41] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," 2013, *arXiv:1312.6203*. [Online]. Available: https://arxiv.org/abs/1312.6203
[42] M. Henaff, J. Bruna, and Y. LeCun, "Deep convolutional networks on graph-structured data," 2015, *arXiv:1506.05163*. [Online]. Available: https://arxiv.org/abs/1506.05163
[43] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proc. NIPS*, 2016, pp. 3844–3852.
[44] R. Levie, M. Federico, X. Bresson, and B. Xavier, "CayleyNets: Graph convolutional neural networks with complex rational spectral filters," *IEEE Trans. Signal Process.*, vol. 67, no. 1, pp. 97–109, Jan. 2019.
[45] C. Wang, B. Samari, and K. Siddiqi, "Local spectral graph convolution for point set feature learning," in *Proc. ECCV*, Sep. 2018, pp. 52–66.
[46] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model CNNs," in *Proc. CVPR*, Jul. 2017, pp. 5115–5124.
[47] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.
[48] M. Gadelha, R. Wang, and S. Maji, "Multiresolution tree networks for 3D point cloud processing," in *Proc. ECCV*, Sep. 2018, pp. 103–118.
[49] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "DeepLab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 4, pp. 834–848, Apr. 2017.
[50] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*. [Online]. Available: https://arxiv.org/abs/1412.6980
[51] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous distributed systems," 2016, *arXiv:1603.04467*. [Online]. Available: https://arxiv.org/abs/1603.04467
[52] T. Le and Y. Duan, "PointGrid: A deep network for 3D shape understanding," in *Proc. CVPR*, Jun. 2018, pp. 9204–9214.

SEON-HO LEE (S'18) received the B.S. degree in electrical engineering from Korea University, Seoul, South Korea, in 2018, where he is currently pursuing the Ph.D. degree. His current research interests include computer vision and machine learning, especially the problems of point cloud compression and 3D deep learning.

HAN-UL KIM (S'14) received the B.S. degree in electrical engineering from Korea University, Seoul, South Korea, in 2014, where he is currently pursuing the Ph.D. degree. His current research interests include computer vision and machine learning.

CHANG-SU KIM (S'95–M'01–SM'05) received the Ph.D. degree in electrical engineering from Seoul National University, with a Distinguished Dissertation Award, in 2000. From 2000 to 2001, he was a Visiting Scholar with the Signal and Image Processing Institute, University of Southern California, Los Angeles, CA, USA. From 2001 to 2003, he coordinated the 3D Data Compression Group with the National Research Laboratory for 3D Visual Information Processing, SNU. From 2003 and 2005, he was an Assistant Professor with the Department of Information Engineering, Chinese University of Hong Kong. In September 2005, he joined the School of Electrical Engineering, Korea University, where he is currently a Professor. He is also an APSIPA Distinguished Lecturer from 2017 to 2018. He has published more than 270 technical articles in international journals and conferences. His current research interests include image processing and computer vision. He is a member of the Multimedia Systems & Application Technical Committee (MSATC) of the IEEE Circuits and Systems Society. He was a recipient of the IEEK/IEEE Joint Award for Young IT Engineer of the Year, in 2009, and the Best Paper Award from the *Journal of Visual Communication and Image Representation* (JVCI), in 2014. He served as an Editorial Board Member of JVCI, an Associate Editor for the IEEE TRANSACTIONS ON IMAGE PROCESSING, a Senior Area Editor for JVCI, and an Associate Editor for the IEEE TRANSACTIONS ON MULTIMEDIA.