


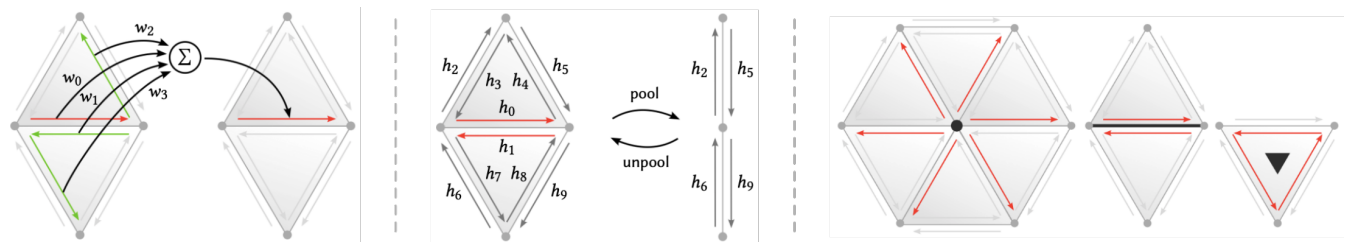


# HalfedgeCNN for Native and Flexible Deep Learning on Triangle Meshes

I. Ludwig  D. Tyson  M. Campen 

Osnabrück University, Germany



**Figure 1:** *HalfedgeCNN* revolves around a triangle mesh's halfedges as central entities. Convolution, pooling, and unpooling operators are defined directly on halfedge neighborhoods. In contrast to other mesh entities (vertices, edges, faces), halfedges combine multiple advantages such as well-defined orientation, constant neighborhood structure, and unique relations to other mesh entities. This allows defining versatile neural networks that, in a sense, operate natively on meshes.

## Abstract

We describe *HalfedgeCNN*, a collection of modules to build neural networks that operate on triangle meshes. Taking inspiration from the (edge-based) *MeshCNN*, convolution, pooling, and unpooling layers are consistently defined on the basis of halfedges of the mesh, pairs of oppositely oriented virtual instances of each edge. This provides benefits over alternative definitions on the basis of vertices, edges, or faces. Additional interface layers enable support for feature data associated with such mesh entities in input and output as well. Due to being defined natively on mesh entities and their neighborhoods, lossy resampling or interpolation techniques (to enable the application of operators adopted from image domains) do not need to be employed. The operators have various degrees of freedom that can be exploited to adapt to application-specific needs.

## CCS Concepts

• **Computing methodologies** → **Shape analysis; Mesh models; Neural networks;**

## 1. Introduction

The impressive advances achieved in recent years in a variety of application areas by neural networks, in particular convolutional neural networks (CNNs), have sparked interest in applying such techniques beyond their classical domains (most prominently 2D pixel and 3D voxel arrays). One recent focus is on surfaces, i.e. 2-manifolds. Such domains are more complex than the 2D plane, while 3D volumetric approaches are unnecessarily general as they do not exploit the sub-manifold nature of surfaces.

While, as briefly reviewed in Sec. 2, there are ways to phrase learning tasks on 2-manifolds as (multiple) 2D learning problems, as well as ways to reduce the overhead when applying overly general 3D approaches, it is natural to ask for neural network architectures tailored directly to the surface setting. The key ingredients

of CNNs being operators for convolution, pooling, and unpooling, a central challenge lies in defining those in an adequate way on 2-manifolds.

Probably the most common and flexible geometric representation of surfaces is via triangle meshes. A variety of approaches to defining the above operators on surface meshes have been explored in recent years. Their less regular structure in comparison to pixel or voxel grids brings about challenges regarding the definition of suitable convolution and pooling operators. For instance, a convolution operator requires an identical neighborhood structure everywhere on the domain so as to effectively enable weight sharing, a key property of CNNs [LBBH98, FM82]. This can be approached using interpolation or resampling techniques that effectively simulate a constant neighborhood structure in a local manner [MBBV15, BMRB16].

More recently, some works have proposed ways to define convolution operators that consume mesh-based data more directly, or natively. One example is SpiralNet [LDCK18] and variations thereof [GCBZ19, BBP\*19]; it establishes a linear ordering of a vertex' neighboring vertices, so as to enable the application of 1D convolutions or RNNs. Another example is MeshCNN [HHF\*19]; it operates on data associated with a mesh's edges and exploits the local edge neighborhood, which (in contrast to vertex neighborhoods) is constant in a triangle mesh, to define a convolution operator.

### 1.1. Contribution

We describe HalfedgeCNN, a network that revolves around the concept of *halfedges* in a triangle mesh. It can be viewed as a generalization of MeshCNN, and it in particular:

- removes the need to restrict to symmetric filters due to orientation ambiguities;
- increases expressiveness by being able to represent a broader range of functions;
- increases flexibility regarding the choice of convolution neighborhoods and pooling rules;
- enables dealing with data that is not edge based but vertex based, face based, or oriented-edge based in a more direct manner.

Like in the classical halfedge mesh data structure [Wei85, Ket98] we view each edge of a triangle mesh as a pair of oppositely oriented halfedges. Internally, the network generally operates on data (*features*) associated with halfedges. Through interface layers also per-vertex, per-edge, and per-face data can be handled in network input and output when required, without information loss or degradation on the input side.

### 2. Related Work

In recent years we could witness the proposal of quite a variety of approaches to make deep learning, in particular using CNNs, applicable to 2-manifold domains [BBL\*17], most relevantly in the form of surface triangle meshes [HL21]. These range from globally or locally reducing surface-based settings to 2D image settings, to defining novel operators and architectures (in particular for convolution and pooling) dedicated to the triangle mesh setting.

**Image Reduction** One approach consists of applying classical 2D image pixel grid based CNNs. To this end the surface (or rather the input data associated with it, such as coordinates, colors, descriptors) needs to be mapped to the plane, as a whole or in pieces. This can be done by means of rendering techniques in multi-view methods [SMKLM15, CMW\*17, SBZB15] or by means of mapping methods that unfold the surface to the plane [MGA\*17, ES-KBC17, HSBH\*19, SBR16]. Challenges lie in dealing with aspects such as visibility, topology, and distortion.

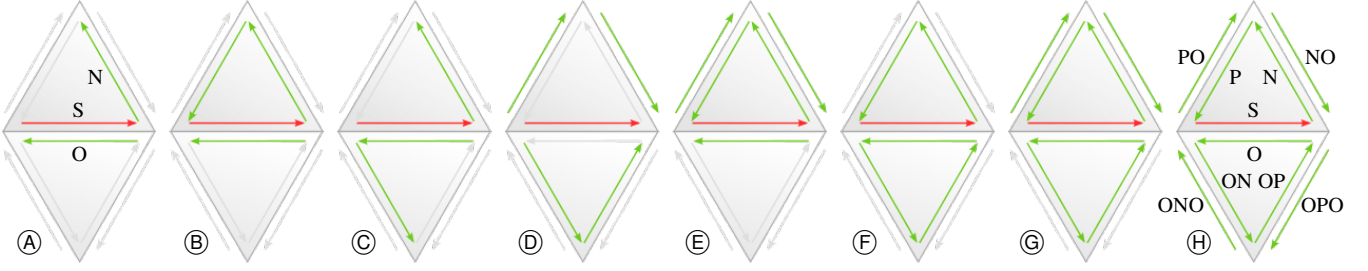
**Local Resampling** Another idea consists of, around each point of interest (typically each vertex), resampling the input data associated with the mesh onto a fixed (e.g. polar) grid structure, so as to establish constant neighborhoods that enable the application of a common convolution operator [MBBV15, BMRB16, KBLB12].

This can then be used to define convolutional layers for a neural network architecture. The need to resample, based on interpolating the input data that is often defined per vertex, can be seen as a downside, because a loss of information and fidelity is inevitable. An alternative are unstructured “continuous” convolutions, that may assume a fixed neighborhood size but do not assume a fixed neighborhood grid structure [YLB\*21, MKK21, WEH20].

**Graph-Based Approaches** The vertices and edges of a mesh form a graph. Hence, general graph neural networks using graph convolutions [SGT\*08, HBL15] can easily be applied. Per se, such an approach does not exploit the manifold surface nature of the mesh, though. A number of proposals have been made to inject geometric surface information in this context [KJP\*18, SACO22].

**Mesh-Based Approaches** A few recent methods focus on truly embracing the fact that the input is a mesh describing a 2-manifold surface, and on operating natively on the mesh-associated input data. The method of Feng et al. [FFY\*19] is face based, exploiting the constant neighborhood structure of three edge-adjacent faces per face. It therefore also assumes input and output signals of the network to be face based. The work of Hertz et al. [HHGCO20] likewise uses face based convolutions. Larger or dilated convolution neighborhoods in the face based setting can be used when restricting (e.g. via remeshing) to largely regular meshes (with subdivision connectivity) [HLG\*22]. The MeshCNN method [HHF\*19, BEB21] is edge based, exploiting the constant neighborhood structure of four directly adjacent edges per edge. Input and output signals are assumed to be edge based in this case. In both cases (face based and edge based) there are ordering ambiguities among the neighboring elements, requiring sorting [HHF\*19], maximum selection [HHGCO20], averaging, or other forms of “symmetrization”, with an associated loss of information. The method of Milano et al. [MLR\*20] can be viewed as a combination thereof, natively taking edge based and face based features as input. One configuration of their method, the dual graph with double nodes discussed in their supplementary material, though not framed in terms of halfedges, can effectively be viewed as a particular instance of our framework, with a particular convolution neighborhood choice (namely  $\textcircled{D}$  as of Sec. 3.1).

In the case of vertex based signals to be processed, the situation is more intricate because the vertex neighborhood structure commonly (and often inevitably) is variable across a mesh. This precludes the direct definition of a shared convolution operator. A convolution-like operator can still be formed by convolving only fixed-size subsequences of the one-ring neighborhood and pooling over all subsequences cyclically. Like in the above face or edge centered approaches, this pooling (e.g. by averaging) is necessary to deal with the ordering ambiguity. The vertex layer of Liu et al. [LKC\*20] roughly follows this approach (additionally inserting subsequence-specific input features). Let us remark that the oriented *half-flap* operator used in that work might easily be mistaken to be closely related or even equivalent to part of our *halfedge* centered approach. However, this operator (which specifically consumes vertex based latent features combined with halfedge based input features) serves as a logical sub-unit of vertex-to-edge and vertex-to-vertex convolution-like layers (using averaging over sub-



**Figure 2:** Illustration of the different neighborhoods of a central halfedge (red) considered in Sec. 3.1. On the far left the minimal neighborhood (S,O,N) is shown, consisting of only the central halfedge itself, its opposite, and its next halfedge. The neighborhood on the far right contains all halfedges belonging to the five edges of two edge-adjacent faces.

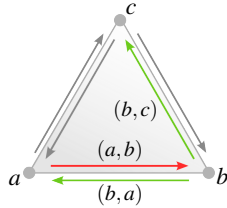
sequences, as discussed above). In particular that work does not involve any cascadable halfedge-to-halfedge layers (whether of convolution or pooling type) that consume and produce latent features living on halfedges.

**Non-Convolutional Approaches** Less related are approaches focusing on non-convolutional networks. This includes techniques that essentially bring mesh vertices into one-dimensional orders to enable the applications of recurrent neural networks (RNNs), using random walks [LT20] or spiral patterns [LDCK18, GCBZ19, BBP\*19]. Also point cloud based techniques [QSMG17] can easily be applied to the set of vertex points, albeit not exploiting the potentially useful mesh connectivity information.

### 3. Halfedge-Based Convolution

Assume we are given a closed manifold triangle mesh. The two halfedges corresponding to an edge  $\{a,b\}$  between two vertices  $a$  and  $b$  are the ordered tuples  $(a,b)$  and  $(b,a)$ ; they can be viewed as (oppositely) oriented edges. We say a halfedge  $(a,b)$  belongs (see Fig. 6) to the vertex  $a$ , to the edge  $\{a,b\}$ , and to the triangle  $(a,b,c)$ , where the latter is a cyclic list (i.e. equivalent to  $(b,c,a)$  and  $(c,a,b)$ ) ordered counterclockwise (by convention).

The adjacency among the set of all halfedges of a triangle mesh is fully represented by two pieces of information per halfedge  $(a,b)$ : the *opposite* halfedge (i.e.  $(b,a)$ ) and the *next* halfedge (i.e.  $(b,c)$  if there is a triangle  $(a,b,c)$ ).



#### 3.1. Neighborhoods

A minimal halfedge neighborhood that can therefore be used to define a reasonable convolution operator centered at a halfedge is the halfedge itself (S), its opposite halfedge (O), and its next halfedge (N). We abbreviate this neighborhood (S,O,N), illustrated in Fig. 2 left. It is minimal in the sense that repeated convolutions performed over this neighborhood allow information exchange between arbitrary halfedges in a mesh. Smaller neighborhoods, (S,O) or (S,N), would confine information exchange to within individual edges or triangles—similar to how, e.g.,  $1 \times 3$ -convolutions in pixel grids would limit exchange to within columns (or rows).

Similar to how differently sized convolution kernels are employed in CNNs on grid data depending on the specific use case, we may define also larger (non-minimal) ordered neighborhoods. This includes the following, as illustrated in Fig. 2, where we, e.g., use NO to denote the next→opposite halfedge of S, and abbreviate NN as P (because the *previous* halfedge in a triangle is the next halfedge of the next halfedge):

- (A) (S,O,N)
- (B) (S,O,N,P)
- (C) (S,O,N,ON)
- (D) (S,NO,PO,ON,OP)
- (E) (S,O,N,P,NO,PO)
- (F) (S,O,N,P,ON,OP)
- (G) (S,O,N,P,ON,OP,NO,PO)
- (H) (S,O,N,P,ON,OP,NO,PO,ONO,OPO)

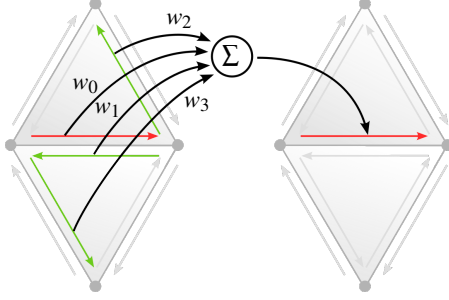
Even larger neighborhoods could be defined. However, they could easily be improper depending on the local mesh connectivity around the center halfedge S. We say a neighborhood is proper if it contains no halfedge of the mesh more than once. The above neighborhoods, due to being confined to the set of halfedges belonging to the edges of two edge-adjacent triangles, are always proper as long as the mesh contains no loop edges (of type  $(a,a)$ ) and no vertices of valence below 3. While impropriety is not an issue per se, it leads to potentially less useful convolution results of questionable comparability depending on the local tessellation; we therefore restrict our considerations to the above neighborhood options.

#### 3.2. Convolution

For any such choice  $\eta = (\eta_0, \dots, \eta_{n-1})$  of neighborhood, of size  $n$ , the halfedge based convolution operation (Fig. 3) is then simply defined at a halfedge  $h$  as

$$f_{\text{out}}(h) = \sum_{i=0}^{n-1} w_i \cdot f_{\text{in}}(\eta_i(h)), \quad (1)$$

where  $\eta_i(h)$  is the  $\eta_i$ -neighbor halfedge of  $h$ ,  $f_{\text{in}} \in \mathbb{R}^{n_c \times n_h}$  is the feature input (with  $n_c$  channels for  $n_h$  halfedges) to the convolution layer,  $f_{\text{out}} \in \mathbb{R}^{n'_c \times n_h}$  its output (with  $n'_c$  channels), and  $w_i \in \mathbb{R}^{n_c \times n'_c \times n_n}$  the learnable weights. As usual, a bias can be added and a nonlinear activation function can follow.



**Figure 3:** Illustration of the convolution operation, on the example of the  $(S,O,N,ON)$  neighborhood, centered at a halfedge (red).

Notice that there is no ambiguity regarding order or orientation among the neighborhood elements. We therefore do not need to impose symmetries onto the operation (as in MeshCNN) or apply local rotational pooling operations (as in SpiralNet) to achieve invariance to ambiguities.

In the case of meshes with boundary, where neighborhoods can be partial, the sum may skip the missing entries, akin to zero-padding commonly used in image convolutions.

#### 4. Halfedge-Based Pooling

Pooling, i.e. increasing the receptive field of subsequent layers without increasing their complexity (number of connections and weights), can be performed in a mesh-based setting by means of mesh decimation or element agglomeration. The simplest and most local operation to that end is the edge collapse, as employed for that purpose in MeshCNN.

##### 4.1. Pooling

The main degrees of freedom, besides the choice of edges to collapse, lie in the choice of function that combines the features associated with the edges that are removed and merged, respectively. Fig. 4 illustrates the setting and the indexing used in the following. Directly adapting the choice proposed for MeshCNN to our halfedge setting would correspond to the following, where the two halfedges belonging to an edge are treated indifferently (which we will refer to as *edge-pooling* in the following):

$$f_{\text{out}}(h_2/h_5) = \left( \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, 0, 0, 0, 0 \right) \left( f_{\text{in}}(h_0), \dots, f_{\text{in}}(h_9) \right)^{\text{T}}$$

$$f_{\text{out}}(h_6/h_9) = \left( \frac{1}{6}, \frac{1}{6}, 0, 0, 0, 0, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6} \right) \left( f_{\text{in}}(h_0), \dots, f_{\text{in}}(h_9) \right)^{\text{T}}$$

We can, however, choose the coefficients more flexibly. Conceptually, any combination of coefficients could be used, or functions beyond linear combinations (e.g. max pooling). Exploiting the opposite-orientation nature of halfedges, the following combination appears to be a sensible option:

$$f_{\text{out}}(h_2) = \left( \frac{1}{3}, 0, \frac{1}{3}, 0, \frac{1}{3}, 0, 0, 0, 0, 0 \right) \left( f_{\text{in}}(h_0), \dots, f_{\text{in}}(h_9) \right)^{\text{T}}$$

$$f_{\text{out}}(h_5) = \left( \frac{1}{3}, 0, 0, \frac{1}{3}, 0, \frac{1}{3}, 0, 0, 0, 0 \right) \left( f_{\text{in}}(h_0), \dots, f_{\text{in}}(h_9) \right)^{\text{T}}$$

$$f_{\text{out}}(h_6) = \left( 0, \frac{1}{3}, 0, 0, 0, 0, \frac{1}{3}, 0, \frac{1}{3}, 0 \right) \left( f_{\text{in}}(h_0), \dots, f_{\text{in}}(h_9) \right)^{\text{T}}$$

$$f_{\text{out}}(h_9) = \left( 0, \frac{1}{3}, 0, 0, 0, 0, 0, \frac{1}{3}, 0, \frac{1}{3} \right) \left( f_{\text{in}}(h_0), \dots, f_{\text{in}}(h_9) \right)^{\text{T}}$$

We use this variant (called *halfedge-pooling*) as a default in our halfedge-based pooling layer.

The consideration of the fixed pooling coefficients as learnable parameters may be a worthwhile aspect for future work.

##### 4.2. Unpooling

For unpooling, executed as an un-collapse operation on the mesh, we can analogously distinguish between oppositely oriented halfedges and, with indexing again following Fig. 4, set

$$f_{\text{out}}(h_2) = f_{\text{out}}(h_4) = f_{\text{in}}(h_2)$$

$$f_{\text{out}}(h_3) = f_{\text{out}}(h_5) = f_{\text{in}}(h_5)$$

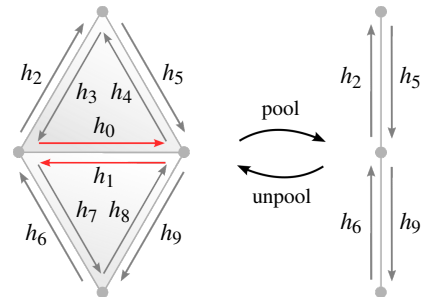
$$f_{\text{out}}(h_6) = f_{\text{out}}(h_8) = f_{\text{in}}(h_6)$$

$$f_{\text{out}}(h_7) = f_{\text{out}}(h_9) = f_{\text{in}}(h_9)$$

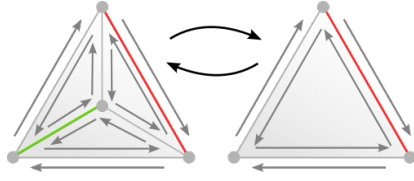
$$f_{\text{out}}(h_0) = \frac{1}{2}f_{\text{in}}(h_2) + \frac{1}{2}f_{\text{in}}(h_5)$$

$$f_{\text{out}}(h_1) = \frac{1}{2}f_{\text{in}}(h_6) + \frac{1}{2}f_{\text{in}}(h_9)$$

**Mesh Regularity** Uncontrolled edge collapses can cause a mesh to become *topologically irregular*, for instance containing two edges between one pair of vertices, or even loop edges. This not only increases demands on the generality of the mesh data structure, it also affects the propriety of convolution neighborhoods (cf. Sec. 3.1). This can be avoided by omitting collapses of edges that violate the *link condition* [DEGN98], which, however, may preclude the desired pooling order. More flexibility can be achieved using a special handling of vertices of valence 3 next to a to-be-collapsed edge, as these would cause an irregularity if the edge is collapsed. By first removing such a vertex with a 3:1 triangle collapse, as illustrated in Fig. 5, the edge collapse can be enabled. A similar special treatment, while not described in the respective paper, can also be found in the prototype implementation of MeshCNN.



**Figure 4:** Illustration of the pooling (and unpooling) operation centered at a pair of halfedges (red). The indexing is used in Sec. 4 to define the accompanying feature update formulas.



**Figure 5:** Special pre-handling of a valence 3 vertex (center) next to a to-be-collapsed edge (red): The opposite edge (green) is collapsed first, effectively performing a 3:1 triangle collapse. Afterwards the red edge is no longer in violation of the link condition due to the valence 3 vertex.

## 5. Vertex/Edge/Face Interface Layers

The above described convolution and pooling layers allow for the composition of deep networks that have per-halfedge values as input and output. Depending on the use case, it may be desirable or more natural to deal with values associated with other mesh entities.

Notice that the belonging-relationships (as defined in Sec. 3) between halfedges and vertices, edges, faces are 1:k, 1:2, and 1:3 relationships, respectively; see Fig. 6. This allows us to define x-to-halfedge interface layers that can be prepended to a network. For example, a vertex-to-halfedge layer takes as input a value (a feature vector) per vertex and assigns it to all uniquely associated halfedges in its output:

$$f_{\text{out}}(h) = f_{\text{in}}(\text{from}(h)), \quad (2)$$

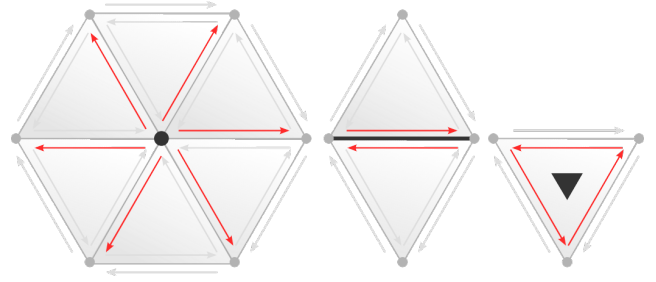
where  $\text{from}(h)$  yields the vertex the halfedge  $h$  belongs to. Analogously, one defines edge-to-halfedge and face-to-halfedge layers.

Regarding network output, an averaging operation over the belonging halfedges can be used to define halfedge-to-x layers. For the example of a halfedge-to-vertex layer:

$$f_{\text{out}}(a) = \frac{1}{\text{valence}(a)} \sum_{(a,b)} f_{\text{in}}((a,b)). \quad (3)$$

Analogously, one defines halfedge-to-edge and halfedge-to-face layers (where the averaging is over a fixed number of 2 or 3 halfedges, respectively). These layers should not be misunderstood as post-processing operations; they can be used as part of a network architecture that is learned end-to-end. We note that related operations are sometimes used in other mesh-based learning approaches, e.g. edge-to-vertex [HMGCO20] or face-to-vertex [HHGCO20] averaging.

Note that the halfedge-to-edge output layer effectively performs a symmetrization, in the sense that potential feature differences between the two halfedges of an edge are lost – which is inevitable if edge-based network output is asked for. This does not mean, however, that the use of halfedges in the preceding network layers is pointless; the input features as well as the latent features are free to be non-symmetric, with positive effects as observed in the experiment in Sec. 7.2. The data in Table 1 in Sec. 7.3 provides further insight into the non-symmetry of the halfedge based features in latent layers.



**Figure 6:** Illustration of the halfedges (red) belonging to a vertex, an edge, or a face (marked in black), in 1:k, 1:2, and 1:3 relationships, respectively.

## 6. Input Features

The surface signals that are relevant as input data to the network of course strongly depend on the application scenario. For geometric learning tasks naturally some kind of geometric information should be (part of) the input.

In particular, we can easily use the same edge-based input used by Hanocka et al. [HHF\*19], namely, for each edge, its dihedral angle, the two opposite inner angles, and the height-base-ratios of the two adjacent triangles, or, as discussed by Barda et al. [BEB21], the edge’s dihedral angle and its (normalized) length. This is possible simply via an edge-to-halfedge input layer, i.e. effectively assigning the same value to both halfedges belonging to an edge.

Importantly, in the halfedge based setting we can easily also take oriented information into account. Instead of dealing with the ordering ambiguity of an edge’s two adjacent faces (e.g. by averaging or sorting the feature values associated with the two faces by value [HHF\*19]), the halfedge orientation can be exploited, avoiding such ambiguities. For instance, we can use as input features *per halfedge*: the dihedral angle of its edge, the opposite inner angle in the unique triangle that the halfedge belongs to, and the base-height ratio of the unique triangle that the halfedge belongs to. The benefit of this avoided need for symmetrization is evaluated in Sec. 7.

Note that also data that natively is vertex or face based can be taken as input via an interface layer (Sec. 5), e.g. the discrete Gaussian curvature at a vertex or the roundness of triangle, and, if desired, extrinsic information like vertex coordinates or face normal vectors.

## 7. Evaluation

Given the genericity of the presented framework, it can be used for a variety of geometric learning tasks, with different settings and in conjunction with various pooling rules, attention mechanisms, etc. [BEB21, MLR\*20]. Note that it is not our intention to claim that the described approach is in any way strictly better for a specific task than all other potential alternative approaches. Rather, we are interested in evaluating the benefits and potential negative effects of using the *halfedge* centered setting as compared to the *edge* centered setting. To this end we consider the learning problems also used in the evaluation of the edge centered MeshCNN [HHF\*19, Sec. 5] for comparison in the following.

We adopt the network architectures used in the MeshCNN evaluation as directly as possible for HalfedgeCNN, so as to enable fair and insightful comparisons and ablations. Concretely, we only

- exchange the original edge based convolution layers with our halfedge based convolution layers,
- depending on the experiment, exchange the original (un)pooling rules with our halfedge-pooling or edge-pooling,
- depending on the experiment, use halfedge based input features or, via a prepended edge-to-halfedge layer, the original edge based input features.

Apart from that, all other architecture details including hyperparameters, pooling hierarchy sizes, collapse selection by feature magnitude, and training data augmentation strategy are kept the same.

Regarding the convolution, we focus our experiments on a selection of the convolution neighborhoods discussed in Sec. 3.1, as listed in the table below. The (increasing) number of halfedges (besides the central halfedge) involved in these neighborhoods is listed in the table as well. We will refer to the neighborhoods by this neighborhood size in the plots (on the horizontal axes) in the following.

Neighborhood Name:	Ⓐ	Ⓒ	Ⓓ	Ⓕ	Ⓖ	Ⓗ
Neighborhood Size:	2	3	4	5	7	9

### 7.1. Classification

We first consider a shape classification task on the basis of the SHREC dataset [LGB\*11], cf. Fig. 7, using a split of each class into 16 training and 4 test examples.

Because training results deviate slightly from execution to execution of the training process (due to random initialization of the learnable weights), we generally train each network (HalfedgeCNN as well as MeshCNN) for each setting in each experiment 30 times, and report test accuracies averaged over these 30 runs in the following. Note that, to ensure comparability, we also rerun the MeshCNN experiments in this manner in the same environment on the same hardware, instead of simply reporting the values from the original paper—for instance, for the classification task we obtain an accuracy of around 99.0%, while the original paper reports 98.6%.

#### Oriented Input Features

The benefit of not having to symmetrize input features can be observed in Fig. 8. In this plot, HalfedgeCNN was used with various convolution neighborhood settings (horizontal axis). Edge-pooling (see Sec. 4.1) was used, mimicking the edge-based pooling of MeshCNN. The use of oriented, i.e. halfedge-based, input

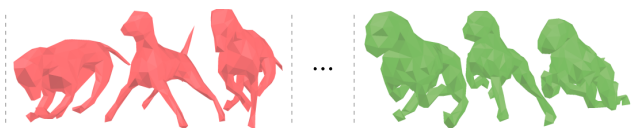


Figure 7: Examples of models (3 of 20) from two of the 30 classes of the classification dataset.

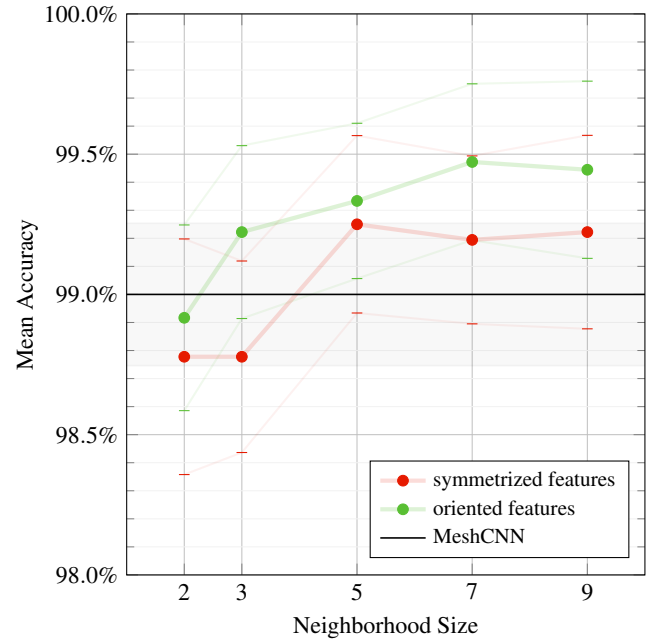


Figure 8: Classification task. Comparison of using the symmetrized edge-based input features of MeshCNN (5 values per edge—opposite inner angles, height-base ratios, dihedral angle—sorted to make them orientation independent), assigned to both halfedges of an edge, versus oriented halfedge-based input features (the same 5 values, not pairwise sorted by magnitude but ordered based on the halfedge’s orientation). In addition, the mean accuracy of MeshCNN is shown as a baseline (black) for comparison, and the associated variance (over the 30 runs) is indicated by a one standard deviation thick corridor (light gray). For the others the analogous corridor is delineated by the thin ‘graphs’ above and below the bold mean ‘graph’.

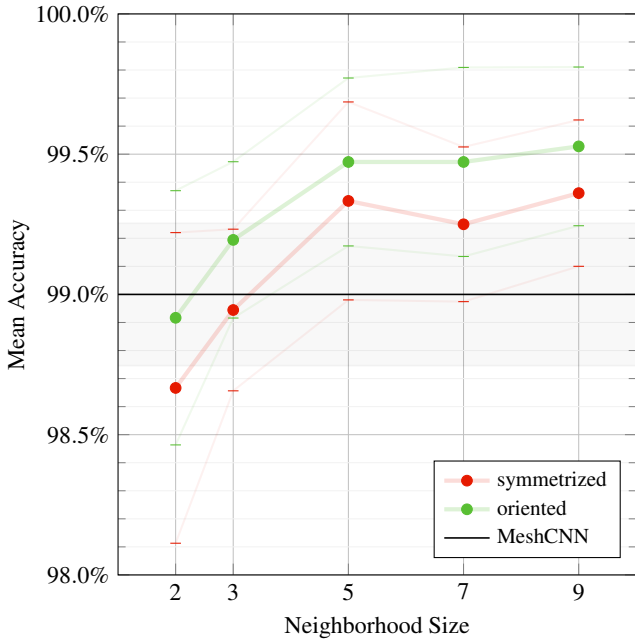
features (green) turns out to be consistently beneficial over symmetrized features (red), regardless of the chosen convolution neighborhood size, though the magnitude of the benefit varies.

It can also be seen that the mean classification accuracy of HalfedgeCNN with non-symmetrized features, exploiting one aspect enabled by its halfedge nature, is higher than that of MeshCNN for all but the smallest neighborhood (Ⓐ). Even for this minimal neighborhood it is very close though, while at the same time the number of learnable parameters is reduced by almost 40%, from 1323K of MeshCNN to 806K.

Interestingly, also with the symmetrized input features HalfedgeCNN shows some benefit, though to a lesser extent and with somewhat higher variance, and only for the larger neighborhood sizes.

#### Halfedge-Pooling

The effect of switching to halfedge-based pooling can be observed in Fig. 9. Notice that this brings the performance of the symmetrized input features even for a small neighborhood like Ⓒ



**Figure 9:** Classification task. Same setting as in Fig. 8, but using halfedge-based pooling in HalfedgeCNN, causing some small additional benefit for most settings.

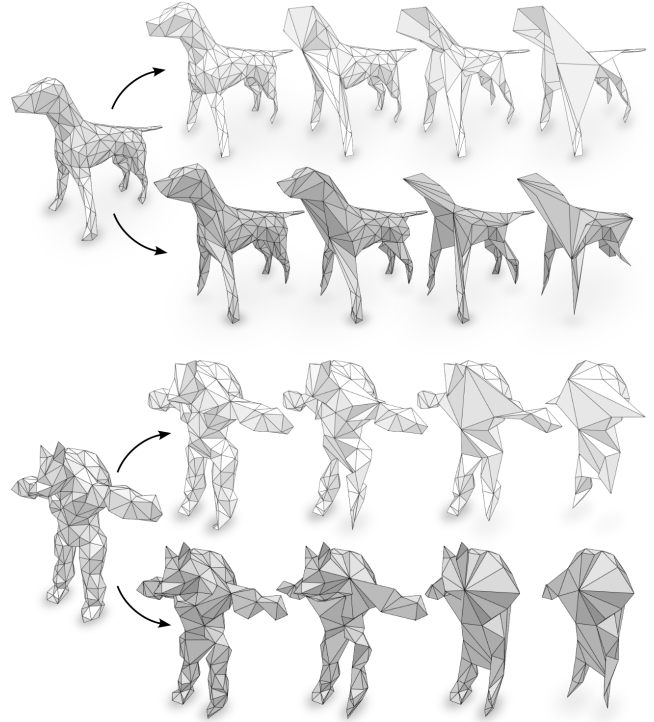
(size 3) very close to MeshCNN. In combination with oriented input features, enabled by our halfedge setting, we again observe a further advantage. In particular, notice that the error (the accuracy gap to 100%) of MeshCNN is more than halved when using the oriented halfedge features with halfedge-pooling and the largest neighborhood  $\mathbb{H}$  (which is the equivalent of the MeshCNN neighborhood).

In Fig. 10 it can be seen that HalfedgeCNN with halfedge-pooling can indeed cause pooling behavior significantly different from MeshCNN; its implications, beyond the reported quantitative differences, are hard to interpret visually, though.

We repeat the classification experiment for different training/test splits of the SHREC dataset (10/10 and 4/16 in addition to the above 16/4) and observe similar behavior: for very small neighborhood sizes the classification accuracy is consistently similar or lower than that of the MeshCNN baseline, for the larger neighborhoods it is consistently higher, across these various splits.

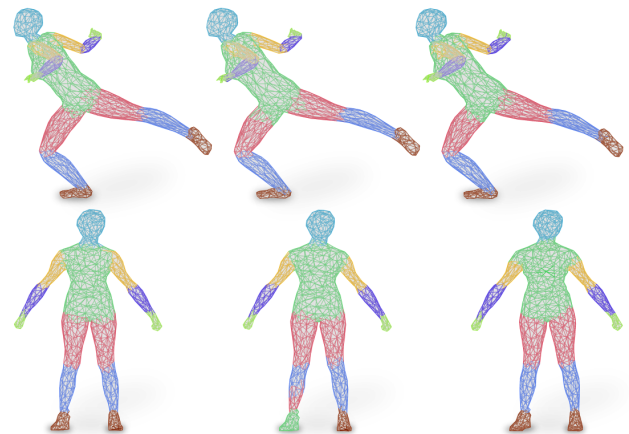
## 7.2. Segmentation

We now consider a segmentation task on the basis of the Human Body Segmentation dataset [MGA\*17], cf. Fig. 11. We again use the same setup as in the evaluation of MeshCNN in [HHF\*19], except that for both, HalfedgeCNN and MeshCNN, we reduce the number of epochs to 300 as we did not observe benefits beyond that. Note that in contrast to the classification task, here also unpooling layers (Sec. 4.2) come into play in the U-Net architecture.

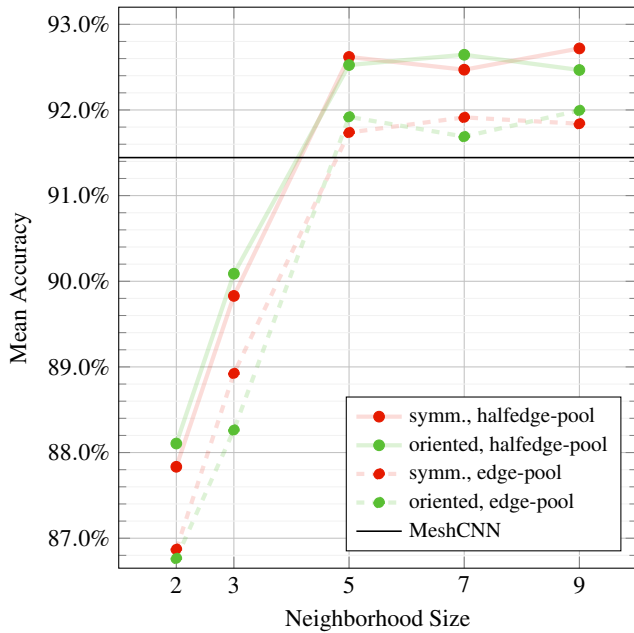


**Figure 10:** Examples of the pooling hierarchy observed for two test models of the classification task with HalfedgeCNN (lighter) and MeshCNN (darker).

Just like in the above classification task, we always train the networks 30 times and report the average accuracy.



**Figure 11:** Examples of the body part segmentation task, for illustration. Left: ground truth segmentation. Center: prediction using trained MeshCNN. Right: prediction using trained HalfedgeCNN. Differences are often in the details; the lower one is an example with clearly visible prediction deviations from ground truth.



**Figure 12:** Segmentation task. Accuracy for different neighborhoods, with edge-pooling and halfedge-pooling, in comparison to MeshCNN.

### Oriented Input Features

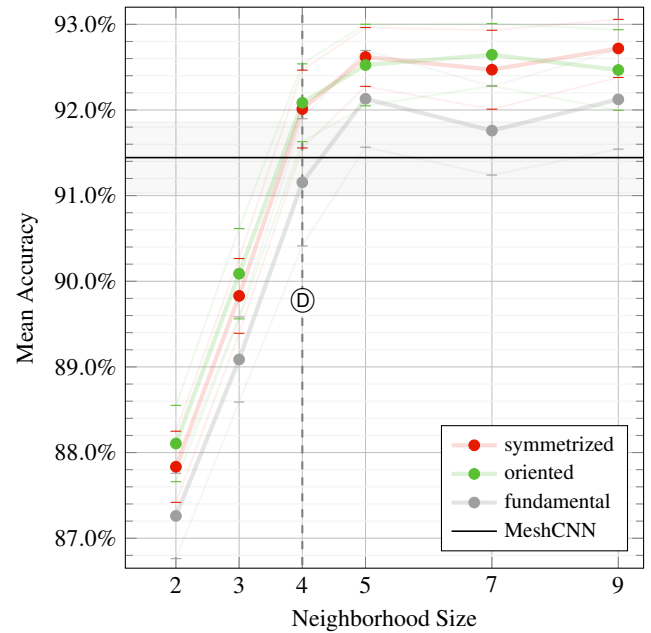
In Fig. 12 we can see that in this learning task, in contrast to the classification task in Sec. 7.1, the choice of either symmetrized or oriented input features does not make a consistent difference. It can also be observed that the flexibility in terms of convolution neighborhoods, that HalfedgeCNN offers, can be exploited beneficially: While rather small neighborhood sizes proved adequate for the classification task (in terms of accuracy, and with potential benefits in terms of training and inference time, due to a lower number of operations needed), for the segmentation task larger neighborhoods turn out to be strongly advisable—notwithstanding that different architectures or hyperparameters may of course lead to beneficial behavior of smaller convolution neighborhoods as well.

### Halfedge-Pooling

Switching to halfedge-pooling leads to a consistent increase in accuracy relative to edge-pooling, over both input feature modes and all neighborhood sizes, as can be seen in Fig. 12, comparing dashed (edge-pooling) and full (halfedge-pooling) lines. The benefit over MeshCNN is increased by a factor of about 2–4 for the larger convolution neighborhoods. Interestingly, the benefit of halfedge-pooling over edge-pooling is much more pronounced and consistent in this segmentation task than in the above classification task. This may be related to the fact that the segmentation network makes use of pooling as well as unpooling layers in its U-Net architecture.

### 7.3. Further Observations

As mentioned in Sec. 2, one variant of the framework proposed by Milano et al. [MLR<sup>+</sup>20] can be viewed as performing halfedge-



**Figure 13:** Segmentation task. Same setting as in Fig. 12, additionally including neighborhood  $\textcircled{D}$  of size 4. Furthermore, the results of instead using the fundamental form input features for HalfedgeCNN are shown. Variance corridors are indicated like in Fig. 8.

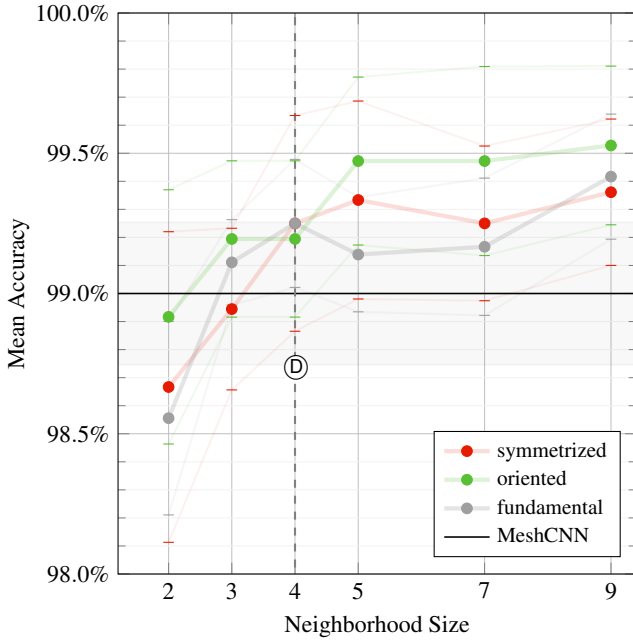
based convolutions, fixed to neighborhood  $\textcircled{D}$ . We performed experiments also with this neighborhood and did not find it to perform particularly well. With the oriented input features, there is even a peculiar drop in accuracy compared to the next smaller as well as the next larger neighborhood (see Fig. 14). For the segmentation task this neighborhood shows more reasonable behavior (Fig. 13), but in particular for the proposed fundamental form features, neighborhood  $\textcircled{E}$  of size 5 performs better.

Another interesting observation is that on the segmentation task, the use of the fundamental form input features (instead of the default) leads to lower accuracy than MeshCNN when using edge-pooling (e.g.  $-0.35\%$  for  $\textcircled{E}$ ,  $-0.61\%$  for  $\textcircled{G}$ ,  $-0.50\%$  for  $\textcircled{H}$ ). In combination with the use of our halfedge-pooling, by contrast, a higher accuracy is achieved ( $+0.69\%$  for  $\textcircled{E}$ ,  $+0.32\%$  for  $\textcircled{G}$ ,  $+0.68\%$  for  $\textcircled{H}$ ).

Training time per epoch (on a system with Core i9-12900K and GeForce RTX 3080) with our HalfedgeCNN implementation that very closely follows the implementation of MeshCNN published by the authors, even when using the largest convolution neighborhood  $\textcircled{H}$ , interestingly is higher by a factor of 1.48 only (15.6s vs 23.1s) on the classification task and 1.27 only (38.3s vs 48.7s) on the segmentation task—despite the fact that the number of entities (halfedges instead of edges) as well as the convolution size (10 instead of 5) are doubled.

Attempting to get an idea to what extent the additional degrees of freedom of the halfedge based setting are exploited by the network, Table 1 reports the average non-symmetry of features within pairs





**Figure 14:** Classification task. Same setting as in Fig. 9, additionally including neighborhood  $\textcircled{D}$  of size 4. Furthermore, the results of instead using the fundamental form input features for HalfedgeCNN is shown.

of halfedges. Indeed, there are clear deviations. Even when feeding symmetric features into the network, the features of latent layers diverge within halfedge pairs, indicating that some use is made of the degrees of freedom—which was to be expected based on the improvements observed in the above experiments.

## 8. Conclusion

We have described HalfedgeCNN, a collection of modules to build neural networks that operate on triangle meshes. The key characteristic is the fact that all operations are centered on halfedges. From a conceptual point of view this can provide benefits over operations based on other mesh entities, such as higher flexibility and avoidance of orientation ambiguities. Our experiments indicate that, depending on the application scenario, these conceptual benefits can materialize as concrete advantages.

We therefore believe that this proposal is a valuable addition to the range of available options. The further exploration of application-specific practical benefits, as well as the task of generally bringing some order into the growing zoo of alternative approaches for geometric deep learning on surfaces, provide interesting and worthwhile avenues for future work.

## Acknowledgements

The authors wish to thank Steffen Hinderink for helpful discussions and the authors of MeshCNN for open-sourcing their implementation and data. This work was supported by the Deutsche Forschungsgemeinschaft (DFG) - 456666331.

Layer:	input	pool-1	pool-2	pool-3	pool-4
<b>Classification</b>					
oriented	7.17%	13.17%	13.33%	13.39%	13.32%
symmetrized	0.00%	14.75%	14.85%	14.87%	15.09%
<b>Segmentation</b>					
oriented	7.25%	13.06%	12.85%	12.83%	–
symmetrized	0.00%	14.50%	14.43%	14.38%	–

**Table 1:** Relative deviation between the feature values of the two halfedges of an edge, averaged over all edges and all features during all test set inferences of the classification task and the segmentation task, with neighborhood  $\textcircled{F}$  and halfedge-pooling. Shown is the relative deviation in the input and after each of the four (three) convolution+pooling double-layers of the network, for symmetrized as well as for oriented input features. Notice that even if the input features are symmetric, i.e. equal for paired halfedges, the features of halfedges develop individually in the further layers.

## References

- [BBL\*17] BRONSTEIN M. M., BRUNA J., LECUN Y., SZLAM A., VANDERGHEYNST P.: Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine* 34, 4 (2017), 18–42. 2
- [BBP\*19] BOURITSAS G., BOKHNYAK S., PLOUMPIS S., BRONSTEIN M., ZAFEIRIOU S.: Neural 3D morphable models: Spiral convolutional networks for 3D shape representation learning and generation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)* (2019). 2, 3
- [BEB21] BARDA A., EREL Y., BERMANO A. H.: MeshCNN fundamentals: Geometric learning through a reconstructable representation, 2021. URL: <https://arxiv.org/abs/2105.13277>. 2, 5
- [BMRB16] BOSCAINI D., MASCI J., RODOLÀ E., BRONSTEIN M.: Learning shape correspondence with anisotropic convolutional neural networks. *Advances in neural information processing systems* 29 (2016). 1, 2
- [CMW\*17] CHEN X., MA H., WAN J., LI B., XIA T.: Multi-view 3D object detection network for autonomous driving. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition* (2017), pp. 1907–1915. 2
- [DEGN98] DEY T. K., EDELSBRUNNER H., GUHA S., NEKHAYEV D. V.: Topology preserving edge contraction. In *Publ. Inst. Math. (Beograd)* (1998). 4
- [ESKBC17] EZUZ D., SOLOMON J., KIM V. G., BEN-CHEN M.: GWCNN: A metric alignment layer for deep shape analysis. *Computer Graphics Forum* 36, 5 (2017), 49–57. 2
- [FFY\*19] FENG Y., FENG Y., YOU H., ZHAO X., GAO Y.: Meshnet: Mesh neural network for 3D shape representation. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2019), vol. 33.1, pp. 8279–8286. 2
- [FM82] FUKUSHIMA K., MIYAKE S.: Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position. *Pattern recognition* 15, 6 (1982), 455–469. 1
- [GCBZ19] GONG S., CHEN L., BRONSTEIN M., ZAFEIRIOU S.: Spiralnet++: A fast and highly efficient mesh convolution operator. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops* (2019). 2, 3
- [HBL15] HENAFF M., BRUNA J., LECUN Y.: Deep convolutional networks on graph-structured data, 2015. URL: <https://arxiv.org/abs/1506.05163>. 2

- [HHF\*19] HANOCKA R., HERTZ A., FISH N., GIRYES R., FLEISHMAN S., COHEN-OR D.: MeshCNN: a network with an edge. *ACM Transactions on Graphics (TOG)* 38, 4 (2019). 2, 5, 7
- [HHGCO20] HERTZ A., HANOCKA R., GIRYES R., COHEN-OR D.: Deep geometric texture synthesis. *ACM Transactions on Graphics (TOG)* 39, 4 (2020). 2, 5
- [HL21] HANOCKA R., LIU H.-T. D.: An introduction to deep learning on meshes. In *SIGGRAPH 2021 Courses*. ACM, 2021. 2
- [HLG\*22] HU S.-M., LIU Z.-N., GUO M.-H., CAI J.-X., HUANG J., MU T.-J., MARTIN R. R.: Subdivision-based mesh convolution networks. *ACM Transactions on Graphics (TOG)* 41, 3 (2022). 2
- [HMGCO20] HANOCKA R., METZER G., GIRYES R., COHEN-OR D.: Point2mesh: A self-prior for deformable meshes. *ACM Trans. Graph.* 39, 4 (2020). 5
- [HSBH\*19] HAIM N., SEGOL N., BEN-HAMU H., MARON H., LIPMAN Y.: Surface networks via general covers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (2019), pp. 632–641. 2
- [KBLB12] KOKKINOS I., BRONSTEIN M. M., LITMAN R., BRONSTEIN A. M.: Intrinsic shape context descriptors for deformable shapes. In *2012 IEEE Conference on Computer Vision and Pattern Recognition* (2012), IEEE, pp. 159–166. 2
- [Ket98] KETTNER L.: Designing a data structure for polyhedral surfaces. In *Proceedings of the fourteenth annual symposium on Computational geometry* (1998), pp. 146–154. 2
- [KJP\*18] KOSTRIKOV I., JIANG Z., PANOZZO D., ZORIN D., BRUNA J.: Surface networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2018), pp. 2540–2548. 2
- [LBBH98] LECUN Y., BOTTOU L., BENGIO Y., HAFFNER P.: Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86, 11 (1998), 2278–2324. 1
- [LDCK18] LIM I., DIELEN A., CAMPEN M., KOBBELT L.: A simple approach to intrinsic correspondence learning on unstructured 3D meshes. In *The European Conference on Computer Vision (ECCV) Workshops* (2018). 2, 3
- [LGB\*11] LIAN Z., GODIL A., BUSTOS B., DAUDI M., HERMANS J., KAWAMURA S., KURITA Y., LAVOUA G., SUETENS P. D., ET AL.: Shape retrieval on non-rigid 3D watertight meshes. In *Eurographics workshop on 3D object retrieval (3DOR)* (2011). 6
- [LKC\*20] LIU H.-T. D., KIM V. G., CHAUDHURI S., AIGERMAN N., JACOBSON A.: Neural subdivision. *ACM Transactions on Graphics (TOG)* 39, 4 (2020). 2
- [LT20] LAHAV A., TAL A.: Meshwalker: Deep mesh understanding by random walks. *ACM Transactions on Graphics (TOG)* 39, 6 (2020). 3
- [MBBV15] MASCI J., BOSCAINI D., BRONSTEIN M., VANDERGHEYNST P.: Geodesic convolutional neural networks on riemannian manifolds. In *Proceedings of the IEEE international conference on computer vision workshops* (2015), pp. 37–45. 1, 2
- [MGA\*17] MARON H., GALUN M., AIGERMAN N., TROPE M., DYM N., YUMER E., KIM V. G., LIPMAN Y.: Convolutional neural networks on surfaces via seamless toric covers. *ACM Transactions on Graphics (TOG)* 36, 4 (2017). 2, 7
- [MKK21] MITCHEL T. W., KIM V. G., KAZHDAN M.: Field convolutions for surface CNNs. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (2021). 2
- [MLR\*20] MILANO F., LOQUERCIO A., ROSINOL A., SCARAMUZZA D., CARLONE L.: Primal-dual mesh convolutional neural networks. *Advances in Neural Information Processing Systems* 33 (2020), 952–963. 2, 5, 8
- [QSMG17] QI C. R., SU H., MO K., GUIBAS L. J.: Pointnet: Deep learning on point sets for 3D classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2017), pp. 652–660. 3
- [SACO22] SHARP N., ATTAIKI S., CRANE K., OVSIANIKOV M.: Diffusionnet: Discretization agnostic learning on surfaces. *ACM Transactions on Graphics (TOG)* 41, 3 (2022). 2
- [SBR16] SINHA A., BAI J., RAMANI K.: Deep learning 3D shape surfaces using geometry images. In *European conference on computer vision* (2016), Springer, pp. 223–240. 2
- [SBZB15] SHI B., BAI S., ZHOU Z., BAI X.: DeepPano: Deep panoramic representation for 3-D shape recognition. *IEEE Signal Processing Letters* 22, 12 (2015), 2339–2343. 2
- [SGT\*08] SCARSELLI F., GORI M., TSOI A. C., HAGENBUCHNER M., MONFARDINI G.: The graph neural network model. *IEEE transactions on neural networks* 20, 1 (2008), 61–80. 2
- [SMKLM15] SU H., MAJI S., KALOGERAKIS E., LEARNED-MILLER E.: Multi-view convolutional neural networks for 3D shape recognition. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)* (2015). 2
- [WEH02] WIERSMA R., EISEMANN E., HILDEBRANDT K.: CNNs on surfaces using rotation-equivariant features. *ACM Transactions on Graphics (ToG)* 39, 4 (2020). 2
- [Wei85] WEILER K.: Edge-based data structures for solid modeling in curved-surface environments. *IEEE Computer Graphics and Applications* 5, 1 (1985), 21–40. 2
- [YLB\*21] YANG Z., LITANY O., BIRDAL T., SRIDHAR S., GUIBAS L.: Continuous geodesic convolutions for learning on 3D shapes. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision* (2021), pp. 134–144. 2