Mesh Convolutional Networks With Face and Vertex Feature Operators

Daniel Perez, Yuzhong Shen, *Senior Member, IEEE*, and Jiang Li, *Senior Member, IEEE*

Abstract—Deep learning techniques have proven effective in many applications, but these implementations mostly apply to data in one or two dimensions. Handling 3D data is more challenging due to its irregularity and complexity, and there is a growing interest in adapting deep learning techniques to the 3D domain. A recent successful approach called MeshCNN consists of a set of convolutional and pooling operators applied to the edges of triangular meshes. While this approach produced superb results in classification and segmentation of 3D shapes, it can only be applied to edges of a mesh, which can constitute a disadvantage for applications where the focuses are other primitives of the mesh. In this study, we propose face-based and vertex-based operators for mesh convolutional networks. We design two novel architectures based on the MeshCNN network that can operate on faces and vertices of a mesh, respectively. We demonstrate that the proposed face-based architecture outperforms the original MeshCNN implementation in mesh classification and mesh segmentation, setting the new state of the art on benchmark datasets. In addition, we extend the vertex-based operator to fit in the Point2Mesh model for mesh reconstruction from clean, noisy, and incomplete point clouds. While no statistically significant performance improvements are observed, the model training and inference time are reduced by the proposed approach by 91% and 20%, respectively, as compared with the original Point2Mesh model.

Index Terms—Geometric deep learning, mesh, classification, segmentation, feature selection

1 Introduction

Deep learning has achieved superb results in different applications such as classification, regression, object recognition, etc. [2]. In most cases, deep learning algorithms are applied to 2D data (images) or 1D data (text, audio. etc.). Recently, there has been an increasing interest in applying these techniques to 3D shapes, which constitutes a significantly harder task due to the complexity and irregularity of the data. There have been multiple approaches to apply deep learning models on different representations of 3D data such as 2D projections, voxel grids, point clouds, etc. [3]. Recently, Hanocka *et al*. proposed MeshCNN [1], a convolutional neural network (CNN) based model that produced state-of-the-art results in classification and segmentation of 3D shapes. The authors of this method designed convolutional and pooling operators that take into account the connectivity of the mesh. They proposed an edge-based architecture in which the input consists of features extracted from the edges of the mesh and their neighbors.

One of the main limitations of MeshCNN is that it can only be applied to edges of geometric meshes. While this does not constitute a problem for some tasks (e.g., classification of the whole shapes), it can become an issue when the task is to focus on a different primitive of the mesh. For instance, deforming the edges or faces of a mesh is not as trivial as deforming its vertices due to the connectivity within the shape. Thus, a vertex-based approach would be desirable in a generative problem in which different types of primitives need to be modified. A different example can be a specific project that requires segmentation of faces or vertices of a mesh instead of the edges. While the input and output of MeshCNN can be adapted in all of these cases to intermediate edge-based representations, it is desirable to have networks that can directly handle these primitives.

The main goal of this study is to design two alternative implementations of mesh convolutional networks that can be applied to faces and vertices of a triangular mesh directly while preserving or improving the performance of the original network. Specifically, we propose two novel architectures for mesh convolutional neural networks that can be applied directly to the faces and vertices of a mesh. To achieve this goal, we modify three main operations of MeshCNN including feature extraction, neighborhood selection, and pooling. The main contributions of this study are a face-based and a vertex-based mesh convolutional neural networks that can be applied directly to the faces and vertices of the mesh, respectively. The remainder of this paper is organized as follows. Section 2 provides a review of the relevant literature. Section 3 describes in detail the methodology followed in the design of our networks. Section 4 compares the proposed model with the original edge-based MeshCNN. Section 5 includes a case study that shows how our vertex-based approach performs better in a generative task. Finally, Section 6 discusses the results obtained, and Section 7 draws conclusions from the study.

- *Daniel Perez and Yuzhong Shen are with the Department of Computational Modeling and Simulation Engineering, Old Dominion University, Norfolk, VA 23529 USA. E-mail: {dpere013, yshen}@odu.edu.*
- *Jiang Li is with the Department of Electrical and Computer Engineering, Old Dominion University, Norfolk, VA 23529 USA. E-mail: jli@odu.edu.*

Manuscript received 12 Apr. 2021; revised 20 Oct. 2021; accepted 12 Nov. 2021. Date of publication 18 Nov. 2021; date of current version 31 Jan. 2023.
(Corresponding author: Daniel Perez.)
Recommended for acceptance by J. Tierny.
Digital Object Identifier no. 10.1109/TVCG.2021.3129156

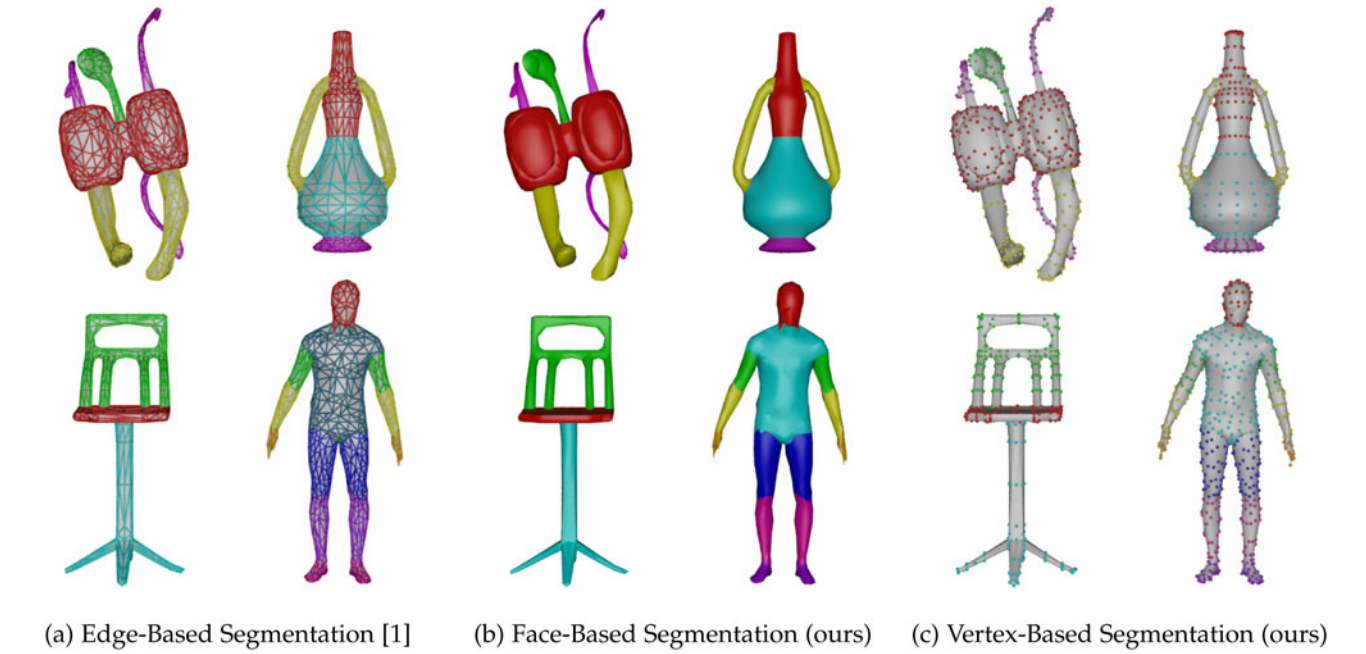

Fig. 1. MeshCNN [1] successfully performs classification and segmentation on the edges of geometric triangular meshes (a). We propose two novel and alternative implementations that operate on the faces (b) and vertices (c) of the meshes for classification and segmentation.

2 Related Work

2.1 Geometric Deep Learning

Deep learning models have achieved superb results in many different applications when applied to 2D data such as images or 1D data such as text or audio. This type of data has a grid-like (i.e., euclidean) structure that makes it very suitable to be fed to models like multilayer perceptrons (MLPs) or CNNs. Recently, there has been an increased interest in applying deep learning models to non-euclidean domains such as graphs, molecules, or 3D shapes. This field is known as geometric deep learning [4].

The term geometric deep learning covers a wide variety of topics. In this paper, we are exclusively interested in deep learning models applied to 3D shapes. One of the first approaches to apply deep learning models to 3D shapes consisted of rendering views of the models from different angles and positions and applying them to traditional 2D models such as CNNs [5], [6]. Another exploratory technique consisted of feeding the models with 3D voxel grids that are analogous to deep image representations in 2D [7], [8], [9]. While these techniques have the advantage of directly using deep learning models that have proven successful in the 2D domain, they do not make use of the geometry or connectivity information of the 3D shapes, and their computational complexity is often extremely high.

Another direction of geometric deep learning focuses on designing deep learning models that can work with point clouds. One of the most popular implementations that follow this approach is PointNet [10]. The authors of this method proposed a network in which a shared MLP layer (i.e., a 1x1 convolution) was applied per vertex to obtain the high-level representations, and a global pooling layer was added at the end of the network to guarantee invariance to the order of the points. An improvement of this model referred to as PointNet++ was proposed by the same team to use the closest neighbors of a vertex as additional information in the model [11]. A similar network was proposed in [12], but in this case, the neighborhood information was dynamically updated per layer based on the distance in the feature space. While these networks benefit from using the geometry information of the 3D shapes, the actual connectivity between the vertices was not fed to the models.

Other works focused on the design of models that can be fed with the connectivity of the shapes. A common technique extracts the Laplacian of the graph representation and operates on the spectral domain of the mesh [13], [14], [15]. The main limitation of this approach is that it generally requires the meshes fed to the model to have a fixed topology, which limits its applicability to cases where all meshes have exactly the same connectivity. Other approaches attempted parameterizing the 3D mesh to a 2D space [16], [17]. However, parameterizing 3D shapes is an arduous process, and, similar to the methods that deal with multi-view and voxel representations, the networks in these approaches do not adapt specifically to the mesh structures. To overcome these issues, a model known as MeshCNN was recently proposed by [1]. The authors of this method designed convolutional and pooling operators that took into account the connectivity of triangular meshes. They proposed an edge-based architecture in which the input consists of features extracted from edges and their neighbors. This method produced state-of-the-art results in classification and segmentation of 3D meshes, and it has been successfully adapted for other tasks such as subdivision [18], generation of geometric textures [19], and point cloud to mesh translation [20]. Section 2.2 describes in greater detail the design of this approach. It is worth mentioning that MeshCNN is not the only deep learning model that can directly take geometric meshes as input. For instance, a method known as MeshWalker [21] can process a 3D mesh by randomly "walking" through vertices of the mesh to

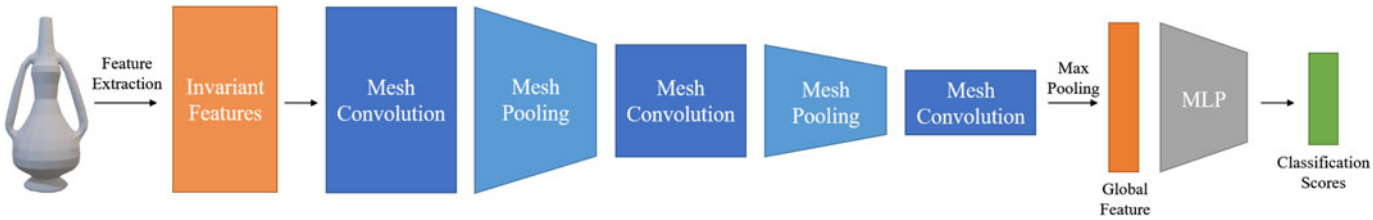

Fig. 2. MeshCNN network for classification of geometric triangular meshes.

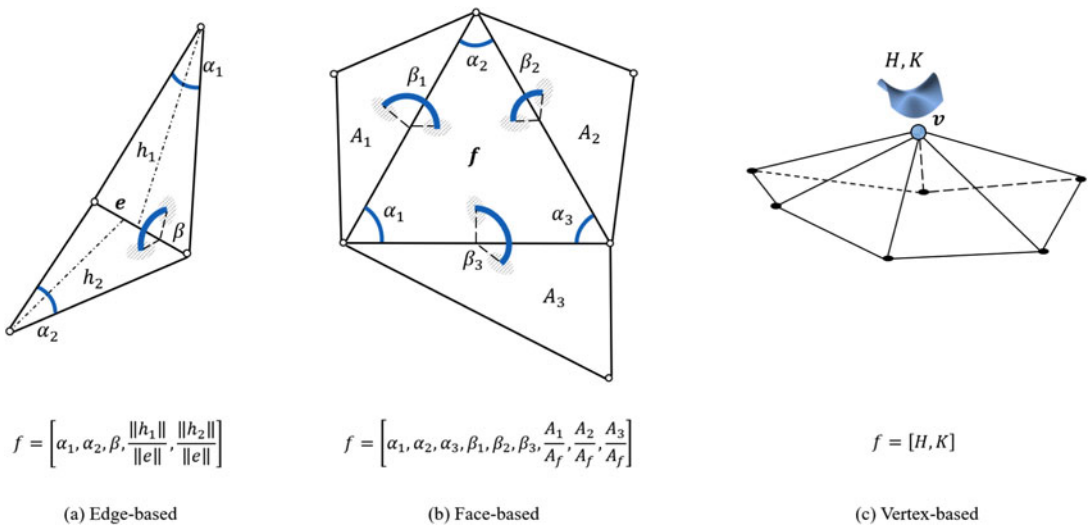

Fig. 3. Feature operators for the original implementation of MeshCNN (a) and our proposed face-based (b) and vertex-based (c) architectures.

extract features for classification and segmentation. RigNet [22] computes edge features from a mesh that are subsequently fed to a MLP model for character rigging. In [23], the authors propose an attention-based model that takes features from both edges and faces of geometric meshes for classification and segmentation. Other related work in which deep learning models are fed with geometric meshes for various applications include extracting rotation-equivariant features that are fed to a harmonic network [24], feeding the models with the curvature of the meshes [25], [26], or simulating diffusion and computing spatial gradients [27].

2.2 MeshCNN

The architecture of a MeshCNN network [1] for classification is shown in Fig. 2. First, a set of invariant features is extracted from the triangular mesh. This representation is fed to a combination of mesh convolutional and pooling layers, which is then followed by a max pooling operation to obtain a global feature representation invariant to the order of the primitives. Finally, the global feature is fed to an MLP model for shape classification. For the task of segmentation, the authors propose a U-Net style network where the MLP model is substituted by a reversed version of the convolutional and pooling layers. MeshCNN has three unique operations that make it different from a conventional CNN: feature extraction, neighbor selection in the convolutional layer, and mesh pooling. In this paper, we propose two novel implementations in which each of the operations is modified so that they can be fed with faces and vertices of a mesh. This subsection provides a summary of each of the operations of the original network.

MeshCNN is fed with features extracted from the edges of meshes. To extract these features, the authors take advantage of the constant neighborhood of the edges in a watertight manifold triangular mesh, where each edge is always adjacent to two faces, and thus four neighbor edges. This is illustrated in Fig. 3a. Once this neighborhood is established, for each edge, MeshCNN extracts the inner angles in the adjacent faces (α_1, α_2), the dihedral angle between the faces (β), and two edge ratios between the length of the edge and the length of the perpendicular vector in each face. These features are invariant to similarity transformations, i.e., if a mesh is translated, rotated, or scaled, the values of these features will remain the same.

Then, the authors combine the extracted features of each edge with the features of its neighbors. This is similar to how CNNs operate in images, in which the convolutional kernels are applied to a pixel and its neighbors. Specifically, the neighbors of each edge are defined as the remaining edges of the edge's adjacent faces. To further ensure invariance in the convolutions, the authors apply a set of symmetric operations to the edge's neighbors as shown in Fig. 4a, where e_0 is the selected edge and e_1, e_2, e_3, and e_4 are the selected neighbors.

Finally, the authors propose a pooling operation following the principles of edge collapse [28], in which edges with lower activations from the previous convolutional layer are pooled first. Therefore, the network is able to effectively learn simplified representations of the raw data. Following this method, an edge is collapsed into a vertex, and the four neighbors of the edge are merged into two resulting edges (one per adjacent face) as shown in Fig. 5a.

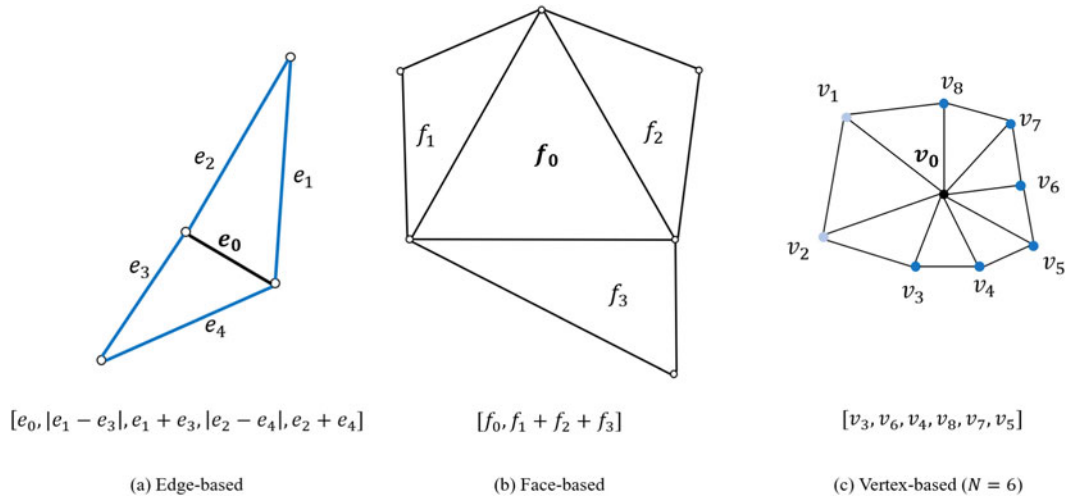

Fig. 4. Neighborhood operators for the original implementation of MeshCNN (a) and our proposed face-based (b) and vertex-based (c) architectures.

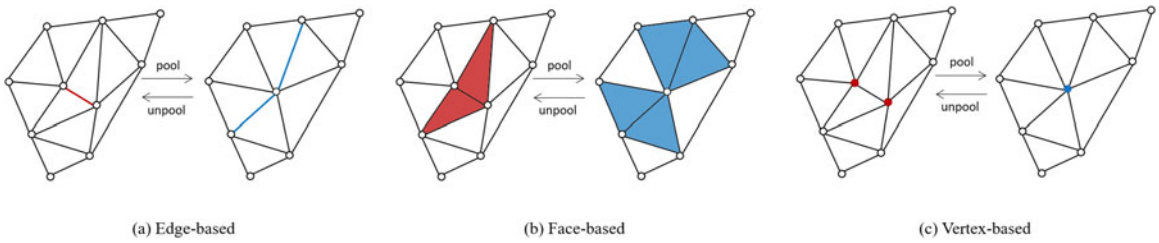

Fig. 5. Pooling operators for the original implementation of MeshCNN (a) and our proposed face-based (b) and vertex-based (c) architectures.

3 Methodology

Our proposed face-based and vertex-based networks follow the same aggregation and convolutional principles of MeshCNN. Specifically, the input of the network is combined in a feature tensor with dimensions $n_f \times n_p \times (1 + n_n)$, where n_f is the number of features, n_p is the number of primitives (face or vertex) and n_n is the number of neighbors per primitive. This input tensor can then be passed through the CNN using general matrix multiplication operations. While this part of our networks does not change with respect to MeshCNN, we completely redesign each of the operations to aggregate and process the data so that the network can be applied to faces and vertices of triangular meshes. The following subsections discuss each implementation proposed in this study.

3.1 Face-Based Architecture

3.1.1 Features

Fig. 3b shows the feature operators for our proposed face-based network. For each face, we need to select a set of features that are invariant to similarity transformations. Inspired by the original edge-based implementation of MeshCNN [1], we take advantage of the constant neighborhood of the face in a watertight manifold triangular mesh, where a face is adjacent to three other faces. With this in mind, we define a set of invariant features that are relative to the three neighbors of each face. Fig. 3b shows the extracted features. Specifically, for each face, we extract the three angles of the face $(\alpha_1, \alpha_2, \alpha_3)$, the three dihedral angles between the face and its neighbors $(\beta_1, \beta_2, \beta_3)$, and three ratios between the area of each neighbor and the area of the face. These features can be considered analogous to the edge-based features of the original MeshCNN implementation (Fig. 3a), but adapted to faces.

3.1.2 Neighborhood

The neighborhood selection of our proposed face-based implementation is depicted in Fig. 4b. For each face, we select the three faces that share an edge with it as the primitive's neighborhood. Then, we combine the features of the selected neighbors with a summation operation so that the selection order of the neighbors is invariant within the network. The symmetric operation (summation) is selected empirically as demonstrated in Section 4.1.

3.1.3 Pooling and Unpooling

Different from [1], the activations of the convolutional layer in our face-based network do not relate directly to the edges of the mesh. Instead, the activations correspond to the faces of the mesh. Incremental decimation of faces from a geometric mesh is particularly complicated and it easily leads to bad connectivity in the decimated mesh. Generally, it is preferred to keep the decimation simple by removing smaller parts of the mesh such as edges or vertices [29]. To overcome this challenge, the edges to be pooled are selected by looking at the activations of their two adjacent faces. Specifically, the network selects the edges in which the combination of the activations of its adjacent faces contributes the least to the network's objective. As in [1], the activations of the primitives are determined by computing ℓ_2-norm of their corresponding feature vector. Fig. 5b shows the pooling operation in our proposed face-based implementation.

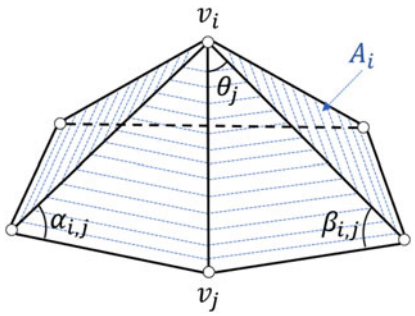

Fig. 6. Quantities used in the discretization of the mean and Gaussian curvatures.

The red faces in the diagram represent the faces with the lowest activations in the previous convolutional layer. The edge collapse operation is performed so that both faces are removed, and the features of the resulting neighbors (in blue) are computed as the average between their feature values and the values of the collapsed faces.

Unpooling is required in certain networks, such as the U-Net style networks used for segmentation. The unpooling operation is the inverse of pooling, and it is used to upsample and recover the previously pooled data. We implement unpooling layers by recovering the previously pooled faces in a way that is similar to [1]. Specifically, we keep track of the primitives removed in each pooling layer and match each unpooling layer with a pooling layer. Then, the unpooling layer recovers the faces that had been previously pooled. The features of the new unpooled faces (red faces in Fig. 5b) are computed as the average of its neighbor faces (blue faces in Fig. 5b).

3.2 Vertex-Based Architecture

3.2.1 Features

The vertices of a mesh contain the geometric information of the shape, as well as other optional information such as texture coordinates, normals, or colors. For the purposes of this study, we assume that every vertex only contains 3D coordinates (x, y, z) of the shape. Two vertices of a mesh are considered neighbors (within a 1-ring neighborhood) if they share an edge. Unlike faces and edges, a vertex can have any number of neighbors, which makes defining a constant neighborhood such as the one in [1] (Fig. 3a) computationally impossible. To overcome this, we use the curvature of each vertex as its invariant input features. Specifically, we extract the mean curvature and Gaussian curvature of each vertex.

Mean curvature is the average of the two principle curvatures (maximal curvature and minimal curvature) of a surface at a certain point [29]. The absolute mean curvature H at a vertex v_i can be calculated as shown in Equation (1), where $\Delta f(v_i)$ is the Laplace-Beltrami operator of vertex v_i.

$$H(v_i) = \frac{1}{2}||\Delta f(v_i)|| \tag{1}$$

The Laplace-Beltrami operator is a generalization of the Laplace operator to functions on surfaces [30]. The Laplace-Beltrami operator at v_i can be computed as shown in Equation (2), where A_i is the sum of the area of the faces adjacent to v_i, $\mathcal{N}_1(v_i)$ is the 1-ring neighborhood of v_i, and $\alpha_{i,j}$ and $\beta_{i,j}$ are the opposite angles of the adjacent triangles to edge (i, j), as shown in Fig. 6. This equation, which has been utilized in different applications [31], is considered to be the most popular discretization of the Laplace-Beltrami of a geometric mesh, and it provides a discrete approximation of the mean curvature of vertex v_i [29].

$$\Delta f(v_i) = \frac{1}{2A_i} \sum_{v_j \in \mathcal{N}_1(v_i)} (\cot \alpha_{i,j} + \cot \beta_{i,j})(f_j - f_i) \tag{2}$$

The Gaussian curvature of a surface at a point is the product of the principal curvatures (maximal curvature and minimal curvature) at the given point [29]. The Gaussian curvature at a vertex v_i can be discretized using Equation (3), where θ_j corresponds to the angles of the incident triangles of vertex v_i, as shown in Fig. 6. This discretization of the Gaussian curvature was proposed in [32].

$$K(v_i) = \frac{1}{A_i} \left(2\pi - \sum_{v_j \in \mathcal{N}_1(v_i)} \theta_j \right) \tag{3}$$

The discrete operators H and K define the curvature of a vertex, and they are completely invariant to similarity transformations, which makes them good candidates for features of our vertex-based network.

3.2.2 Neighborhood

Different from edges and faces, defining a constant neighborhood for a vertex is non-trivial. In this case, we cannot know beforehand the exact number of neighbors per vertex. If the features of all neighbors were to be included in each vertex, vertices with a greater number of neighbors would have more data than those with a smaller number of neighbors. This can cause the neural network to give more weight to some vertices than others, which can negatively affect the accuracy of the neural network. To avoid this, we select the N closest neighbors of a vertex as its neighborhood and include their features in the input representation sorted from closest to farthest. If a vertex has less than N neighbors, we set the features of the remaining neighbors as zero. In other words, $N_i = \min(N, |\mathcal{N}_1(v_i)|)$ for a vertex v_i. Fig. 4c depicts the neighborhood selection method for our vertex-based approach when the number of neighbors is set to $N = 6$.

3.2.3 Pooling and Unpooling

Fig. 5c shows the pooling and unpooling operations in our vertex-based network. Similar to the face-based approach, we combine the (vertex-based) activations (i.e., ℓ_2 norm of the feature vector) of the previous convolutional layer per edge and pool the edges that have the smallest combined activations. In this case, two vertices (red in Fig. 5c) are pooled into one vertex (blue in Fig. 5c), and the features of the pooled vertex are computed as the average between the features of the unpooled vertices. If the network has unpooling layers, each of them is connected to a pooling layer so that the same collapsed vertices are recovered.

TABLE 1
Testing Accuracies of the Face-Based Network on the SHREC6 Dataset With Different Concatenations of Neighborhood Operators

$f_1+f_2+f_3$	$\lvert f_1-f_2\rvert+\lvert f_1-f_3\rvert+\lvert f_2-f_3\rvert$	$f_1\times f_2+f_1\times f_3+f_2\times f_3$	$f_1\times f_2\times f_3$	$f_1^2+f_2^2+f_3^2$	$f_1^3+f_2^3+f_3^3$	Test Accuracy
X						95.00%
X	X					92.38%
X		X				80.24%
	X					80.24%
X	X	X				77.86%
		...				
X		X		X	X	9.52%
X					X	9.29%
X	X	X		X	X	8.81%
	X	X	X	X	X	8.33%
	X	X			X	8.10%

The order of the experiments is presented from highest to lowest testing accuracy. Only the 5 best and worst experiments are reported.

4 Results

In this section, we present the results obtained in different tasks and compare them with the original implementation of MeshCNN [1]. First, we present hyperparameter optimization results for the proposed networks. Then, we compare our proposed models with the original implementation of MeshCNN for the tasks of classification and segmentation of different datasets.

4.1 Hyperparameter Optimization

4.1.1 Face-Based Architecture

We carry out two experiments to determine the best configuration for selecting the neighbor information in the kernels of our proposed networks. First, we perform an experiment to determine the symmetric functions that are applied to the primitive's neighborhood in the face-based network. To do this, we utilized six different symmetric operations to combine the features of the 3 neighbors of a face (f_1, f_2, f_3). The operations are listed in the first row of Table 1. We train a face-based network for classification using all the possible concatenations of operations. Table 1 reports the testing accuracies for the five best and worst experiments. For more details, we include the results of all our experiments (64 different combinations) in Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TVCG.2021.3129156. Generally, we observe that using non-linear symmetric operations negatively affects the performance of the network.

Looking at Table 1, it can be seen that the option that produces the best results is including the features of the face neighbors in a single summation operation. Based on these results, we establish that the best neighborhood configuration for the face-based network is to only include the summation operation. The remaining results reported with the face-based architecture follow this design.

4.1.2 Vertex-Based Architecture

We carry out an experiment to determine the optimum number of neighbors for the vertex-based network. As explained in Section 3.2, the number of neighbors for a vertex is not constant within the mesh, and to tackle this issue, we include the N closest neighbors of the vertex in the input of the network. To guarantee invariance in the network, we include the features sorted from closest to farthest within the neighborhood, which serves as the symmetric operation of this implementation. Fig. 7 reports the classification accuracies using different values of N. It can be seen that while the difference is not very large by different values of N, the best results are obtained when $N = 6$, which is also the average number of vertex neighbors in a manifold mesh [29]. We set $N = 6$ in our experiments.

4.2 Classification Results

We design and train edge-based, face-based and vertex-based networks for classifying the two datasets included in the original paper of MeshCNN [1]: SHREC and Engraved Cubes. In every case, the edge-based network is designed using the same parameters as in the original implementation. The face-based and vertex-based networks are designed using the same parameters with some minor modifications. Specifically, the convolutional filters are slightly changed so that the number of trainable parameters is roughly the same in all networks, and the pooling resolution is changed in each implementation so that, in each pooling layer, the number of resulting primitives is roughly the same. Additionally, we apply data augmentation to the training set following the same configuration as that in the edge-based implementation [1]. Specifically, we augment the training data with 5% edge flips and 20% slide vertices.

4.2.1 SHREC

We evaluate the design of our face-based and vertex-based networks on the task of classifying different splits of the

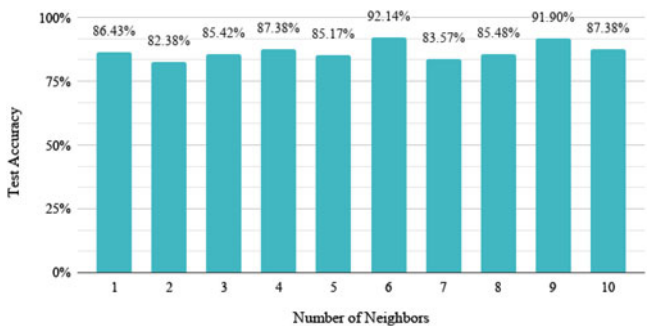

Fig. 7. Mean testing accuracies in the SHREC6 dataset using different number of vertex neighbors (N).

TABLE 2
Testing Accuracies on Classification Datasets With Different Training and Testing Splits

Dataset	Split	Edge-based [1]	Face-based (ours)	Vertex-based (ours)	Prima-Dual [23]	MeshWalker [21]
SHREC	1	0.34 ± 0.07	$\mathbf{0.56 \pm 0.01}$*	0.35 ± 0.07	–	–
	3	0.52 ± 0.07	$\mathbf{0.84 \pm 0.05}$*	0.66 ± 0.03*	–	–
	6	0.85 ± 0.03	$\mathbf{0.93 \pm 0.03}$*	0.88 ± 0.03	–	–
	10	0.94 ± 0.01	$\mathbf{0.97 \pm 0.01}$*	$\mathbf{0.97 \pm 0.01}$*	0.99	0.97
	16	0.98 ± 0.01	$\mathbf{0.99 \pm 0.01}$	$\mathbf{0.99 \pm 0.01}$	0.99	0.99
Engraved Cubes	1	0.22 ± 0.04	0.22 ± 0.06	$\mathbf{0.23 \pm 0.04}$	–	–
	10	0.79 ± 0.02	$\mathbf{0.80 \pm 0.02}$	0.77 ± 0.02	–	–
	50	$\mathbf{0.98 \pm 0.004}$	$\mathbf{0.98 \pm 0.01}$	$\mathbf{0.98 \pm 0.003}$	–	–
	100	$\mathbf{0.99 \pm 0.002}$	$\mathbf{0.99 \pm 0.003}$	$\mathbf{0.99 \pm 0.01}$	–	–
	170	$\mathbf{0.99 \pm 0.001}$	$\mathbf{0.99 \pm 0.002}$	$\mathbf{0.99 \pm 0.02}$	0.94	0.99

A '' sign means that the difference with respect to the results using the edge-based implementation is statistically significant (by t-test). Best results are highlighted in bold.*

SHREC dataset [33]. This dataset contains closed two-manifold meshes divided into 30 different classes. Each class in the dataset contains 20 meshes with a similar number of primitives but different topologies. A split in this dataset refers to the number of training samples per class. We compute 5 different splits with 1, 3, 6, 10, and 16 training samples per class. For each split, we perform 5 different experiments and show our results in Table 2 in the form of meanstd. Our experimental results show that our two networks perform significantly better than the original approach proposed by [1], which constituted the previous state of the art in this dataset. Generally, the face-based network is the one that produces the best results, especially for the splits with less training data.

4.2.2 Engraved Cubes

The authors of the original edge-based implementation of MeshCNN [1] produced their own dataset to further evaluate their model in a classification task. Their dataset, referred to in this study as "Engraved Cubes", consists of a set of cubes with shallow icon engravings divided into 23 classes. We train and test the original implementation and our proposed face-based and vertex-based networks in 5 different splits of this dataset. For each split, we perform 5 different experiments and report our results in the form of mean$\pm$std in Table 2. It can be seen that the results on this dataset are very similar across all the implementations of MeshCNN. We believe that this happens due to the simplicity of this dataset.

The last two columns of Table 2 list the classification accuracies reported in recent work of [23] and [21]. It can be seen that all performances are comparable except the one on Engraved Cubes by the model in [23], which is 5% lower. It is worth noting that these classification accuracies are directly from the original papers and we did not repeat their experiments with our data splits. Additionally, these methods did not report the standard deviation of their experiments and it is difficult to have a statistical comparison. In the future, we plan to conduct a comprehensive comparison study of our models with other related work.

4.3 Segmentation Results

We evaluate our proposed implementation on a segmentation task using the same datasets evaluated in [1]. These datasets include segmentation labels per edge, but the output of our approaches is face-based and vertex-based. To overcome this, we modify the segmentation labels for each dataset and implementation accordingly. As in Section 4.2 we design our networks so that the number of trainable parameters is roughly the same and so that the pooled primitives are proportional to the edge-based implementation.

4.3.1 COSEG

The COSEG dataset contains three different subsets of meshes with models of aliens, vases, and chairs [34]. The original versions of these datasets contain 170, 250, and 330 samples for training, respectively. We generate our own splits of the datasets using the original number of training samples and then decreasing it to 100, 50, 10, and 1. For each number of samples, we compute 5 different splits for a total of 25 experiments per dataset. The mean test accuracies for all the splits are reported in Table 3. It can be seen that the face-based architecture performs significantly better in all the splits, while the performance of the vertex-based architecture is generally not statistically different from the original edge-based architecture.

4.3.2 Human Body Segmentation

The authors of the original edge-based implementation of MeshCNN [1] report state-of-the-art results on the human body segmentation dataset proposed by [35], which contains meshes of human bodies segmented in 8 different classes. Similar to our previous experiments, we customize the original datasets for the evaluation of our models. For this case, we compute splits with 380, 100, 50, 10, and 1 training samples and report our results in Table 3. It can be seen that the face-based implementation produces significantly better results than the edge-based implementation in all the splits of the dataset. As in the case of the COSEG experiments, the performance of the vertex-based network is similar to the edge-based case.

We also include in the last two columns of Table 3 the segmentation accuracies reported by [23] and [21]. It can be seen that our face-based method obtains the best results when compared against these works on the Human Bodies dataset, while the performance in the COSEG dataset varies. In this case, we report slightly lower accuracies than those

TABLE 3
Testing Accuracies on Segmentation Datasets With Different Training and Testing Splits

Dataset	Split	Edge-based [1]	Face-based (ours)	Vertex-based (ours)	Prima-Dual [23]	MeshWalker [21]
COSEG (Aliens)	1	0.42 ± 0.05	**0.48 ± 0.06**	0.38 ± 0.03	–	–
	10	0.66 ± 0.03	**0.71 ± 0.01***	0.53 ± 0.04*	–	–
	50	0.86 ± 0.02	**0.87 ± 0.01**	0.82 ± 0.04	–	–
	100	0.93 ± 0.02	**0.94 ± 0.01**	0.93 ± 0.01	–	–
	170	0.95 ± 0.01	**0.96 ± 0.01**	0.95 ± 0.03	0.98	0.99
COSEG (Vases)	1	0.45 ± 0.08	**0.58 ± 0.09***	0.51 ± 0.11	–	–
	10	0.75 ± 0.03	**0.78 ± 0.04**	0.63 ± 0.02*	–	–
	50	0.85 ± 0.02	**0.89 ± 0.01***	0.75 ± 0.02*	–	–
	100	0.91 ± 0.01	**0.92 ± 0.01**	0.86 ± 0.01*	–	–
	250	0.93 ± 0.01	**0.95 ± 0.01***	0.93 ± 0.01	0.97	0.99
COSEG (Chairs)	1	0.37 ± 0.02	**0.49 ± 0.06***	0.41 ± 0.02*	–	–
	10	0.54 ± 0.04	**0.63 ± 0.03***	0.50 ± 0.02	–	–
	50	0.76 ± 0.01	**0.80 ± 0.01***	0.76 ± 0.02	–	–
	100	0.84 ± 0.02	**0.85 ± 0.02**	0.82 ± 0.02	–	–
	330	0.94 ± 0.01	**0.96 ± 0.01***	0.93 ± 0.02	0.95	0.99
Human Bodies	1	0.26 ± 0.04	**0.43 ± 0.06***	0.41 ± 0.09*	–	–
	10	0.58 ± 0.02	**0.79 ± 0.02***	0.60 ± 0.02	–	–
	50	0.84 ± 0.04	**0.92 ± 0.03***	0.82 ± 0.01	–	–
	100	0.88 ± 0.03	**0.96 ± 0.002***	0.87 ± 0.01	–	–
	380	0.96 ± 0.01	**0.98 ± 0.002***	0.94 ± 0.01*	0.85	0.95

reported by [23] and [21], with the exception of the Chairs subset, in which our face-based approach outperforms [23]. More thorough studies of the performance of these algorithms on extensive datasets are needed.

4.4 Summary of the Results

We include Table 4 to summarize the results reported in the classification and segmentation datasets. The table reports the percentage of experiments in which our proposed networks produced better, worse, or similar results with respect to the original edge-based MeshCNN. We consider a split to be better or worse if the difference between the accuracies is statistically significant. Otherwise, we label it as similar. It can be seen that, generally, the face-based implementation reports significantly better results than the edge-based network, while the vertex-based implementation generally produces similar results.

4.5 Computational Complexity

Our proposed architectures are slower than the original edge-based MeshCNN. This happens due to the pooling layers in the network. The pooling layers of our networks combine the activations of the faces and vertices per edge to determine where to apply edge collapse. This operation is not needed in the original edge-based implementation of MeshCNN, and since it has to be computed in each pooling layer, it significantly affects the computational time of our networks. We include in Fig. 8 the time (in minutes) that it takes to train each network with and without the pooling layers. It can be seen that when the pooling layers are included in the network, our face-based and vertex-based networks are slower than the original edge-based network. However, when no pooling layers are present, the three networks perform similarly in terms of computational time. All the experiments were carried out using a single NVIDIA V100 GPU.

TABLE 4
Overall Comparison of Classification and Segmentation Results by Our Face-Base and Vertex-Based Methods to the Original Edge-Based Implementation of MeshCNN [1]

Task	Dataset	Face-based versus Original			Vertex-based versus Original		
		Better	Worse	Similar	Better	Worse	Similar
Classification	SHREC	**80%**	0%	20%	40%	0	**60%**
	Cubes	0%	0%	**100%**	0	0	**100%**
Segmentation	Aliens	20%	0%	**80%**	0	20%	**80%**
	Vases	**60%**	0%	40%	20%	20%	**60%**
	Chairs	**80%**	0%	20%	20%	0	**80%**
	Humans	**100%**	0%	0%	20%	20%	**60%**

5 Case Study: Vertex-Based Implementation of Point2Mesh

Point2Mesh is a recently proposed method for automatic mesh reconstruction from a point cloud [20]. A diagram of the model is shown in Fig. 9. This method first computes an

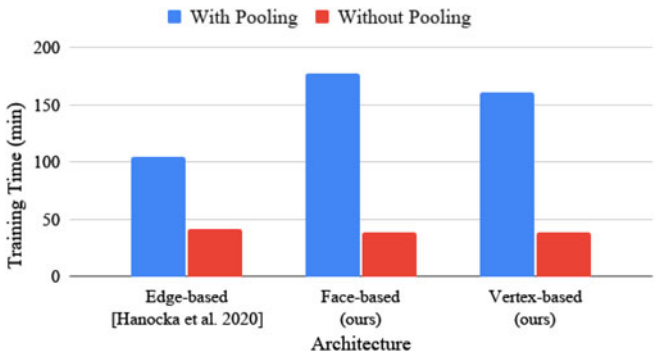

Fig. 8. Computational time of training each type of network for classification of the SHREC10 dataset.

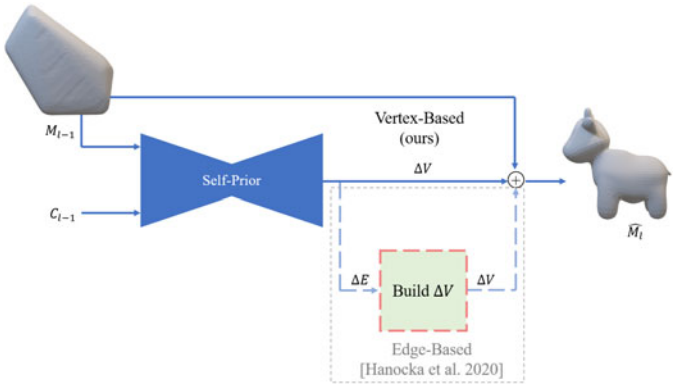

Fig. 9. Original version of Point2Mesh model and our vertex-based implementation. The original version uses the "Build ΔV" module to compute vertex displacements (ΔV), we eliminate the module and use the "Self-Prior" module to directly generate ΔV.

initial coarse approximation of the reconstructed mesh, and then a self-prior iteratively learns to displace the vertices of the mesh. The self-prior is defined as an autoencoder that follows the principles of MeshCNN. The goal of the self-prior is to learn displacements for the vertices of the mesh. However, since the network is designed using the original edge-based implementation of MeshCNN, the displacements of the network are produced by edge. Specifically, the network produces a set of edge displacements ΔE that consists of pairs of vertex displacements. Because one vertex is adjacent to several edges, the produced vertex displacements must be combined so that the deformation is consistent. To tackle this issue, the authors propose an extra module referred to as "Build ΔV", which averages the vertex displacements in ΔE following Equation (4), where v_i is computed as the average of all the vertex positions in its adjacent edges, and n is the valence of v_i.

$$v_i = \frac{1}{n}\sum_{j \in n} e_j(v_i) \tag{4}$$

While aggregating the edge-based output produces very good results, a vertex-based implementation of MeshCNN could be used for the self-prior so that the "Build ΔV" module is not needed. We design a vertex-based version of Point2Mesh where the "Build ΔV module is excluded, as shown in Fig. 9, and compare it against the original implementation in different qualitative and quantitative experiments.

5.1 Visual Results

We train edge-based and vertex-based networks for mesh reconstruction of five different point-clouds provided in the original source code of Point2Mesh [20]. In every case, we design the networks following the same configuration as in the original version of the network. Fig. 10 shows the reconstructed meshes from clean and complete input point clouds by the original edge-based network and our proposed vertex-based implementation. It can be seen that our approach produces reconstructions that are similar to the original edge-based approach. In the remainder of this Section, we will further explore the capability of our proposed methods for mesh reconstruction from noisy and incomplete point clouds.

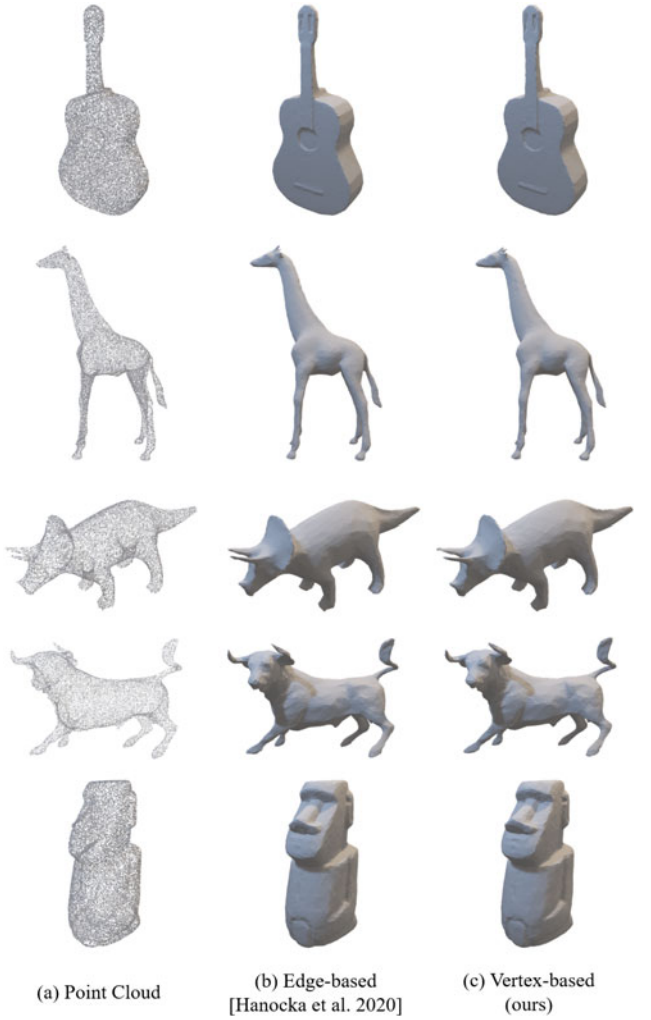

Fig. 10. Mesh reconstructions from clean and complete point clouds obtained by the edge-based approach (b) and by our proposed vertex-based approach (c).

5.2 Evaluation Metrics

Similarly to [20], we quantitatively evaluate our method by sampling points on the reconstructed and ground truth meshes and compute the Hausdorff distance and F-score as the comparison metrics.

The Hausdorff distance is considered to be the sharpest distance error estimate between two meshes [29], and it is the maximum of the minimum distance between two sets of points, as shown in Equation (5). Because this distance metric is not symmetric (i.e., $d_H(A,B) \neq d_H(B,A)$), it is normally preferred to use the symmetric Hausdorff distance, defined as the maximum of both distances (Equation (6)).

$$d_H(A,B) = \max_{a \in A} \min_{b \in B} ||a - b|| \tag{5}$$

$$d_{SH}(A,B) = \max\{d_H(A,B), d_H(B,A)\} \tag{6}$$

The F-score between two geometric meshes is a metric first proposed by [36] and it can be defined as the harmonic mean between the precision $P(\tau)$ and recall $R(\tau)$ at a user-specified distance threshold τ, as shown in Equation (7). With this metric, a given reconstruction would have an F-score of 100 in the best-case scenario and of 0 in the worst case.

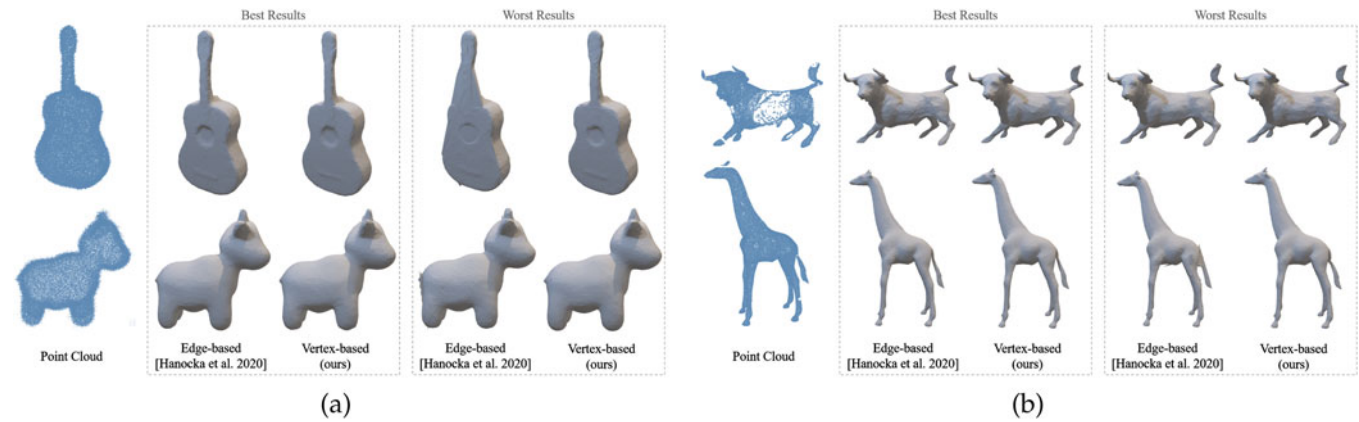

Fig. 11. Mesh reconstruction results from noisy (a) and incomplete (b) point clouds.

TABLE 5
Quantitative Performance Metrics for Mesh Reconstruction From Noisy and Incomplete Point Clouds

Experiment	Mesh	Hausdorff Distance		F-score	
		Edge-based [20]	Vertex-based (ours)	Edge-based [20]	Vertex-based (ours)
Noisy Point Clouds	Guitar	0.0059 ± 0.0094	**0.0020± 0.0008**	**99.19± 0.78**	94.81± 6.17
	Cow	0.0086± 0.0006	**0.0084± 0.0022**	**76.52± 9.46**	66.00± 6.90
Incomplete Point Clouds	Bull	0.0076± 0.0063	**0.0043± 0.0018**	77.34± 21.74	**87.66± 8.10**
	Giraffe	0.0027± 0.0009	**0.0018± 0.0016**	94.13± 7.66	**96.79± 4.45**

Hausdorff distance (lower is better) and F-score (higher is better) for five different experiments in the form of mean±std.

$$F(\tau) = \frac{2P(\tau)R(\tau)}{P(\tau) + R(\tau)} \tag{7}$$

It is important to note that, while these distortion-based metrics can help us estimate the quality of our approach, they are not necessarily linked to the visual quality of the results [37].

5.3 Mesh Reconstruction From Noisy Point Clouds

We evaluate our approach on a denoising task following an experimental setup similar to that used in [20]. Specifically, we sample 75,000 points from a ground truth mesh and then we add a small amount of Gaussian noise to each coordinate (x, y, z). We perform each experiment five times and show in Fig. 11a the best and worst visual results for each network and mesh. It can be seen that, while there is not a perceptual difference between the edge-based and vertex-based network in the best case, the worst results produced by the original edge-based network tend to be worse than those produced by our vertex-based approach. For more details, we include the reconstructions of all the denoising experiments in Appendix B, available in the online supplemental material.

Additionally, Table 5 reports the quantitative results for shape denoising in terms of Hausdorff distance and F-score. For each experiment, we sample 25,000 points from the reconstruction and the ground truth and set a distance threshold of $\tau = 0.002$. It can be seen that our vertex-based approach reports slightly better results in terms of Hausdorff distance, but slightly worse results in terms of F-score. However, the difference between both approaches is not statistically significant in either case.

5.4 Mesh Reconstruction From Incomplete Point Clouds

We test our approach and the original edge-based network in the task of mesh reconstruction from incomplete point clouds. We sample 75,000 points from two ground truth meshes and manually remove points from certain areas of the meshes to come up with incomplete point clouds similar to the ones used in [20]. We perform 5 different experiments for each point cloud and network and show qualitative and quantitative results in Fig. 11b and Table 5, respectively. The visual results in Fig. 11b show that there are not perceptual differences between the best reconstructions of our approach and the original edge-based method. Similar to the denoising task, we observe that the worse results of the edge-based network contain more noise than the results obtained with our method. The reconstructed meshes from every experiment are included in Appendix B, available in the online supplemental material. Quantitatively, Table 5 shows that our approach performs better both in terms of Hausdorff distance and F-score. However, the difference between our approach and the original method is not statistically significant.

5.5 Computational Time

The results produced by the edge-based implementation were already superb, and beating these results is very challenging. However, our network is significantly faster than

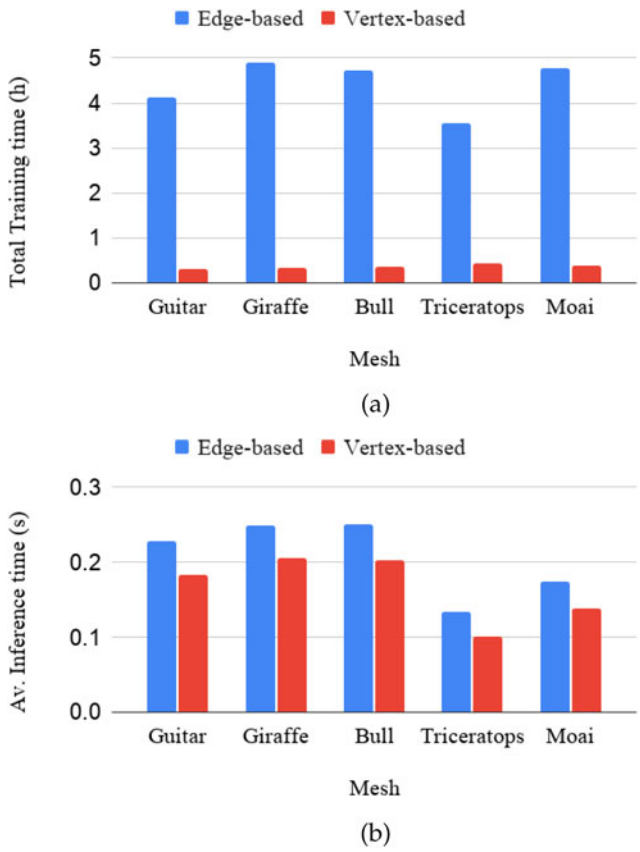

Fig. 12. Total training time (a) and average inference time (b) for mesh reconstruction using the edge-based approach and our proposed vertex-based approach.

the original one, as shown in Fig. 12. This happens because our vertex-based network directly produces displacements for the mesh vertices, and thus, it does not need to aggregate the output using the "Build ΔV" module. Specifically, when the two networks are trained for 6,000 iterations, the edge-based approach takes an average of 4.41 hours to finalize training, while our vertex-based approach only takes 36.71 minutes. This corresponds to a percentage decrease of about 91%. The difference is lower in terms of inference time. In this case, the edge-based approach takes about 0.21 seconds to compute a reconstruction, while our vertex-based approach takes an average of 0.17 seconds, for an average percentage decrease of about 20%. All the experiments were computed using a single NVIDIA V100 GPU.

6 Discussion

We have proposed two novel implementations of mesh convolutional neural networks that can directly handle face-based or vertex-based inputs. Our proposed networks do not only outperform the original MeshCNN in mesh classification and segmentation [1], but also significantly reduce model training and inference time for the Point2Mesh model in mesh reconstruction [20], demonstrating that the face-based and vertex-based operators are highly effective in mesh convolutional networks.

In the mesh classification task, we test our networks on the SHREC dataset [33] and the Engraved Cubes dataset [1] with different training and testing data splits. Our experiments

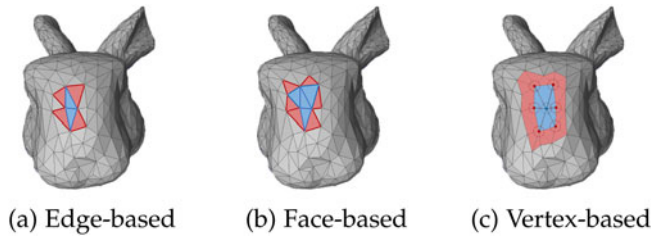

Fig. 13. Comparison of the receptive field used as an input in the different implementations of MeshCNN.

show that both of our face-based and vertex-based networks outperform the original MeshCNN for classifying the SHREC dataset, setting a new state of the art. On the Engraved Cubes dataset, our networks perform similarly as compared to the original MeshCNN. This dataset has a lower complexity than that of SHREC. First, it contains fewer classes than that of SHREC (23 classes versus 30 classes). Second, the meshes in the Engraved Cubes dataset are simple. Specifically, they are formed by cubes with shallow engravings of different shapes, making actual representations of the meshes equivalent to 2D silhouettes. We believe that the low complexity of this dataset makes the differences among the results produced by different networks negligible.

In the segmentation task, while our face-based approach outperforms the original MeshCNN, our vertex-based approach performs similarly. As in the mesh classification task, we observe that the higher the complexity of a dataset, the larger the difference among performances produced by the different approaches. For mesh segmentation, the Human Body dataset is more complex than COSEG as it contains more classes than COSEG (8 versus 3-4), and each mesh in the dataset has a higher resolution. The high complexity makes the Human Body dataset more challenging to be segmented than the COSEG dataset, which we believe leads to the larger margins in performance differences of our face-based architecture over the original MeshCNN.

Additionally, we conduct a case study in which we adapt Point2Mesh [20] to follow our vertex-based implementation for mesh reconstruction from clean, noisy, and incomplete point clouds. While experimental results do not show statistical differences between our approach and the original Point2Mesh, our approach achieves a 91% reduction of model training time and about 20% reduction of model inference time. This demonstrates that using a network specifically designed for a task is a better solution than post-processing results to convert from one primitive to another.

The goal of this study is to design and develop vertex-based and face-based implementations for MeshCNN. We have designed features, neighborhood operations, and pooling layers that are fundamentally modified to fit in the MeshCNN architecture so that it can take faces and vertices as inputs. While the proposed approaches produce promising results, we intend to gain theoretical insights into the differences between the face and edge features in the next step of our research. One observation is that the input receptive fields by different approaches are different as shown in Fig. 13, which could lead to the different performances. Our future work will focus on the theoretical understanding of the different operators as well as on exploring other geometry processing techniques that can potentially benefit deep

learning for 3D data in general. Other extensions of this work could be investigating implications of different methods for the pooling criteria, studying the possibility of taking non-manifold meshes directly as inputs, and offering a systematic and extensive comparison with other mesh-based architectures in the literature [18], [19], [21], [22], [23].

7 Conclusion

In this paper, we propose a face-based and a vertex-based implementation for mesh convolutional neural networks. We demonstrate the advantages of our novel implementations for mesh classification and mesh segmentation and achieve new state of the art for benchmark mesh classification datasets. In addition, we incorporate the vertex-based operator into a Point2Mesh model for mesh reconstruction from point clouds and significantly reduce model training and inference time. The source codes of our implementations will be made available to the public in the coming months.

References

[1] R. Hanocka, A. Hertz, N. Fish, R. Giryes, S. Fleishman, and D. Cohen-Or , "MeshCNN: A network with an edge," *ACM Trans. Graph.*, vol. 38, no. 4, pp. 1–12, 2019.

[2] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[3] W. Cao, Z. Yan, Z. He, and Z. He, "A comprehensive survey on geometric deep learning," *IEEE Access*, vol. 8, pp. 35 929–35 949, 2020.

[4] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: Going beyond euclidean data," *IEEE Signal Process. Mag.*, vol. 34, no. 4, pp. 18–42, Jul. 2017.

[5] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller , "Multi-view convolutional neural networks for 3D shape recognition," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2015, pp. 945–953.

[6] E. Kalogerakis, M. Averkiou, S. Maji, and S. Chaudhuri, "3D shape segmentation with projective convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 3779–3788.

[7] Z. Wu *et al.*, "3D shapenets: A deep representation for volumetric shapes," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 1912–1920.

[8] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "Generative and discriminative voxel modeling with convolutional neural networks," in *Proc. NIPS 3D Deep Learn. Workshop*, 2016.

[9] Y. Li, S. Pirk, H. Su, C. R. Qi, and L. J. Guibas, "FPNN: Field probing neural networks for 3D data," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 307–315.

[10] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep learning on point sets for 3D classification and segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 652–660.

[11] 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. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5099–5108.

[12] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph CNN for learning on point clouds," *ACM Trans. Graph.*, vol. 38, no. 5, pp. 1–12, 2019.

[13] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," in *Proc. Int. Conf. Learn. Representations*, 2014.

[14] M. Henaff, J. Bruna, and Y. LeCun, "Deep convolutional networks on graph-structured data," 2015, *arXiv:1506.05163*.

[15] S. Ebli, M. Defferrard, and G. Spreemann, "Simplicial neural networks," in *Proc. Topological Data Anal. Beyond Workshop NeurIPS*, 2020. [Online]. Available: https://arxiv.org/abs/2010.03633

[16] D. Boscaini, J. Masci, E. Rodolà, and M. Bronstein, "Learning shape correspondence with anisotropic convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 3189–3197.

[17] A. Sinha, J. Bai, and K. Ramani, "Deep learning 3D shape surfaces using geometry images," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 223–240.

[18] H.-T. D. Liu, V. G. Kim, S. Chaudhuri, N. Aigerman, and A. Jacobson, "Neural subdivision," *ACM Trans. Graph.*, vol. 39, no. 4, pp. 1–16, Jul. 2020.

[19] A. Hertz, R. Hanocka, R. Giryes, and D. Cohen-Or , "Deep geometric texture synthesis," *ACM Trans. Graph.*, vol. 39, no. 4, pp. 108:1–108:11, 2020.

[20] R. Hanocka, G. Metzer, R. Giryes, and D. Cohen-Or, "Point2Mesh: A self-prior for deformable meshes," *ACM Trans. Graph.*, vol. 39, no. 4, pp. 1–12, 2020.

[21] A. Lahav and A. Tal, "MeshWalker: Deep mesh understanding by random walks," *ACM Trans. Graph.*, vol. 39, no. 6, pp. 1–13, 2020.

[22] Z. Xu, Y. Zhou, E. Kalogerakis, C. Landreth, and K. Singh, "RigNet: Neural rigging for articulated characters," *ACM Trans. Graph.*, vol. 39, pp. 1–14, 2020.

[23] F. Milano, A. Loquercio, A. Rosinol, D. Scaramuzza, and L. Carlone, "Primal-dual mesh convolutional neural networks," in *Proc. Conf. Neural Inf. Process. Syst.*, 2020.

[24] R. Wiersma, E. Eisemann, and K. Hildebrandt, "CNNs on surfaces using rotation-equivariant features," *ACM Trans. Graph.*, vol. 39, no. 4, pp. 92–1, 2020.

[25] A. A. M. Muzahid, W. Wan, F. Sohel, L. Wu, and L. Hou, "CurveNet: Curvature-based multitask learning deep networks for 3d object recognition," *IEEE/CAA J. Automatica Sinica*, vol. 8, no. 6, pp. 1177–1187, Jun. 2021.

[26] W. He, Z. Jiang, C. Zhang, and A. M. Sainju, "CurvaNet: Geometric deep learning based on directional curvature for 3D shape analysis," in *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discov. & Data Mining*, New York, NY, USA 2020, pp. 2214–2224.

[27] N. Sharp, S. Attaiki, K. Crane, and M. Ovsjanikov, "Diffusion is all you need for learning on surfaces," 2020, *arXiv:2012.00888*.

[28] H. Hoppe, "View-dependent refinement of progressive meshes," in *Proc. 24th Annu. Conf. Comput. Graph. Interactive Techn.*, 1997, pp. 189–198.

[29] M. Botsch, L. Kobbelt, M. Pauly, P. Alliez, and B. Lévy, *Polygon Mesh Processing*. Boca Raton, F.L, USA: CRC Press, 2010.

[30] S. Rosenberg and R. Steven, *The Laplacian on a Riemannian Manifold: An Introduction to Analysis on Manifolds*. Cambridge, U.K.: Cambridge Univ. Press, 1997.

[31] Z. Afrose and Y. Shen, "Mesh color sharpening," *Adv. Eng. Softw.*, vol. 91, 2016, Art. no. 3643.

[32] M. Meyer, M. Desbrun, P. Schröder, and A. H. Barr, "Discrete differential-geometry operators for triangulated 2-manifolds," in *Proc. Vis. Math.*, 2003, pp. 35–57.

[33] Z. Lian *et al.*, "Shape retrieval on non-rigid 3D watertight meshes," in *Proc. Eurograph. Workshop 3D Object Retrieval.*, 2011.

[34] Y. Wang, S. Asafi, O. Van Kaick , H. Zhang, D. Cohen-Or, and B. Chen, "Active co-analysis of a set of shapes," *ACM Trans. Graph.*, vol. 31, no. 6, pp. 1–10, 2012.

[35] H. Maron *et al.*, "Convolutional neural networks on surfaces via seamless toric covers," *ACM Trans. Graph.*, vol. 36, no. 4, pp. 1–10, 2017.

[36] A. Knapitsch, J. Park, Q.-Y. Zhou, and V. Koltun, "Tanks and temples: Benchmarking large-scale scene reconstruction," *ACM Trans. Graph.*, vol. 36, no. 4, pp. 1–13, 2017.

[37] Y. Blau and T. Michaeli, "The perception-distortion tradeoff," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 6228–6237.

Daniel Perez received the BS degree in computer engineering from Old Dominion University, Norfolk VA, in 2016, and the ME and PhD degrees in modeling and simulation in 2019 and 2021, respectively. He was conducting research with Pixar Animation Studios and Google LLC, where he is currently a software engineer. He has authored or coauthored multiple papers in different areas and worked on several projects during his studies at Old Dominion University. His research interests include computer graphics and deep learning. He was the recipient of Gene Newman Award for excellence in modeling simulation research in 2018.

Yuzhong Shen (Senior Member, IEEE) received the BSEE degree from Fudan University, Shanghai, China, in 1990, the MSCE degree from Mississippi State University, Starkville, in 2000, and the PhD degree in electrical engineering from the University of Delaware, Newark, in 2004. From 1990 to 1998, he was an engineer and a senior engineer with HuaGuang Technologies, Weifang, China. From 1998 to 2000, he was a research assistant with the National Science Foundation Engineering Research Center for Computational Field Simulation, Mississippi State University. From 2004 to 2006, he was a senior research scientist with Virginia Modeling, Analysis, and Simulation Center, Old Dominion University, Norfolk, VA, where he is currently a professor with the Department of Computational Modeling and Simulation Engineering, with a joint appointment with the Department of Electrical and Computer Engineering. His research interests include visualization and computer graphics, virtual reality and augmented reality, modeling and simulation, and signal and image processing. He is a member of the IEEE Computer Society Technical Committee on Simulation and the IEEE Computer Society Technical Committee on Visualization and Computer Graphics. He is a member of the Society for Modeling and Simulation International and a member of American Society for Engineering Education.

Jiang Li (Senior Member, IEEE) received the BS degree in electrical engineering from Shanghai Jiaotong University, China, in 1992, the MS degree in automation from Tsinghua University, China, in 2000, and the PhD degree in electrical engineering from the University of Texas at Arlington, TX, in 2004. He is currently a professor with the Department of Electrical and Computer Engineering, Old Dominion University. He has authored or coauthored more than 100 journal and conference papers. His research interests include deep learning, computer-aided medical diagnosis systems, medical signal or image processing, and cybersecurity. From 2004 to 2006, he was a postdoctoral fellow with the Department of Radiology, National Institutes of Health. He joined ODU as an assistant professor in Spring 2007.