MeshCNN: A Network with an Edge

RANA HANOCKA, Tel Aviv University
AMIR HERTZ, Tel Aviv University
NOA FISH, Tel Aviv University
RAJA GIRYES, Tel Aviv University
SHACHAR FLEISHMAN, Amazon
DANIEL COHEN-OR, Tel Aviv University

Polygonal meshes provide an efficient representation for 3D shapes. They explicitly capture both shape surface and topology, and leverage non-uniformity to represent large flat regions as well as sharp, intricate features. This non-uniformity and irregularity, however, inhibits mesh analysis efforts using neural networks that combine convolution and pooling operations. In this paper, we utilize the unique properties of the mesh for a direct analysis of 3D shapes using *MeshCNN*, a convolutional neural network designed specifically for triangular meshes. Analogous to classic CNNs, MeshCNN combines specialized convolution and pooling layers that operate on the mesh edges, by leveraging their intrinsic geodesic connections. Convolutions are applied on edges and the four edges of their incident triangles, and pooling is applied via an edge collapse operation that retains surface topology, thereby, generating new mesh connectivity for the subsequent convolutions. MeshCNN learns which edges to collapse, thus forming a task-driven process where the network exposes and expands the important features while discarding the redundant ones. We demonstrate the effectiveness of MeshCNN on various learning tasks applied to 3D meshes.

CCS Concepts: • **Computing methodologies → Neural networks**; **Shape analysis**.

Additional Key Words and Phrases: Geometric Deep Learning, Shape Analysis, Convolutional Neural Network, Shape Segmentation

ACM Reference Format:
Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. 2019. MeshCNN: A Network with an Edge. *ACM Trans. Graph.* 38, 4, Article 90 (July 2019), 12 pages. https://doi.org/10.1145/3306346.3322959

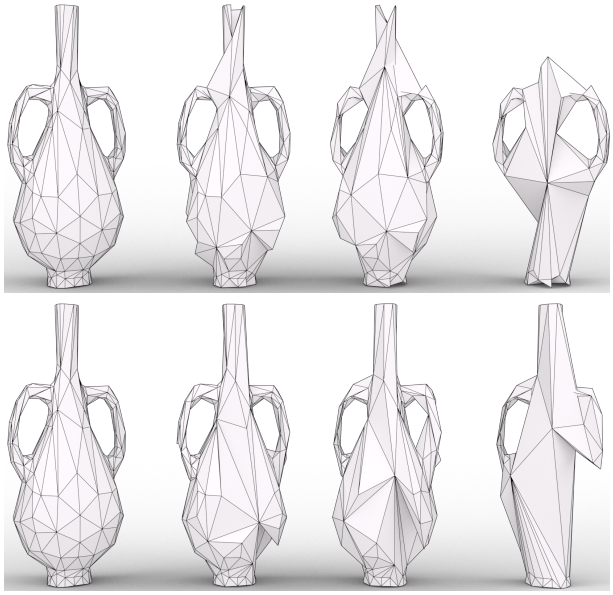


Fig. 1. Mesh pooling operates on irregular structures and adapts spatially to the task. Unlike geometric simplification (removes edges with a minimal geometric distortion), mesh pooling delegates which edges to collapse to the network. Top row: MeshCNN trained to classify whether a vase has a *handle*, bottom row: trained on whether there is a *neck* (top-piece).

1 INTRODUCTION

Three dimensional shapes are prevalent in the field of computer graphics, but also a major commodity in related fields such as computer vision and computational geometry. Shapes around us, and in particular those describing natural entities, are commonly composed of continuous surfaces. For computational reasons, and to facilitate data processing, various discrete approximations for 3D shapes have been suggested and utilized to represent shapes in an array of applications. A favorite of many, the polygonal mesh representation, or mesh, for short, approximates surfaces via a set of 2D polygons in 3D space [Botsch et al. 2010]. The mesh provides an efficient, non-uniform representation of the shape. On the one hand, only a small number of polygons are required to capture large, simple, surfaces. On the other hand, representation flexibility supports a higher resolution where needed, allowing a faithful reconstruction, or portrayal, of salient shape features that are often geometrically intricate. Another advantageous characteristic of the mesh is the inherent ability to encode connectivity information. This forms a comprehensive representation of the underlying surface.

These advantages are apparent in comparison to another popular option: the point cloud representation. Despite its simplicity and direct relation to common data acquisition techniques (scanning), the point cloud representation falls short when a higher quality and preservation of sharp shape features are required.

Authors' addresses: Rana Hanocka, Tel Aviv University; Amir Hertz, Tel Aviv University; Noa Fish, Tel Aviv University; Raja Giryes, Tel Aviv University; Shachar Fleishman, Amazon; Daniel Cohen-Or, Tel Aviv University.

0730-0301/2019/7-ART90 $15.00
https://doi.org/10.1145/3306346.3322959

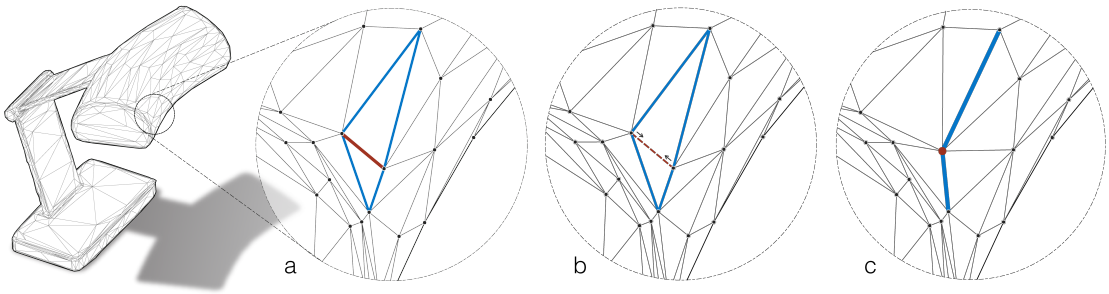


Fig. 2. (a) Features are computed on the edges by applying convolutions with neighborhoods made up of the four edges (blue) of the two incident triangles of an edge (red). The pooling step is shown in (b) and (c).

In recent years, using convolutional neural networks (CNNs) on images has demonstrated outstanding performance on a variety of tasks such as classification and semantic segmentation [2013; 2014; 2018]. The recipe for their success features a combination of convolution, non-linearity and pooling layers, resulting in a framework that is invariant (or *robust*) to irrelevant variations of the input [LeCun 2012; Krizhevsky et al. 2012]. However, since images are represented on a regular grid of discrete values, extending CNNs to work on irregular structures is nontrivial.

Initial approaches bypassed adapting CNNs to irregular data by using regular representations: mapping 3D shapes to multiple 2D projections [Su et al. 2015] or 3D voxel grids [Wu et al. 2015]. While these approaches benefit from directly using well understood image CNN operators, their indirect representation requires prohibitively large amounts of memory with wasteful or redundant CNN computations (*e.g.,* convolutions on unoccupied voxels).

More efficient approaches directly apply CNNs on the irregular and sparse point cloud representation [Qi et al. 2017a]. While these approaches benefit from a compact input representation, they are inherently oblivious to the local surface. Moreover, the notion of neighborhoods and connectivity is ill-defined, making the application of convolution and pooling operations nontrivial. This ambiguity has resulted in a wave of works geared towards overcoming this challenge [Monti et al. 2017; Wang et al. 2018a; Li et al. 2018; Yi et al. 2017].

Aiming to tap into the natural potential of the native mesh representation, we present *MeshCNN*: a neural network akin to the well-known CNN, but designed specifically for meshes. MeshCNN operates directly on irregular triangular meshes, performing convolution and pooling operations designed in harmony with the unique mesh properties. In MeshCNN, the edges of a mesh are analogous to pixels in an image, since they are the basic building blocks which all operations are applied on. We choose to work with edges since every edge is incident to exactly two faces (triangles), which defines a natural fixed-sized convolutional neighborhood of four edges (see Figure 2). We utilize the consistent face normal order to apply a symmetric convolution operation, which learns edge features that are invariant to transformations in rotation, scale and translation.

MeshCNN proposes *mesh pooling*, which operates on irregular structures and spatially adapts to the task. In CNNs, pooling downsamples the number of features in the network, thereby learning to eliminate less informative features. Since features are on the edges, an intuitive approach for down-sampling is to use the well-known mesh simplification technique *edge collapse* [Hoppe 1997]. Traditional edge collapse aims to preserve the original shape by removing edges which introduce a minimal geometric distortion. While it is possible to use traditional edge collapse for pooling, instead, we opt for a learned-edge collapse: by delegating the choice of which edges to collapse to the network. The deleted edges are the ones whose features contribute the least to the used objective (see examples in Figures 1 and 8).

To increase flexibility and support a variety of available data, each pooling layer simplifies the mesh to a predetermined constant number of edges. While our method is mathematically invariant to similarity transformations and edge ordering, it is not guaranteed to be invariant to different triangulations or vertex shifts on the surface. In order to generalize to different meshing and vertex variations, we employ several data augmentation techniques during training, for example edge flips and vertex perturbations, which leads to robustness in practice. To illustrate the aptitude of our method, we perform a variety of experiments on shape classification and segmentation tasks and demonstrate superior results to state-of-the-art approaches on common datasets and on highly non-uniform meshes.

2 RELATED WORK

Many of the operators that we present or use in our work are based on classic mesh processing techniques [Hoppe 1999; Rusinkiewicz and Levoy 2000; Botsch et al. 2010; Kalogerakis et al. 2010], or more specifically, mesh simplification techniques [Hoppe et al. 1993; Garland and Heckbert 1997; Hoppe 1997]. In particular, we use the edge-collapse technique [Hoppe 1997] for our task-driven pooling operator. While classic mesh simplification techniques aim to reduce the number of mesh elements with minimal geometric distortion [Tarini et al. 2010; Gao et al. 2017], in this work we use the mesh simplification technique to reduce the resolution of the feature maps within the context of a neural network. In the following, we revisit relevant work on 3D data analysis using neural networks, organized according to input representation type.

2.0.1 Multi-view 2D projection. Leveraging existing techniques and architectures from the 2D domain is made possible by representing 3D shapes through their 2D projections from various viewpoints.

These sets of rendered images serve as input to subsequent processing by standard CNN models. Su et al. [2015] were the first to apply a multi-view CNN for the task of shape classification, however, this approach (*as is*) cannot perform semantic segmentation. Later, [Kalogerakis et al. 2017] suggested a more comprehensive multi-view framework for shape segmentation: generating image-level segmentation maps per view and then solving for label consistency using CRF (trained end-to-end). Qi et al. [2016] explored both view-based and volumetric approaches, and observed the superiority of the first compared to the methods available at that time. More recently, several works have considered multi-layer representations per view. Sarkar et al. [2018] represent the 3D shape using multi-layered height-maps along with a novel multi-view merging technique. Zhou et al. [2018] train on motion clips to perform view extrapolation. Gomez-Donoso et al. [2017] use three slices of a 3D point cloud (one for each axis) for object classification.

2.0.2 Volumetric. Transforming a 3D shape into a binary voxel form provides a grid-based representation that is analogous to the 2D grid of an image. Operations that are applied on 2D grids can be extended to 3D grids in a straight-forward manner, allowing a natural transference of common image-based approaches to the shape domain. Wu et al. [2015] pioneered this concept, and presented a CNN that processes voxelized shapes for classification and completion. Brock et al. [2016] tackled shape reconstruction using a voxel-based variational autoencoder, and [Tchapmi et al. 2017] combined trilinear interpolation and Conditional Random Fields (CRF) with a volumetric network to promote semantic shape segmentation. Hanocka et al. [2018] used volumetric shape representations to train a network to regress grid-based warp fields for shape alignment, and applied the estimated deformation on the original mesh.

Despite their alluring simplicity, volumetric representations are computationally demanding, requiring significant memory usage. To alleviate this, several acceleration strategies have been proposed, where sparsity of shape occupancy within the volume is exploited for representation reduction [Li et al. 2016; Riegler et al. 2017; Wang et al. 2017; Graham et al. 2017].

2.0.3 Graph. A common generalization of grid-based representations that allows non-regularity, is the graph structure. To support graph-based data analysis, considerable focus has been directed toward the application of neural networks to popular tasks involving data represented in graph form, mainly, social networks, sensor networks in communication, or genetic data. One approach advocates for the processing of the Laplacian of the graph representation [Bruna et al. 2014; Henaff et al. 2015; Defferrard et al. 2016; Kostrikov et al. 2018], and thus operates in the spectral domain. Another approach opts to process the graph directly by extracting locally connected regions and transforming them into a canonical form to be processed by a neural network [Niepert et al. 2016]. Atwood et al. [2016] proposed diffusion-convolution, where diffusion is applied on each node to determine its local neighborhood. Monti et al. [2017] parameterize the surface into local patches using the graph spatial domain. Xu et al. [2017] use directional convolutions on surface patches for the task of semantic segmentation. Yi et al. [2017] use graph convolutions in the spectral domain on the task of 3D segmentation. Kostrikov et al. [2018] use a laplacian surface network for developing a generative model for 3D shapes. Such et al. [2017] introduced the concept of vertex filtering on graphs, but did not incorporate pooling operations for feature aggregation. These methods commonly operate on the *vertices* of a graph.

2.0.4 Manifold. The pioneering work of Masci et al. [2015] introduced deep learning of local features on meshes (intrinsic mesh descriptors similar to [Kokkinos et al. 2012]), which was used for correspondence and retrieval. Specifically, they demonstrate how to make the convolution operations intrinsic to the mesh.

Often, local patches on a manifold shape are approximately Euclidean. This characteristic can be exploited for manifold shape analysis using standard CNNs, by parameterizing the 3D manifold to 2D [Henaff et al. 2015; Boscaini et al. 2016; Sinha et al. 2016; Maron et al. 2017; Ezuz et al. 2017]. Boscaini et al. [2015] use vertex-frequency analysis to learn a local 3D shape descriptor. Another approach parameterizes sphere-type shapes to a planar flat-torus, where convolutions are well defined [Haim et al. 2018; Maron et al. 2017]. Recently, Jiang et al. [2019] parameterize 3D meshes to icosahedral spheres, and propose a new convolution operator which is highly efficient and effective. Poulenard et al. [2018] define a new convolutional layer that allows propagating geodesic information throughout the layers of the network.

Verma et al. [2018] proposed a graph neural network in which the neighborhood of each vertex for the convolution operation is not predefined but rather calculated dynamically based on its features. Tatarchenko et al. [2018] introduced tangent convolution, where a small neighborhood around each point is used to reconstruct the local function upon which convolution is applied. Unlike previous works, they incorporated pooling operations by subsampling on a regular 3D grid. Some generative models have been also proposed. Litany et. al. [2018] introduce an autoencoder that performs shape completion. Ranjan et. al. [2018] demonstrate how 3D faces can be generated by mesh autoencoders.

See [Bronstein et al. 2017] for a comprehensive survey on geometric deep learning. The uniqueness of our approach compared to the previous ones is that our network operations are specifically designed to adapt to the mesh structure. In particular, we learn a unique pooling operator that selects which areas to simplify, based on the target task.

To the best of our knowledge this is the first work that proposes (i) an equivariant convolution on the edges of a mesh and (ii) a learned mesh pooling operation that adapts to the task at hand. In [Ranjan et al. 2018], a fixed pooling operation has been proposed for meshes autoencoders. Learned pooling operations have been proposed in the context of graph neural networks [Ying et al. 2018; Cangea et al. 2018]. A convolution that extracts edge features has been proposed with the dual graph convolution models [Monti et al. 2018] that extend the graph attention networks [Velickovic et al. 2018]. Yet, since these approaches work on general graphs, they do not exploit the unique mesh properties *i.e.,* the polygonal mesh faces. In this work we define operators for meshes, which exploit their unique structure and properties. This allows us to define an equivariant convolution and similarity-invariant input features.

2.0.5 Point Cloud. Arguably the simplest of all representations, the point cloud representation provides a no-frills approximation for

an underlying 3D shape. The close relationship to data acquisition, and ease of conversion from other representations, make the point cloud form a classic candidate for data analysis. Accordingly, recent effort has focused on developing techniques for point cloud shape analysis using neural networks. PointNet [Qi et al. 2017a] proposes to use $1x1$ convolutions followed by global max pooling for order invariance. In its followup work, PointNet++ [Qi et al. 2017b], points are partitioned to capture local structures better. Wang et al. [2018b] take into account local point neighborhood information, and perform dynamic updates driven by similarity calculation between points based on distance in feature space. While most point-based approaches focus on global or mid-level attributes, [Guerrero et al. 2018] proposed a network to estimate local shape properties, *e.g.*, normal and curvature from raw point clouds, while [Williams et al. 2018] learn a geometric prior for surface reconstruction from point clouds. Atzmon et al. [2018] define an efficient convolution operator on point clouds by mapping the point clouds functions into volumetric functions. This makes the method invariant to the order of points and robust to some deformations in the data. Recently, Li et al. [2018] presented PointCNN, which extends the notion of convolution from a local grid to an χ-convolution on points residing in its Euclidean neighborhood.

In this work, unlike previous work, we rely on mesh edges to provide non-uniform, geodesic neighborhood information and have a consistent number of conv-neighbors. Invariant feature computation is performed on the edges, while leveraging mesh decimation techniques, such as edge collapse, in a manner that adheres to shape geometry and topology.

3 OVERVIEW: APPLYING CNN ON MESHES

The most fundamental and commonly used 3D data representation in computer graphics is the non-uniform polygonal mesh; large flat regions use a small number of large polygons, while detailed regions use a larger number of polygons. A mesh explicitly represents the topology of a surface: faithfully describing intricate structures while disambiguating proximity from nearby surfaces (see Figure 3).

Applying the CNN paradigm directly onto triangular meshes, necessitates an analogous definition and implementation of the standard building blocks of a CNN: the convolution and pooling layers. Contrary to images which are represented on a regular grid of discrete values, the key challenge in mesh analysis is the inherent irregularity and non-uniformity of a mesh. In this work, we aim to exploit these challenging and unique properties, rather than bypass them. Accordingly, we design our network to deliberately apply convolution and pooling operations directly on the mesh elements, and avoid conversion to a regular and uniform representation.

3.0.1 Invariant Convolutions. In our setting, all shapes are assumed to be represented as manifold meshes, possibly with boundary edges. Such an assumption guarantees that each edge is incident to two faces (triangles) at most, and is therefore adjacent to either two or four other edges. The vertices of a face are ordered counterclockwise, defining two possible orderings for the four neighbors of every edge. For example, see Figure 4, where the 1-ring neighbors of e can be ordered as

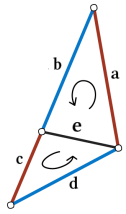


Fig. 4

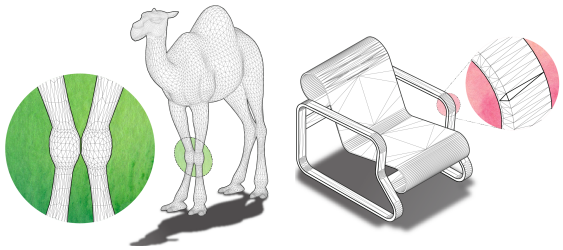
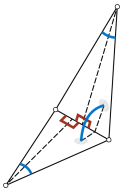


Fig. 3. Polygonal mesh advantages. *Left*: accurate portrayal of shape structure. The mesh, unlike the point cloud, can easily convey the distinct identities of the camel joints (zoom-in) through geodesic separation, despite their proximity in Euclidean space. *Right*: adaptive non-uniform representation. Large flat regions can be represented by a small number of large polygons, with detailed regions represented by a larger number of small polygons.

(a, b, c, d) or (c, d, a, b), depending on which face defines the first neighbor. This obscures the convolutional receptive field, hindering the formation of invariant features.

We take two actions to address this issue and guarantee invariance to similarity transformations (rotation, translation and scale) within our network. First, we carefully design the input descriptor of an edge to contain only relative geometric features that are inherently invariant to similarity transformations. Second, we aggregate the four 1-ring edges into two pairs of edges which have an ambiguity (*e.g.*, a and c, and b and d), and generate new features by applying simple symmetric functions on each pair (*e.g.*, $sum(a, c)$). The convolution is applied on the new symmetric features, thereby eliminating any order ambiguity.

3.0.2 Input Features. The input edge feature is a 5-dimensional vector for every edge: the dihedral angle, two inner angles and two edge-length ratios for each face. The edge ratio is between the length of the edge and the perpendicular (dotted) line for each adjacent face. We sort each of the two face-based features (inner angles and edge-length ratios), thereby resolving the ordering ambiguity and guaranteeing invariance. Observe that these features are all *relative*, making them invariant to *translation*, *rotation* and *uniform scale*.



3.0.3 Global Ordering. The global ordering of the edges is the order in which the edge data (input features) of a particular shape enters the network. This ordering has no influence during the convolutional stage since convolution is performed within local neighborhoods. By extension, fully convolutional tasks *e.g.*, segmentation are unaffected by it. For tasks that require global feature aggregation, such as classification, we follow the common practice suggested by Qi et al. [2017a] in PointNet, and place a global average pooling layer that connects between the convolutional part and the fully-connected part of the network. This layer renders the initial ordering inconsequential and thus guarantees invariance to transformations.

3.0.4 Pooling. Mesh pooling is accomplished by an edge collapse process, as illustrated in Figure 2 (b) and (c). In (b), the dashed edge is collapsing to a point, and, subsequently, the four incident

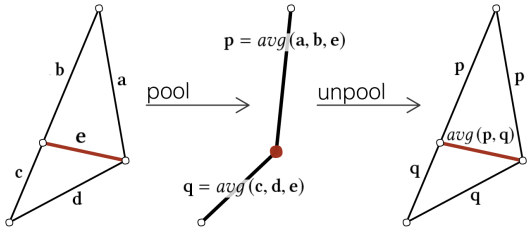


Fig. 5. Feature aggregation during mesh pooling and unpooling.

edges (blue) merge into the two (blue) edges in (c). Note that in this edge collapse operation, five edges are transformed into two. The operator is prioritized by the (smallest norm) edge features, thereby allowing the network to select which parts of the mesh to simplify, and which to leave intact. This creates a task-aware process, where the network learns to determine object part importance with respect to its mission (see Figure 1).

A notable advantage of the nature of our simplification, is that it provides flexibility with regards to the output dimensions of the pooling layer, just before it reaches the final fully connected layer. Pooling also contributes to robustness to initial mesh triangulation. While it does not provide equivariance to triangulation, in practice, by way of continuously collapsing edges and simplifying the mesh, we observe convergence to similar representations despite differences in initial tessellation.

4 METHOD

A grid-based (*e.g.,* image) representation conveniently provides both spatial neighbors (connectivity) and features in a single matrix. However, since irregular meshes do not conform to this format, we must define the features separately from the connectivity. We accomplish this by working within the standard constructs of a mesh.

A mesh is defined by the pair (V, F), where $V = \{\mathbf{v}_1, \mathbf{v}_2 \cdots\}$ is the set of vertex positions in $\mathbb{R}^3$, and F defines the connectivity (triplets of vertices for triangular meshes). Given (V, F), the mesh connectivity is also defined using *edges* E, a set of pairs of vertices.

All the mesh elements V, F and E can be associated with various features (such as normals or colors). In this work, E also holds a set of features. The edge features start out as a set of similarity-invariant geometric features (equivalent to RGB values in the case of an image), and develop a higher abstraction as they progress through the network layers.

In our setting, the mesh provides two attributes to the network: connectivity for convolutional neighbors and the initial geometric input features. The mesh vertices carry no meaning once the input features have been extracted. New vertex positions following edge collapse operations have no effect on the classification and segmentation tasks and they are computed for visualization purposes only.

In what follows, we expand and provide details about our mesh convolution, mesh pooling and mesh unpooling operations.

4.1 Mesh Convolution

We define a convolution operator for edges, where the spatial support is defined using the four incident neighbors (Figure 3). Recall that convolution is the dot product between a kernel k and a neighborhood, thus the convolution for an edge feature e and its four adjacent edges is:

$$e \cdot k_0 + \sum_{j=1}^{4} k_j \cdot e^j, \tag{1}$$

where e^j is the feature of the j^{th} convolutional neighbor of e. Note that as shown in Figure 4, the four neighbors of e, *i.e.,* (e^1, e^2, e^3, e^4), are either (a, b, c, d) or (c, d, a, b), such that each filter value operates on at most two possible edges (for example k_1 on a or c). To guarantee convolution invariance to the ordering of the input data, we apply a set of simple symmetric functions to the ambiguous pairs. This generates a new set of convolutional neighbors that are guaranteed to be order invariant. In our setting, the receptive field for an edge e is given by

$$(e^1, e^2, e^3, e^4) = (|a - c|, a + c, |b - d|, b + d). \tag{2}$$

This leads to a convolution operation that is oblivious to the initial ordering of the mesh elements, and will therefore produce the same output regardless of it. Recall that convolution of a multi-channel tensor with a kernel can be implemented with general matrix multiplication (*GEMM*): by expanding (or *unwrapping*) the image into a column matrix (*i.e., im2col* [Jia 2014]). Equivalently, we build an unwrapped matrix to perform the convolution operation efficiently.

In practice, we can use highly optimized batched operators (*e.g., conv2D*) by aggregating all edge features into a $n_c \times n_e \times 5$ feature-tensor, where n_e is the number of edges, n_c is the number of feature channels, and 5 is for the edge and the convolutional neighbors (equation 2). This matrix is multiplied by a matrix of the weights of the convolutions using standard *GEMM*.

Following the convolution operation, a new batched-feature-tensor is generated, where the new number of features is equal to the number of convolution kernels (just as in images). Note that after each pooling phase, the new connectivity will define the new convolutional neighbors for the next convolution.

4.2 Mesh Pooling

We extend conventional pooling to irregular data, by identifying three core operations that together generalize the notion of pooling:

1) define pooling region given adjacency
2) merge features in each pooling region
3) redefine adjacency for the merged features

For pooling on regular data such as images, adjacency is inherently implied and, accordingly, the pooling region is determined directly by the chosen kernel size. Since features in each region are merged (*e.g.,* via *avg* or *max*) in a way that yields another uniformly spaced grid, the new adjacency is once again inherently defined. Having addressed the three general pooling operations defined above, it is evident that conventional pooling is a special case of the generalized process.

Mesh pooling is another special case of generalized pooling, where adjacency is determined by the topology. Unlike images,

which have a natural reduction factor of, for example 4 for 2×2 pooling, we define mesh pooling as a series of edge collapse operations, where each such edge collapse converts five edges into two. Therefore, we can control the desired *resolution* of the mesh after each pooling operator, by adding a hyper-parameter which defines the number of target edges in the *pooled* mesh. During runtime, extracting mesh adjacency information requires querying special data structures that are continually updated (see [Berg et al. 2008] for details).

We prioritize the edge-collapse order (using a priority queue) by the magnitude of the edge features, allowing the network to select which parts of the mesh are relevant to solve the task. This enables the network to non-uniformly collapse certain regions which are least important to the loss. Recall that collapsing an edge which is adjacent to two faces results in a deletion of three edges (shown in Figure 2), since both faces become a single edge. Each face contains three edges: the minimum edge and two adjacent neighbors of the minimum edge (see minimum edge in red and adjacent neighbors in blue in Figure 2). Each of the features of the three edges in each face are merged into a new edge feature by taking the average over each feature channel.

Edge collapse is prioritized according to the strength of the features of the edge, which is taken as their ℓ_2-norm. The features are aggregated as illustrated in Figure 5, where there are two merge operations, one for each of the incident triangles of the minimum edge feature e, resulting in two new feature vectors (denoted p and q). The edge features in channel index i for both triangles is given by

$$p_i = avg(a_i, b_i, e_i), \text{ and, } q_i = avg(c_i, d_i, e_i), \tag{3}$$

After edge collapse, the half-edge data structure is updated for the subsequent edge collapses.

Finally, note that not every edge can be collapsed. An edge collapse yielding a non-manifold face is not allowed in our setting, as it violates the four convolutional neighbors assumption. Therefore, an edge is considered invalid to collapse if it has three vertices on the intersection of its 1-ring, or if it has two boundary vertices (see [Botsch et al. 2010] for details).

4.3 Mesh Unpooling

Unpooling is the (partial) inverse of the pooling operation. While pooling layers reduce the resolution of the feature activations (encoding or compressing information), unpooling layers increase the resolution of the feature activations (decoding or uncompressing information). The pooling operation records the history from the merge operations (*e.g.,* max locations), and uses them to expand the feature activation. Thus, unpooling does not have *learnable* parameters, and it is typically combined with convolutions to recover the original resolution lost in the pooling operation. The combination with the convolution effectively makes the unpooling a learnable operation.

Each mesh unpooling layer is paired with a mesh pooling layer, to upsample the mesh topology and the edge features. The unpooling layer reinstates the upsampled topology (prior to mesh pooling), by storing the connectivity prior to pooling. Note that upsampling the connectivity is a reversible operation (just as in images). For unpooled edge feature computation, we retain a graph which stores the adjacencies from the original edges (prior to pooling) to the new edges (after pooling). Each unpooled edge feature is then a weighted combination of the pooled edge features. The case of average unpooling is demonstrated in Figure 5.

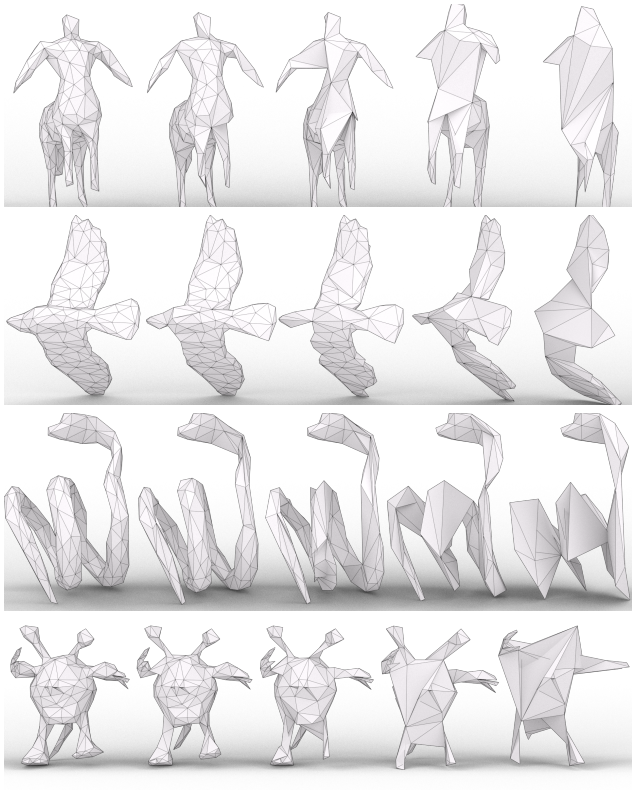


Fig. 6. Intermediate pooled meshes on the SHREC11 classification dataset.

5 EXPERIMENTS

MeshCNN is a general method for applying CNNs directly on triangular meshes, with many applications. Using the building blocks of MeshCNN described in Section 4, we can construct different network configurations for the purpose of solving different tasks. Like conventional CNNs, these building blocks provide a *plug-and-play* framework. For computational efficiency, in the pooling operation, we aggregate the features only once per pooling operation. While the edge sorting and collapse are performed sequentially, this relaxation allows performing the feature aggregation operation on a GPU, which improves the computational efficiency.

In what follows, we demonstrate MeshCNN performance on classification and segmentation tasks. Details on the network architectures used are given in Appendix A.

5.1 Data Processing

Across all sets, we simplified each mesh to *roughly* the same number of edges. Note that as mentioned earlier, MeshCNN does not require

Table 1. SHREC 30 class classification (comparisons taken from [2017]). Split 16 and 10 are the training splits, trained up to 200 epochs.

Classification SHREC		
Method	Split 16	Split 10
MeshCNN	**98.6** %	**91.0**%
GWCNN	96.6%	90.3%
GI	96.6%	88.6%
SN	48.4%	52.7%
SG	70.8%	62.6%

[Ezuz et al. 2017]

the same number of edges across all samples. However, similar to initial resize of images in CNNs, geometric mesh decimation helps to reduce the input resolution and with it the network capacity required for training. Since the classification task learns a global shape description, we usually use a lower resolution (750 edges), compared to the segmentation task (2250 edges).

5.1.1 Augmentation. Several forms of data augmentation exist for generating more data samples for the network. Note that since our input features are similarity-invariant, applying rotation, translation and isotropic scaling (same in x, y and z) does not generate new input features. However, we can generate new features by applying anisotropic scaling on the vertex locations in x, y and z, $< S_x, S_y, S_z >$ (each randomly sampled from a normal distribution $\mu = 1$ and $\sigma = 0.1$), which *will* change the input features to the network. We also shift vertices to different locations on the mesh surface by shifting towards a random vertex in its 1-ring. Furthermore, we augment the tessellation of each object by performing random edge flips. Due to the flexible input resolution, we can also collapse a small random set of edges prior to training.

5.2 Mesh Classification

5.2.1 SHREC. We performed classification on 30 classes from the SHREC dataset [Lian et al. 2011], with 20 examples per class. We follow the setup in [Ezuz et al. 2017], where split 16 and 10 are the number of training examples per class and we stop training after 200 epochs. Since we did not have the exact splits used in [Ezuz et al. 2017], our result is averaged over 3 randomly generated split 16 and 10 sets. Table 1 reports the results. For comparison, we take the evaluations directly from [Ezuz et al. 2017], who compare against: SG [Bronstein et al. 2011] (bag-of-features representation), SN [Wu et al. 2015] (volumetric CNN), GI [Sinha et al. 2016] (CNN on fixed geometry images) and finally GWCNN [2017] (learned geometry images). The advantage of our method is apparent. We visualize some examples of mesh pooling simplifications of this dataset in Figure 6. We observe that mesh pooling behaves in a consistent *semantic* manner (see Figure 11).

5.2.2 Cube Engraving. To illustrate the distinctive power of MeshCNN, we modeled a set of cubes with shallow icon engravings (see Figure 7). We use 23 classes from the MPEG-7 binary shape [Latecki and Lakamper 2000] dataset, with roughly 20 icons per class. We set aside three icons per class for the test set, and use the remainder for training. For each icon, we randomly sample 10 different locations (position, rotation and cube face) to inset the icon. Each cube has approximately 500 faces, which means that detailed shapes have fewer triangles in the flat areas, while less detailed shapes have more triangles in flat areas. This set contains a total of 4600 shapes with 3910 / 690 in the train / test split. We plan to release this dataset as well as the data synthesis code after publication.

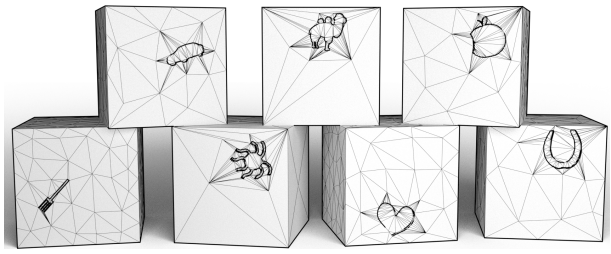


Fig. 7. Engraved cubes classification dataset. We generate 23 different classes (*e.g.,* car, heart, apple, etc.) by extruding stickers from MPEG-7 [Latecki and Lakamper 2000], and randomly placing them on the cube.

Table 2. Results on engraved cubes (shown in Figure 7).

Cube Engraving Classification		
Method	Input Res	Test Acc
MeshCNN	750	**92.16**%
PointNet++	4096	64.26%
UGSCNN	750	61.97%

We train MeshCNN to classify the cubes. We show the quantitative results in Table 2. To visualize the effect of mesh pooling on the classification task, we extracted the intermediate results following each mesh pooling operation (shown in Figure 8). Observe how MeshCNN learned to reduce the edges irrelevant to the classification task (flat cube surfaces) while preserving the edges within and surrounding the icon engraving.

We also trained point-based and mesh-based approaches on this set and show results in Table 2. We trained PointNet++ using a geodesic sampling of the mesh and used the coordinates as well as the normals for the input features. In addition, we trained a spherical mesh-based CNN approach UGSCNN [Jiang et al. 2019]. We mapped the cubes to an icosahedral spherical mesh representation for the input. While this example may be considered contrived, it is meant to highlight that MeshCNN excels on 3D shapes that contain a large variance in geometric resolution.

5.3 Mesh Segmentation

Another application of MeshCNN is consistent shape segmentation, which is an important building block for many applications in shape analysis and synthesis. We used supervised learning to train MeshCNN to predict, for every edge, the probability of belonging to a particular segment on the COSEG [Wang et al. 2012] and Human Body Segmentation [Maron et al. 2017] datasets. Since both datasets provide ground truth segmentation per face, we generated edge-level semantic labeling on the simplified meshes based on the labels from the original resolution.

The most straightforward MeshCNN semantic segmentation configuration would be to use a sequence of mesh convolution layers

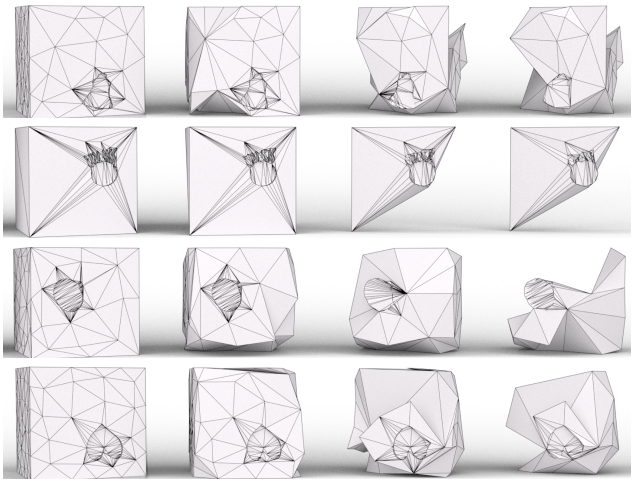


Fig. 8. MeshCNN trained to predict the class of icon engraving. Observe how the network learns to preserve important edges and remove redundant edges with regards to the classification task.

(along with normalization and non-linear activation units). However, incorporating mesh pooling enables MeshCNN to learn a segmentation-driven edge collapse. Recall that mesh pooling reduces the input mesh resolution, which is no longer consistent with the ground truth edge-level labels. To this end, we use the mesh unpooling layer to upsample the resolution back to the original input size.

5.3.1 COSEG. We evaluate the performance of MeshCNN on the task of segmentation on the COSEG dataset, which contains three large sets: *aliens*, *vases* and *chairs* containing 200, 300 and 400 models in each respectively. We split each shape category into 85%/15% train/test splits. We compare to PointNet, PointNet++ and PointCNN and report the best accuracy for all methods in Table 3. Our technique achieves better results than all the other methods on this dataset.

We believe that this is due to the fact that our network is tailored to the mesh structure, which gives it an advantage over the other strategies. To further demonstrate this, we report also the results for the case of random pooling (collapsed edges are picked randomly) and exhibit that this change reduces the performance of the network. In addition, the final segmentation predictions from MeshCNN semantic segmentation network with pooling and unpooling layers on a held out test set is shown in Figure 9. This also manifests how the pooling performed is adapted to the target problem.

5.3.2 Human Body Segmentation. We evaluated our method on the human body segmentation dataset proposed by [Maron et al. 2017]. The dataset consists of 370 training models from SCAPE [Anguelov et al. 2005], FAUST [Bogo et al. 2014], MIT [Vlasic et al. 2008] and Adobe Fuse [Adobe 2016], and the test set is 18 models from SHREC07 [Giorgi et al. 2007] (humans) dataset. The models are manually segmented into eight labels according to the labels in [Kalogerakis et al. 2010]. Recently, [Poulenard and Ovsjanikov 2018] reported results on this dataset for their method with comparisons to GCNN [Masci et al. 2015], PointNet++ [Qi et al. 2017b], Dynamic

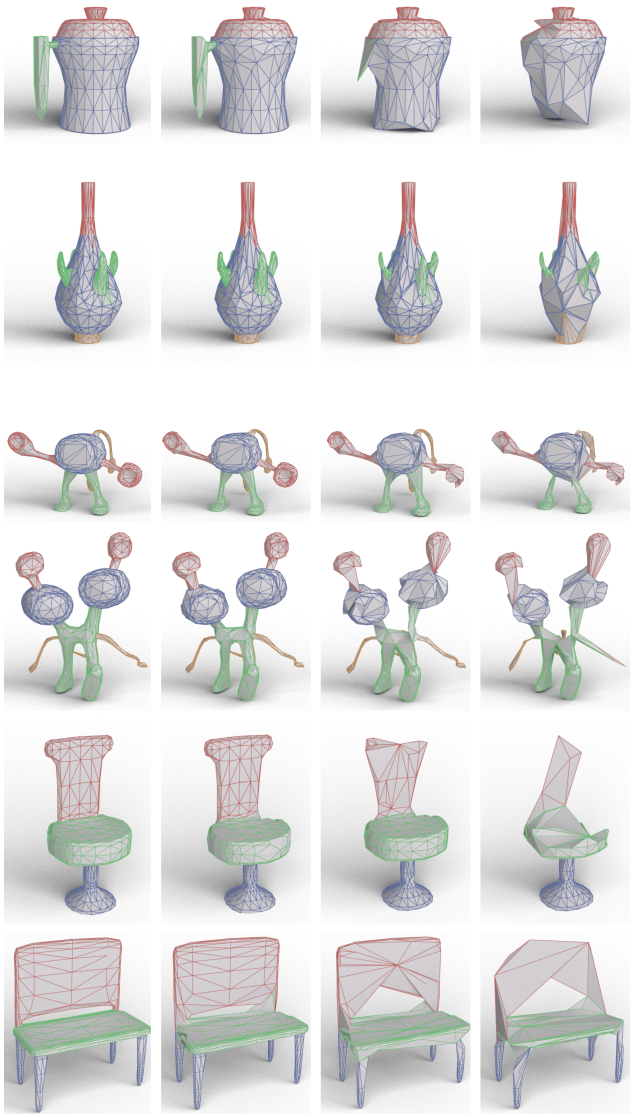


Fig. 9. Semantic segmentation test set results. The segmentation prediction per edge is shown on the left, followed by the intermediate simplified meshes after each pooling layer. For visualization purposes, the edges in the intermediate meshes are colored with the final segmentation predictions. Note the top row: the entire vase handle has collapsed to a single edge.

Table 3. MeshCNN evaluations on COSEG segmentation.

COSEG Segmentation			
Method	Vases	Chairs	Telealiens
MeshCNN (*UNet*)	**97.27%**	**99.63%**	**97.56%**
MeshCNN (*rand. pool*)	96.64%	99.23%	97.01%
PointNet	91.5%	70.2%	54.4%
PointNet++	94.7%	98.9%	79.1%
PointCNN	96.37%	99.31%	97.40%

Table 4. Human body segmentation results (cited comparisons taken from [Poulenard and Ovsjanikov 2018]). Latest comparison taken from [Haim et al. 2018].

Human Body Segmentation			
Method	# Features	Accuracy	
MeshCNN	5	**92.30%**	
SNGC	3	91.02%	[2018]
Toric Cover	26	88.00%	[2018]
PointNet++	3	90.77%	[2018]
DynGraphCNN	3	89.72%	[2018]
GCNN	64	86.40%	[2018]
MDGCNN	64	89.47%	[2018]

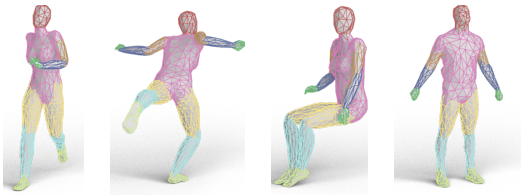


Fig. 10. Human shape segmentation results on [Maron et al. 2017].

Graph CNN [Wang et al. 2018b], and Toric Cover [Maron et al. 2017]. We take the reported results directly from [Poulenard and Ovsjanikov 2018] and list them in Table 4. We added to the table the recent results by Haim et al. [2018], reporting state-of-the-art results on this set. Also in this case, MeshCNN has an advantage over the other methods (some are graph/manifold based and others are point based), which we believe results from the adaptivity of MeshCNN both to the mesh structure and the task at hand. Figure 10 presents some qualitative results of MeshCNN.

5.4 Additional Evaluations

5.4.1 Computation Time. Our non-optimized PyTorch [Paszke et al. 2017] implementation takes an average of 0.21/0.13 seconds per example when training on segmentation / classification with 2250/750 edges using a GTX 1080 TI graphics card.

5.4.2 Tessellation Robustness. We examine the robustness of our method to differences in triangulations through several qualitative and quantitative experiments using the COSEG segmentation dataset. To this end, we train on the original (reference) dataset, and compare the test set accuracy of three different sets: reference test set, remeshed test set and vertex perturbation test set. The remeshed test set was generated using a different meshing procedure than used on the reference set (Blender vs. MeshLab). The vertex perturbation set was generated by randomly moving 30% of the vertices along the surface, by shifting towards a random vertex in its 1-ring. The quantitative and qualitative results can be seen in Table 5 and Figure 12 respectively. These results imply the network has learned to be resilient to these types of variations. Note that the tesselations in the reference set are extremely different from the meshings in the remeshed set (see examples in Figure 12). In particular, the set with the largest difference is the chair set (due to the different remeshing procedure in large flat regions). Indeed, this domain gap between the remeshed chair sets led to a drop (10 %) in the chair class in Table 5.

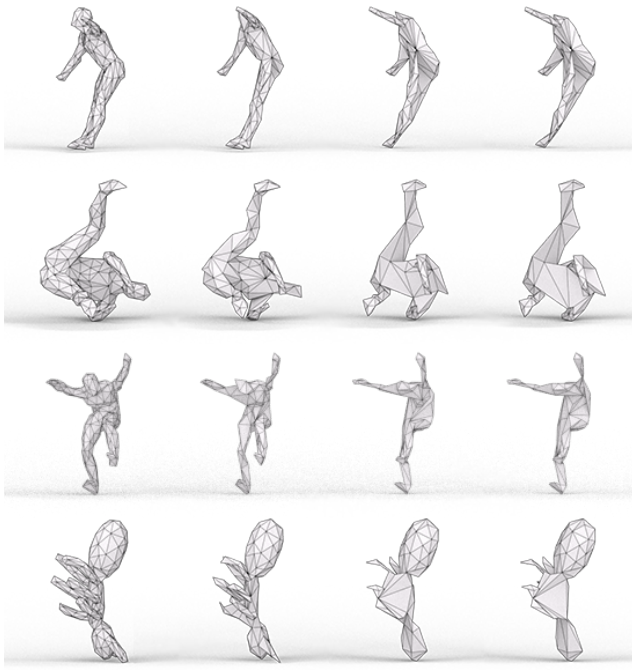


Fig. 11. Adaptive pooing has the potential to expose semantics within a class. We observe a consistent semantic pooling within the same class on the task of shape classification (SHREC). For example, the heads of people are analogously pooled, while similar shape attributes are pooled differently for another class (bottom).

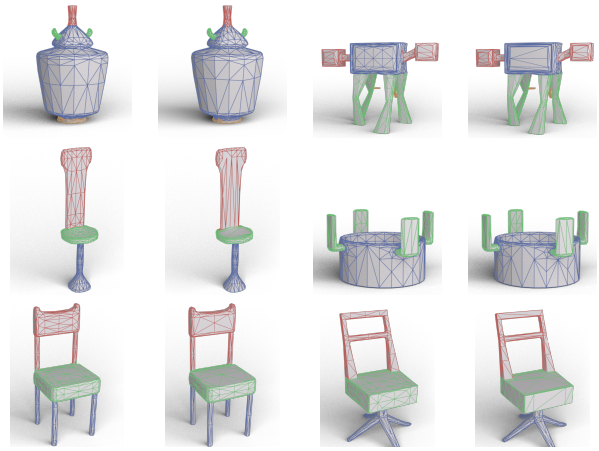


Fig. 12. Robustness to different triangulations (semantic segmentation). Left: test shapes, right: alternate triangulation. Note the significant difference in triangulation in the chair class.

Table 5. Quantitative results on two robustness sets.

Robustness			
Set	Vases	Chairs	Aliens
Reference	97.27%	99.63%	97.56%
Remeshing	96.33%	88.98%	97.77%
Perturbations	97.12%	99.60%	97.34%

Stretched Set 3D points geometric

Fig. 13. Robustness of differential features. We evaluate generalization on stretched set (left), using MeshCNN with geometric features compared to standard 3D points (edge midpoints). While both achieve high accuracy on the standard test set, MeshCNN geometric features (right) generalizes well to the stretched set compared to the 3D points case (middle).

5.4.3 Invariant Features. A noteworthy advantage of working with relative features is that MeshCNN is guaranteed to be invariant to rotations, translations and uniform scaling. Inherently, the commonly used Cartesian coordinates, are sensitive to rigid transformations. To illustrate that, we train MeshCNN on the task of semantic segmentation: (i) using invariant geometric features and, (ii) using the edge midpoints (x, y, z) as input features. To assess the learning generalization, we apply non-uniform scaling along the vertical axis (without training on these types of augmentations). Our relative geometric features achieve 98.44%, compared with 99.63% on the standard test set, while the absolute coordinates deteriorate to 78.27%, compared to 99.11% on the standard test set. Note that while our geometric features are not invariant to non-uniform scaling, they generalize better due to their insensitivity to positioning.

6 DISCUSSION AND FUTURE WORK

We presented MeshCNN, a general method for employing neural networks directly on irregular triangular meshes. The key contribution of our work is the definition and application of convolution and pooling operations tailored to irregular and non-uniform structures. These operations facilitate a direct analysis of shapes represented as meshes in their native form, and hence benefit from the unique properties associated with the representation of surface manifolds with non-uniform structures.

6.0.1 Invariant Convolutions. Our choice of mesh edges as basic building blocks upon which the network operates, is of great importance, as the edge set dictates a simple means to define a local, fixed sized neighborhood for convolutions over irregular structures. By exploiting the unique symmetries specific to triangular meshes, we disambiguate the neighbor ordering duality to enable invariance to transformations. We complement this effort with our choice of input edge features, which are carefully curated to contain solely relative geometric properties rather than absolute positions. We observed that our differential features not only provide invariance to similarity transformations, but also inhibit overfitting, as opposed to the usage of absolute Cartesian coordinates. The generalization of the network was further demonstrated via its ability to perform semantically similar pooling across different objects, naturally leading to better results. Thus, unlike common representations (*e.g.*, point-based), the Cartesian coordinates of the vertices are ignored, and local and non-local features are position-oblivious, allowing better generalization of shape features, and facilitating invariance to similarity transformations. We emphasize that we use vertex positions solely for displaying the evolving meshes, but their positions have no impact on the task.

6.0.2 Spatially Adaptive Pooling. We developed a pooling operation carried out by edge collapse, based on the learned edge features, leading to task-driven pooling guided by the network loss function. Visualizing the series of features the network determined were important, helps gain insights with regards to what the network had actually learned. Investigating this powerful mechanism may lead to a better understanding of the neural net behavior. Mesh pooling provides useful insights into model interpretability and generalization, and we believe gives better clues about robustness to real-world use cases. Instead of using a network-learned edge collapse, it is possible to use a strictly geometric edge collapse (*e.g.,* quadratic edge collapse [Garland and Heckbert 1997]), which gives a predetermined and fixed simplification for every mesh (regardless of the loss function or network weights). However, if the simplification is not dynamically determined by the network then it will not provide the same unique visual insights that we showed in Figures 1, 8, 9 and 11.

6.0.3 Limitations and Future Work. An opportunity for future work could be to apply the spatially-adaptive irregular task-driven pooling to image-based CNN tasks. For instance, high resolution image segmentation typically produces a low resolution segmentation map and upsamples it, possibly with skip connections. The pooling in MeshCNN semantically simplifies regions with uniform features, while preserving complex ones; therefore, in the future, we are interested in applying similar irregular pooling for image segmentation tasks to obtain high resolution segmentation maps, where large uniform regions in the image will be represented by a small number of triangles.

Currently, our implementation performs sequential edge collapses. This operation can potentially be parallelized on a GPU by using a parallel sorting technique [Bozidar and Dobravec 2015] for the edge features (calculated only once per pooling operation) and ensuring that only non-adjacent edges are collapsed simultaneously. Clearly, pooling the features in a non-sequential manner may differ from the sequential one.

Despite the robustness of our method to different triangulations (as demonstrated by our experiments), MeshCNN, like any other network, relies on good training data for a successful generalization. In this sense, much like adversarial noise in images, MeshCNN is vulnerable to adversarial remeshing attacks that may hinder performance. Robustness to such adversarial attacks is therefore an interesting direction for future work.

Another avenue for future research is generative modeling, mesh-upsampling and attribute synthesis for the modification of existing meshes. Our idea is to apply vertex-split in reverse order of the edge collapse operations by bookkeeping the list of edge collapses. Thus, when synthesizing new meshes, the network decides which vertex to split, for example, by splitting vertices that are adjacent to edges with high feature values.

Finally, we identify a promising endeavor in the extension of our proposed strategy, designed for triangular meshes, to general graphs. Edge-collapse based pooling and unpooling can be applied to general graphs in a similar manner to our proposed MeshCNN. As for convolution, we must consider an appropriate alternative suitable for the irregularity of general graphs. An interesting approach may be to process the edges using an attention mechanism [Monti et al. 2018].

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their helpful comments. We also thank Yaron Lipman for his insightful suggestions. This research was supported by ERC-StG grant no. 757497 (SPADE) and by the Israel Science Foundation as part of the ISF-NSFC joint program grant number (2217/15, 2472/17), and partially supported by ISF grant 2366/16. Shachar Fleishman started this work prior to joining Amazon.

REFERENCES

Adobe. 2016. Adobe Fuse 3D Characters. https://www.mixamo.com.

Dragomir Anguelov, Praveen Srinivasan, Daphne Koller, Sebastian Thrun, Jim Rodgers, and James Davis. 2005. SCAPE: Shape Completion and Animation of People. In *ACM SIGGRAPH 2005 Papers (SIGGRAPH '05)*. ACM, New York, NY, USA, 408–416. https://doi.org/10.1145/1186822.1073207

James Atwood and Don Towsley. 2016. Diffusion-convolutional Neural Networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems (NIPS'16)*. Curran Associates Inc., USA, 2001–2009. http://dl.acm.org/citation.cfm?id=3157096.3157320

Matan Atzmon, Haggai Maron, and Yaron Lipman. 2018. Point Convolutional Neural Networks by Extension Operators. *ACM Trans. Graph.* 37, 4 (July 2018), 71:1–71:12.

Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. 2008. *Computational Geometry: Algorithms and Applications* (3rd ed. ed.). Springer-Verlag TELOS, Santa Clara, CA, USA.

Federica Bogo, Javier Romero, Matthew Loper, and Michael J Black. 2014. FAUST: Dataset and evaluation for 3D mesh registration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 3794–3801.

Davide Boscaini, Jonathan Masci, Simone Melzi, Michael M Bronstein, Umberto Castellani, and Pierre Vandergheynst. 2015. Learning class-specific descriptors for deformable shapes using localized spectral convolutional networks. In *Computer Graphics Forum*, Vol. 34. Wiley Online Library, 13–23.

Davide Boscaini, Jonathan Masci, Emanuele Rodolà, and Michael Bronstein. 2016. Learning shape correspondence with anisotropic convolutional neural networks. In *Advances in Neural Information Processing Systems*. 3189–3197.

Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, and Bruno Lévy. 2010. *Polygon mesh processing*. AK Peters/CRC Press.

Darko Bozidar and Tomaz Dobravec. 2015. Comparison of parallel sorting algorithms. *CoRR* abs/1511.03404 (2015).

Andrew Brock, Theodore Lim, J.M. Ritchie, and Nick Weston. 2016. Generative and Discriminative Voxel Modeling with Convolutional Neural Networks. In *NIPS 3D Deep Learning Workshop*.

Alexander M Bronstein, Michael M Bronstein, Leonidas J Guibas, and Maks Ovsjanikov. 2011. Shape google: Geometric words and expressions for invariant shape retrieval. *ACM Transactions on Graphics (TOG)* 30, 1 (2011), 1.

Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. 2017. Geometric Deep Learning: Going beyond Euclidean data. *IEEE Signal Process. Mag.* 34, 4 (2017), 18–42. https://doi.org/10.1109/MSP.2017.2693418

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. 2014. Spectral Networks and Locally Connected Networks on Graphs. In *International Conference on Learning Representations (ICLR)*.

C. Cangea, P. Velickovic, N. Jovanovic, T. Kipf, and P. Lio. 2018. Towards Sparse Hierarchical Graph Classifiers. In *NeurIPS Workshop on Relational Representation Learning*.

Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. 2018. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence* 40, 4 (2018), 834–848.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*. 3844–3852.

Danielle Ezuz, Justin Solomon, Vladimir G. Kim, and Mirela Ben-Chen. 2017. GWCNN: A Metric Alignment Layer for Deep Shape Analysis. *Computer Graphics Forum* (2017). https://doi.org/10.1111/cgf.13244

Xifeng Gao, Daniele Panozzo, Wenping Wang, Zhigang Deng, and Guoning Chen. 2017. Robust structure simplification for hex re-meshing. *ACM Transactions on Graphics* 36, 6 (2017).

Michael Garland and Paul S Heckbert. 1997. Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 209–216.

Daniela Giorgi, Silvia Biasotti, and Laura Paraboschi. 2007. Shape retrieval contest 2007: Watertight models track. *SHREC competition* 8, 7 (2007).

Francisco Gomez-Donoso, Alberto Garcia-Garcia, J Garcia-Rodriguez, Sergio Orts-Escolano, and Miguel Cazorla. 2017. Lonchanet: A sliced-based cnn architecture for real-time 3d object recognition. In *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 412–418.

Benjamin Graham, Martin Engelcke, and Laurens van der Maaten. 2017. 3D Semantic Segmentation with Submanifold Sparse Convolutional Networks. *CoRR* abs/1711.10275 (2017).

Paul Guerrero, Yanir Kleiman, Maks Ovsjanikov, and Niloy J. Mitra. 2018. PCPNet: Learning Local Shape Properties from Raw Point Clouds. *Computer Graphics Forum* 37, 2 (2018), 75–85. https://doi.org/10.1111/cgf.13343

Niv Haim, Nimrod Segol, Heli Ben-Hamu, Haggai Maron, and Yaron Lipman. 2018. Surface Networks via General Covers. *CoRR* abs/1812.10705 (2018).

Rana Hanocka, Noa Fish, Zhenhua Wang, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. 2018. ALIGNet: Partial-Shape Agnostic Alignment via Unsupervised Learning. *ACM Trans. Graph.* 38, 1, Article 1 (Dec. 2018), 14 pages. https://doi.org/10.1145/3267347

Mikael Henaff, Joan Bruna, and Yann LeCun. 2015. Deep Convolutional Networks on Graph-Structured Data. *CoRR* abs/1506.05163 (2015).

Hugues Hoppe. 1997. View-dependent refinement of progressive meshes. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 189–198.

Hugues Hoppe. 1999. New quadric metric for simplifying meshes with appearance attributes. In *Visualization'99. Proceedings*. IEEE, 59–510.

Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. 1993. Mesh optimization. , 19–26 pages.

Yangqing Jia. 2014. Learning Semantic Image Representations at a Large Scale. (2014).

Chiyu Max Jiang, Jingwei Huang, Karthik Kashinath, Prabhat, Philip Marcus, and Matthias Niessner. 2019. Spherical CNNs on Unstructured Grids. In *International Conference on Learning Representations*. https://openreview.net/forum?id=Bkl-43C9FQ

Evangelos Kalogerakis, Melinos Averkiou, Subhransu Maji, and Siddhartha Chaudhuri. 2017. 3D shape segmentation with projective convolutional networks. In *Proc. CVPR*, Vol. 1. 8.

Evangelos Kalogerakis, Aaron Hertzmann, and Karan Singh. 2010. Learning 3D mesh segmentation and labeling. *ACM Transactions on Graphics (TOG)* 29, 4 (2010), 102.

I. Kokkinos, M. M. Bronstein, R. Litman, and A. M. Bronstein. 2012. Intrinsic shape context descriptors for deformable shapes. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 159–166.

Ilya Kostrikov, Zhongshi Jiang, Daniele Panozzo, Denis Zorin, and Burna Joan. 2018. Surface Networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

Longin Jan Latecki and Rolf Lakamper. 2000. Shape similarity measure based on correspondence of visual parts. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22, 10 (2000), 1185–1190.

Yann LeCun. 2012. Learning invariant feature hierarchies. In *European conference on computer vision*. Springer, 496–505.

Yangyan Li, Rui Bu, Mingchao Sun, and Baoquan Chen. 2018. PointCNN. *CoRR* abs/1801.07791 (2018).

Yangyan Li, Soren Pirk, Hao Su, Charles R Qi, and Leonidas J Guibas. 2016. FPNN: Field probing neural networks for 3D data. In *Advances in Neural Information Processing Systems (NIPS)*. 307–315.

Z Lian, A Godil, B Bustos, M Daoudi, J Hermans, S Kawamura, Y Kurita, G Lavoua, and P Dp Suetens. 2011. Shape retrieval on non-rigid 3D watertight meshes. In

Eurographics Workshop on 3D Object Retrieval (3DOR).
Or Litany, Alexander M. Bronstein, Michael M. Bronstein, and Ameesh Makadia. 2018. Deformable Shape Completion With Graph Convolutional Autoencoders. In *CVPR*.
Haggai Maron, Meirav Galun, Noam Aigerman, Miri Trope, Nadav Dym, Ersin Yumer, Vladimir G Kim, and Yaron Lipman. 2017. Convolutional neural networks on surfaces via seamless toric covers. *ACM Trans. Graph* 36, 4 (2017), 71.
Jonathan Masci, Davide Boscaini, Michael Bronstein, and Pierre Vandergheynst. 2015. Geodesic convolutional neural networks on riemannian manifolds. In *Proceedings of the IEEE international conference on computer vision workshops*. 37–45.
Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. 2017. Geometric deep learning on graphs and manifolds using mixture model CNNs. In *Proc. CVPR*, Vol. 1. 3.
Federico Monti, Oleksandr Shchur, Aleksandar Bojchevski, Or Litany, Stephan Gunnemann, and Michael M. Bronstein. 2018. Dual-Primal Graph Convolutional Networks. *CoRR* abs/1806.00770 (2018).
Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. 2016. Learning Convolutional Neural Networks for Graphs. In *International Conference on Machine Learning (ICML)*.
Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. In *NIPS-W*.
Adrien Poulenard and Maks Ovsjanikov. 2018. Multi-directional Geodesic Neural Networks via Equivariant Convolution. In *SIGGRAPH Asia 2018 Technical Papers (SIGGRAPH Asia '18)*. ACM, New York, NY, USA, Article 236, 14 pages. https://doi.org/10.1145/3272127.3275102
Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. 2017a. Pointnet: Deep learning on point sets for 3d classification and segmentation. *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE* 1, 2 (2017), 4.
Charles R. Qi, Hao Su, Matthias Niessner, Angela Dai, Mengyuan Yan, and Leonidas J. Guibas. 2016. Volumetric and multi-view CNNs for object classification on 3d data. In *Computer Vision and Pattern Recognition (CVPR)*. 5648–5656.
Charles R. Qi, Li Yi, Hao Su, and Leonidas J Guibas. 2017b. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. In *Advances in Neural Information Processing Systems (NIPS)*. 5105–5114.
Anurag Ranjan, Timo Bolkart, Soubhik Sanyal, and Michael J. Black. 2018. Generating 3D faces using Convolutional Mesh Autoencoders. In *European Conference on Computer Vision (ECCV)*. Springer International Publishing, 725–741.
Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. 2017. OctNet: Learning deep 3D representations at high resolutions. In *Computer Vision and Pattern Recognition (CVPR)*.
Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*. Springer, 234–241.
Szymon Rusinkiewicz and Marc Levoy. 2000. QSplat: A Multiresolution Point Rendering System for Large Meshes. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '00)*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 343–352. https://doi.org/10.1145/344779.344940
Kripasindhu Sarkar, Basavaraj Hampiholi, Kiran Varanasi, and Didier Stricker. 2018. Learning 3D Shapes as Multi-Layered Height-maps using 2D Convolutional Networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 71–86.
Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. 2013. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229* (2013).
Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).
Ayan Sinha, Jing Bai, and Karthik Ramani. 2016. Deep learning 3D shape surfaces using geometry images. In *European Conference on Computer Vision*. Springer, 223–240.
Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Millers. 2015. Multi-view Convolutional Neural Networks for 3D Shape Recognition. In *International Conference on Computer Vision (ICCV)*.
F. P. Such, S. Sah, M. A. Dominguez, S. Pillai, C. Zhang, A. Michael, N. D. Cahill, and R. Ptucha. 2017. Robust Spatial Filtering With Graph Convolutional Neural Networks. *IEEE Journal of Selected Topics in Signal Processing* 11, 6 (Sept 2017), 884–896.
Marco Tarini, Nico Pietroni, Paolo Cignoni, Daniele Panozzo, and Enrico Puppo. 2010. Practical quad mesh simplification. In *Computer Graphics Forum*, Vol. 29. Wiley Online Library, 407–418.
Maxim Tatarchenko, Jaesik Park, Vladlen Koltun, and Qian-Yi Zhou. 2018. Tangent Convolutions for Dense Prediction in 3D. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 3887–3896.
Lyne P. Tchapmi, Christopher B. Choy, Iro Armeni, JunYoung Gwak, and Silvio Savarese. 2017. SEGCloud: Semantic Segmentation of 3D Point Clouds. In *3DV*.
Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph Attention Networks. In *International Conference on Learning Representations*.
Nitika Verma, E. Boyer, and Jakob Verbeek. 2018. FeaStNet: Feature-Steered Graph Convolutions for 3D Shape Analysis. In *CVPR*.

Table 6. Network configurations. The segmentation network has a symmetric up network. Adam optimization, $lr = 0.0002$, and group norm ($g = 16$). Data augmentation with 5% edge flips and 20% vertex perturbations.

Classification	Segmentation (Down)
MeshConv $f_{in} \times 32$	ResConv $f_{in} \times 32$
MeshPool $\longrightarrow$ 600	MeshPool $\longrightarrow$ 1800
MeshConv 32×64	ResConv 32×64
MeshPool $\longrightarrow$ 450	MeshPool $\longrightarrow$ 1350
MeshConv 64×128	ResConv 64×128
MeshPool $\longrightarrow$ 300	MeshPool $\longrightarrow$ 600
MeshConv 128×256	ResConv 128×256
MeshPool $\longrightarrow$ 279	
GlobalAvgPool	
FC 256×100	
FC 100×30	

Daniel Vlasic, Ilya Baran, Wojciech Matusik, and Jovan Popović. 2008. Articulated mesh animation from multi-view silhouettes. In *ACM Transactions on Graphics (TOG)*, Vol. 27. ACM, 97.
Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun, and Xin Tong. 2017. O-CNN: Octree-based Convolutional Neural Networks for 3D Shape Analysis. *ACM Trans. Graph.* 36, 4, Article 72 (July 2017), 11 pages. https://doi.org/10.1145/3072959.3073608
Yunhai Wang, Shmulik Asafi, Oliver van Kaick, Hao Zhang, Daniel Cohen-Or, and Baoquan Chen. 2012. Active co-analysis of a set of shapes. *ACM Transactions on Graphics (TOG)* 31, 6 (2012), 165.
Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. 2018a. Dynamic graph CNN for learning on point clouds. *arXiv preprint arXiv:1801.07829* (2018).
Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. 2018b. Dynamic Graph CNN for Learning on Point Clouds. *arXiv preprint arXiv:1801.07829* (2018).
Francis Williams, Teseo Schneider, Claudio Silva, Denis Zorin, Joan Bruna, and Daniele Panozzo. 2018. Deep Geometric Prior for Surface Reconstruction. *arXiv preprint arXiv:1811.10943* (2018).
Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 2015. 3D shapenets: A deep representation for volumetric shapes. In *Computer Vision and Pattern Recognition (CVPR)*. 1912–1920.
Haotian Xu, Ming Dong, and Zichun Zhong. 2017. Directionally Convolutional Networks for 3D Shape Segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*. 2698–2707.
Li Yi, Hao Su, Xingwen Guo, and Leonidas Guibas. 2017. SyncSpecCNN: Synchronized Spectral CNN for 3D Shape Segmentation. In *Computer Vision and Pattern Recognition (CVPR)*.
Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. 2018. Hierarchical Graph Representation Learning with Differentiable Pooling. In *Advances in Neural Information Processing Systems*. 4805–4815.
Tinghui Zhou, Richard Tucker, John Flynn, Graham Fyffe, and Noah Snavely. 2018. Stereo Magnification: Learning View Synthesis Using Multiplane Images. *ACM Trans. Graph.* 37, 4 (July 2018), 65:1–65:12.

A TRAINING CONFIGURATIONS

For classification we use the same network architecture for the SHREC and Cube engraving datasets (starts with 750 edges). For the segmentation task, for both the COSEG and human body datasets, we use a Unet [Ronneberger et al. 2015] type network (starts with 2250 edges). We detail the network configurations and learning parameters in Table 6.