CNNs on Surfaces using Rotation-Equivariant Features

RUBEN WIERSMA, ELMAR EISEMANN, and KLAUS HILDEBRANDT, Delft University of Technology

This paper is concerned with a fundamental problem in geometric deep learning that arises in the construction of convolutional neural networks on surfaces. Due to curvature, the transport of filter kernels on surfaces results in a rotational ambiguity, which prevents a uniform alignment of these kernels on the surface. We propose a network architecture for surfaces that consists of vector-valued, rotation-equivariant features. The equivariance property makes it possible to locally align features, which were computed in arbitrary coordinate systems, when aggregating features in a convolution layer. The resulting network is agnostic to the choices of coordinate systems for the tangent spaces on the surface. We implement our approach for triangle meshes. Based on circular harmonic functions, we introduce convolution filters for meshes that are rotation-equivariant at the discrete level. We evaluate the resulting networks on shape correspondence and shape classifications tasks and compare their performance to other approaches.

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

Additional Key Words and Phrases: Geometric Deep Learning, CNNs on Surfaces, Surface Networks, Rotation-Equivariance, Circular Harmonic Filters, Shape Classification, Shape Segmentation, Shape Correspondence

ACM Reference Format:
Ruben Wiersma, Elmar Eisemann, and Klaus Hildebrandt. 2020. CNNs on Surfaces using Rotation-Equivariant Features. *ACM Trans. Graph.* 39, 4, Article 92 (July 2020), 12 pages. https://doi.org/10.1145/3386569.3392437

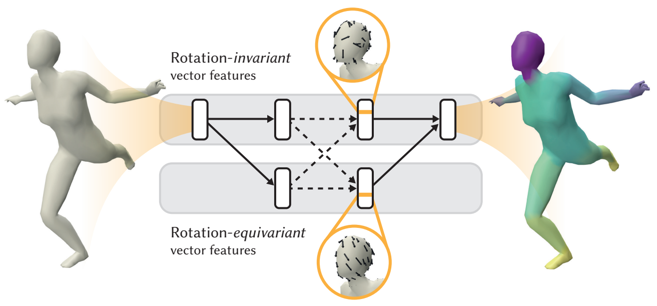

Fig. 1. We propose CNNs on surfaces that operate on vectors and separate rotation-equivariant and rotation-invariant features.

1 INTRODUCTION

The success of Deep Learning approaches based on convolutional neural networks (CNNs) in computer vision and image processing has motivated the development of analogous approaches for the analysis, processing, and synthesis of surfaces. Along these lines, approaches have been proposed for problems such as shape recognition [Su et al. 2015], shape matching [Boscaini et al. 2016], shape segmentation [Maron et al. 2017], shape completion [Litany et al. 2018], curvature estimation [Guerrero et al. 2018], and 3D-face synthesis [Ranjan et al. 2018].

In contrast to images, which are described by regular grids in a Euclidean domain, surfaces are curved manifolds and the grids on these surfaces are irregular. In order to still use regular grids, one can work with multiple projections of the surface on planes [Su et al. 2015] or with volumetric grids [Wu et al. 2015].

An alternative to learning on regular grids is generalized deep learning, often referred to as geometric deep learning [Bronstein et al. 2017], which targets irregularly sampled manifolds and graphs. A central element of such geometric CNNs is a generalized convolution operator. For CNNs on images, the convolution layers are built from convolution kernels, which are transported across the image. As a result, the parameters that define one kernel describe the convolution across the whole image, which significantly reduces the number of parameters that need to be learned. This is a motivation for exploring constructions of generalized convolution operators on surfaces based on convolution kernels.

Authors' address: Ruben Wiersma, r.t.wiersma@tudelft.nl; Elmar Eisemann, e.eisemann@tudelft.nl; Klaus Hildebrandt, k.a.hildebrandt@tudelft.nl, Delft University of Technology.

To apply a convolution kernel defined on $\mathbb{R}^2$ to a function at a point on a surface, the Riemannian exponential map is used to locally lift the function from the surface to a function defined on the tangent plane at the point. By identifying the tangent plane with $\mathbb{R}^2$, the convolution of the kernel and the lifted function can be computed. In this way, the convolution kernel can be applied everywhere on the surface. However, a problem arises, since there is a rotational degree of freedom when $\mathbb{R}^2$ is identified with a tangent plane. Moreover, the transport of filters on a surface depends on the chosen path. If a filter is transported along two different ways from one point of a surface to another, the transported filters are rotated against each other. This *rotation ambiguity* problem is fundamental and caused by the curvature of the surface.

The rotation ambiguity problem can be addressed by specifying a coordinate system at each point of the surface, *e.g.* according to the principal curvature directions [Boscaini et al. 2016; Monti et al. 2017; Pan et al. 2018] or the direction of maximum activation [Masci et al. 2015; Sun et al. 2018]. As a consequence, however, the coordinate systems in the local neighborhoods are arranged in different patterns for each point. For a network this means that, when features are aggregated to form the next layer of the network, the features are not only dependent on the sequence of convolution kernels that are applied, but also on the arrangement of coordinate systems in the local neighborhoods. Loosely speaking, the information contained in the features in the neighborhood of a point can be arbitrarily rotated against the coordinate system at the point. One can think of this as in a cubist painting, where the elements that make up a structure, for example the eyes, nose, and mouth on a face, are rotated against each other.

Multi-Directional Geodesic CNNs (MDGCNN) [Poulenard and Ovsjanikov 2018] provide an alternative approach to the rotation ambiguity problem. The idea is to sample the rotational degree of freedom regularly, to compute the convolutions for each of the sample directions, and to feed the results for all directions into the network. The multi-directional features are pooled at the final layer. The disadvantage of this approach is that each filter must not only be evaluated once at each point, but also for rotated versions of the filter and the results need to be stored. To build an operable network, the filters are only evaluated in some sample directions and the results are linearly interpolated to get results in intermediate directions, introducing inaccuracies in each consecutive layer.

We introduce a novel network architecture that does not suffer from the rotation ambiguity problem. The features in this network are rotation-equivariant and organized in streams of different equivariance classes (Figure 1). These streams interact with each other in the network and are finally merged into a rotation-invariant output. The resulting network is independent of the choice of coordinate systems in the tangent spaces, which means that it does not suffer from the rotation ambiguity problem. To realize this network, we work with vector-valued, instead of scalar-valued, features and introduce rotation-equivariant convolution and pooling operators on meshes. The convolution operators use the Riemannian exponential map and parallel transport to convolve vector-valued kernels on $\mathbb{R}^2$ with tangent vector fields on meshes.

As kernels on $\mathbb{R}^2$ we use the circular harmonics, which are known to be rotation-equivariant. We prove that the resulting discrete convolution operators on meshes are equivariant with respect to rotations of the coordinate system in the tangent spaces. Due to the rotation-equivariance property, it suffices to compute the convolution at each point with respect to an arbitrary reference coordinate system in the tangent plane. Then, if the result with respect to any other coordinate system is required, one only needs to transform the result of the convolution in the reference coordinate system to the other coordinate system. The rotation-equivariance property is still valid if several of these filters are applied consecutively, *e.g.* in the deeper layers of a network. Rotation-equivariance enables the vector-valued convolution operator to always align the features in a local neighborhood of a point to the coordinate system at the point. Our network architecture builds upon Harmonic Nets [Worrall et al. 2017], which are used for rotation-equivariant learning on images. Therefore, we call the networks *Harmonic Surface Networks* (HSN).

We implement the Harmonic Surface Networks for triangle meshes. Based on circular harmonic filters, we derive discrete convolution filters that are equivariant with respect to basis transformations in the tangent spaces associated to the vertices of the mesh. The parameters of the filters separate radial and angular direction, which leads to a low number of parameters for each kernel, when compared to other filter kernels for surface meshes. We experimentally analyze the properties and performance of the HSNs and compare the performance to other geometric CNNs for surface meshes.

In summary, our main contributions are:

- We introduce Harmonic Surface Networks, which combine a vector-valued convolution operation on surfaces and rotation-equivariant filter kernels to provide a solution to the rotation ambiguity problem of geometric deep learning on surfaces.
- Based on circular harmonics, we derive convolution filters for meshes that have the desired rotational-equivariance properties at the discrete level, and, additionally, allow for separating learning of parameters in radial and angular direction.
- We analyze the Harmonic Surface Networks for surface meshes and compare the performance to alternative approaches.

2 RELATED WORK

In this section, we provide a brief overview of work on geometric deep learning closely related to our approach. For a comprehensive survey on the topic, we refer to [Bronstein et al. 2017].

Charting-based methods. Closest to our work are so-called *charting-based methods.* For convolution, these methods use local parametrizations to apply filter kernels to features defined on the surface. A seminal approach in this direction are the Geodesic CNNs (GCNN, [Masci et al. 2015]). They consider a parametric model of convolution filters in $\mathbb{R}^2$ using Gaussian kernels placed on a regular polar grid. For convolution, the filters are mapped to the surface using the Riemannian exponential map. Other intrinsic methods use anisotropic heat kernels [Boscaini et al. 2016], learn the shape parameters of the kernels [Monti et al. 2017], learn not only the parameters of the kernels but also pseudo-coordinates of the local parametrizations [Verma et al. 2018], or use B-Splines [Fey et al. 2018] or Zernike polynomials [Sun et al. 2018] instead of Gaussians.

A challenging problem for these constructions is the lack of canonical coordinate systems on a surface, inducing the rotation ambiguity problem, discussed in the introduction. To address the ambiguity, maximum pooling over the responses to a sample of different rotations can be used [Masci et al. 2015; Sun et al. 2018] or the coordinate systems can be aligned to the (smoothed) principal curvature directions [Boscaini et al. 2016; Monti et al. 2017; Pan et al. 2018]. Both approaches have their downsides. Angular pooling discards directional information and the alignment to principal curvature directions can be difficult as the principal curvature directions are not defined at umbilic points and can be unstable in regions around umbilic points. A sphere, for example, consists only of umbilic points. A solution to this problem are the Multi-Directional Geodesic CNNs (MDGCNN) [Poulenard and Ovsjanikov 2018]. At every point, the filters are evaluated with respect to multiple choices of coordinate systems. This can be formalized by describing the features by so-called directional functions instead of regular functions. Parallel transport is used to locally align the directional functions for convolution. Our approach provides a different solution to the rotation ambiguity problem that does not require computing the filters in multiple directions and storing the results. Once the filter in one direction is computed, the rotation-equivariance property of our filters allows us to obtain the results for the other directions by applying a rotation. We compare our approach to MDGCNN in Section 5. Parallel Transport Vector Convolution [Schonsheck et al. 2018] is a convolution operation for vector fields on manifolds that was used to learn from vector valued data on surfaces. Similar to this approach, we also use parallel transport to define convolution for vector fields on surface. A concept for the construction of Gauge-equivariant convolution filters on manifolds is introduced in [Cohen et al. 2019] along with a concrete realization of a Gauge CNN for the icosahedron. In recent work, an adaption of Gauge CNNs to surface meshes was introduced [de Haan et al. 2020].

Instead of local charts, one can also parametrize an area globally and then learn on the parameter domain [Haim et al. 2019; Maron et al. 2017]. An advantage of this approach is that standard CNNs can be applied to the parameter domain. A disadvantage is that global parametrizations lead to larger metric distortion than local parametrizations. In addition, typically different global parametrization methods are needed for surfaces of different genus.

Spectral methods and graph CNNs. An alternative to charting-based methods are spectral methods. For images, CNNs can operate in the Fourier domain. Spectral Networks [Bruna et al. 2014] generalize CNNs to graphs using the spectrum of a graph Laplacian. To reduce the computational complexity, ChebNet [Defferrard et al. 2016] and Graph Convolutional Networks (GCN) [Kipf and Welling 2017] use local filters based on the graph Laplacian. Recently, various approaches for defining CNNs on graphs have been introduced, we refer to [Wu et al. 2019; Zhou et al. 2018] for recent surveys. This line of work diverges from our approach since we specialize to discrete structures that describe two-dimensional manifolds. Furthermore, we aim at analyzing the surface underlying a mesh, which means our method aims to be agnostic of the connectivity of the mesh, *i.e.* the graph underlying a mesh. The concept of local filtering has also been applied to surfaces using the Laplace–Beltrami and Dirac operator [Kostrikov et al. 2018]. MeshCNN [Hanocka et al. 2019] generalizes graph CNNs to mesh structures by defining convolution and pooling operations for triangle meshes.

Point clouds. PointNet [Qi et al. 2017a] is an approach for creating CNNs for unordered sets of points. First, a neighborhood is constructed around each point with a radius ball. To retrieve a response for point i, a function is applied to each neighbor of i and the maximum activation across i's neighbors is stored as the new response for i. PointNet++ [Qi et al. 2017b] is an extension of PointNet with hierarchical functionality. Because PointNet applies the same function to each neighbor, it is effectively rotation-invariant. DGCNN [Wang et al. 2019] extends PointNet++ by dynamically constructing neighborhood graphs within the network. This allows the network to construct and learn from semantic neighborhoods, instead of mere spatial neighborhoods. PointCNN [Li et al. 2018] learns a χ-transformation from the point cloud and applies the convolutions to the transformed points. TextureNet [Huang et al. 2019] uses 4-rotational symmetric fields to define the convolution domains and applies operators that are invariant to 4-fold symmetries. While we evaluate our approach on meshes, our method can be used to process point clouds sampled from a surface.

Symmetric spaces. Specialized approaches for symmetric surfaces such as the sphere [Cohen et al. 2018; Kondor et al. 2018] have been proposed. On the one hand, the approaches profit from the highly symmetric structure of the surfaces, while on the other hand they are limited to data defined on these surfaces.

Rotation-equivariance. Our approach builds on Harmonic Networks [Worrall et al. 2017]. This work is part of a larger effort to create group-equivariant networks. Different approaches such as steerable filters [Cohen and Welling 2017; Freeman and Adelson 1991; Liu et al. 2012; Weiler and Cesa 2019], hard-baking transformations in CNNs [Cohen and Welling 2016; Fasel and Gatica-Perez 2006; Laptev et al. 2016; Marcos et al. 2016], and learning generalized transformations [Hinton et al. 2011] have been explored. Most relevant to our approach are steerable filters, since we use features that can be transformed, or steered, with parallel transport. The core idea of steerable filters is described by [Freeman and Adelson 1991] and applied to learning by [Liu et al. 2012]. The key ingredient for these steerable filters is to constrain them to the family of circular harmonics. [Worrall et al. 2017] added a rotation offset and multi-stream architecture to develop Harmonic Networks. The filters in Harmonic Networks are designed in the continuous domain and mapped to a discrete setting using interpolation. Harmonic Networks was built on by [Thomas et al. 2018] for Tensor Field Networks. Tensor Field Networks achieve rotation- and translation-equivariance for 3D point clouds by moving from the family of circular harmonics to that of spherical harmonics. In doing so, they lose the phase offset parameter, as it is not commutative in SO(3). Spherical harmonics were also used for rigid motion-invariant processing of point clouds [Poulenard et al. 2019].

3 BACKGROUND

Harmonic Networks. Harmonic Nets (H-Nets) [Worrall et al. 2017] are rotation-equivariant networks that can be used to solve computer vision tasks, such as image classification or segmentation, in such a way that a rotation of the input does not affect the output of the network. H-Nets restrict their filters to circular harmonics, resulting in the following filter definition:

$$W_m(r, \theta, R, \beta) = R(r)e^{\mathrm{i}(m\theta+\beta)}, \tag{1}$$

where r and θ are polar coordinates, $R : \mathbb{R}_+ \longrightarrow \mathbb{R}$ is the *radial profile*, $\beta \in [0, 2\pi)$ is a *phase offset*, and $m \in \mathbb{Z}$ is the rotation order. The cross-correlation of W_m with a complex function x at a point p is given by the integral

$$[W_m \star x](p) = \int_0^{\epsilon} \int_0^{2\pi} R(r)e^{\mathrm{i}(m\theta+\beta)}x(r, \theta)\, r\, d\theta\, dr. \tag{2}$$

This filter is rotation-equivariant with respect to rotations of the domain of the input to the filter:

$$[W_m \star x^{\phi}](p) = e^{\mathrm{i}m\phi}[W_m \star x^0](p), \tag{3}$$

where $x^0(r, \theta)$ is a complex function and $x^{\phi}(r, \theta) = x(r, \theta - \phi)$ is the same function defined on a rotated domain. The rotation order m determines how the output of the filters changes when the domain of the input is rotated. For $m = 0$, the result does not change and the filter is rotation invariant. For $m \geq 1$ the result is rotated by an angle that is m times the original angle, which we refer to as m-equivariance.

An important property of these filters is that, if filters of orders m_1 and m_2 are chained, rotation-equivariance of order $m_1 + m_2$ is obtained:

$$[W_{m_1} \star [W_{m_2} \star x^{\phi}] = e^{\mathrm{i}m_1\phi}e^{\mathrm{i}m_2\phi}[W_{m_1} \star [W_{m_2} \star x^0] \tag{4}$$

$$= e^{\mathrm{i}(m_1+m_2)\phi}[W_{m_1} \star [W_{m_2} \star x^0]. \tag{5}$$

This is integral to the rotation-equivariance property of the network as a whole. The network architecture of H-Nets is structured in

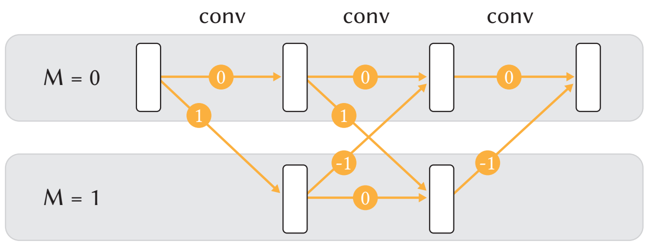

Fig. 2. Harmonic Networks separate the result of different rotation order convolutions into streams of M-equivariance.

separate streams per rotation order as illustrated in Figure 2. For each feature map inside a stream of order M, we require that the sum of rotation orders m_i along any path reaching that feature map equals M. In the last layer of an H-Net, the streams are fused to have the same rotation order. The rotation order of the output stream determines the class of equivariance of the network and is chosen to match the demands of the task at hand. For rotation invariant tasks, the last layer has order $m = 0$. Still, in the hidden layers of the network, rotation-equivariant streams capture and process information that is not rotation-invariant, which yields improved performance compared to networks built only from rotation-invariant filters.

Riemannian exponential map. Let v be a vector in the tangent plane $T_p\mathcal{S}$ at a point p of a surface $\mathcal{S}$. Then, there is exactly one geodesic curve starting at p in direction v. If we follow this geodesic curve until we have covered a length that equals the norm of v, we end up at a point q on the surface. The Riemannian exponential map associates each vector v in $T_p\mathcal{S}$ to the corresponding point q on the surface. This map is suitable as a local parametrization of the surface, because it is a local diffeomorphism, *i.e.*, bijective and smooth with smooth inverse. Furthermore, the mapping is an isometry in radial direction away from p. The construction of the Riemannian exponential map and its inverse, the logarithmic map, is illustrated in Figure 3. An example of the Riemannian exponential map is the azimuthal projection of the sphere, which is used in cartography and is included in the emblem of the United Nations.

In graphics, the Riemannian exponential map is used, for example, for texture decalling [Schmidt et al. 2006] and shape modeling [Schmidt and Singh 2010]. In geometric deep learning [Bronstein et al. 2017], it is used to map filter kernels defined on the tangent plane to filters defined on the surface and to lift functions defined on the surface to functions defined on the tangent plane. Recent approaches for computing the Riemannian exponential map on surface meshes are based on heat diffusion [Herholz and Alexa 2019; Sharp et al. 2019b]. These methods reduce the computation of the exponential map to solving a few sparse linear systems and can be accurately computed globally. Alternatives are approaches based on Dijkstra's algorithm [Melvær and Reimers 2012; Schmidt et al. 2006].

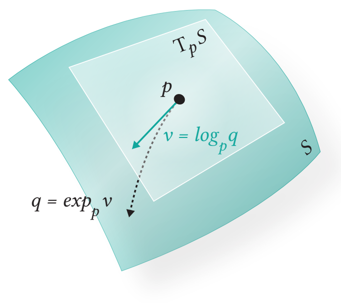

Fig. 3. The Riemannian exponential- and logarithmic map.

Parallel transport of vectors. In the Euclidean plane one can describe vector fields by specifying x and y coordinates relative to a global coordinate system. There is no such coordinate system on curved surfaces. To be able to compare vectors at neighboring points p and q of a surface, we use the parallel transport along the shortest geodesic curve c connecting the points. If v is a tangent vector at p which has the angle α with the tangent vector of c at p, then the vector transported to q is the tangent vector that has an angle of α with the tangent of c at q and has the same length as v.

To describe tangent spaces on meshes and to compute the parallel transport, we use the Vector Heat Method [Sharp et al. 2019b], which we also use to calculate the Riemannian exponential map.

4 METHOD

In this section, we describe the building blocks of Harmonic Surface Networks. We start by introducing notation and then discuss convolutions, linearities, non-linearities, and pooling. Finally, we discuss why the networks provide a solution to the rotation ambiguity problem by analyzing how the choices of coordinate systems in the tangent spaces affect the convolution operations and HSNs.

Notation. The features in an HSN are associated to the vertices (or just a subset of the vertices) of a triangle mesh. To each vertex, we assign a coordinate system in the tangent plane at the vertex and use complex numbers to represent tangent vectors with respect to the coordinate system. We denote the feature vector of rotation order M in network layer l at vertex i by $\mathbf{x}^l_{i,M}$. To simplify the notation, we leave out the indices l and M when they are not relevant. For HSN, these feature vectors are complex-valued. Every operation applied to these vectors is performed element-wise. For example, when we multiply a feature vector with a complex number, each component of the vector is multiplied by this number. We use parallel transport along the shortest geodesic from vertex j to vertex i to transport the feature vectors. The transport depends on the geometry of the mesh, the chosen coordinate systems at the vertices, and the rotation order of the feature. It amounts to a rotation of the features. In practice, we store the rotation as a complex number with the angle of rotation ϕ_{ji} as its argument and apply rotations to vectors via complex multiplication:

$$P_{j\to i}(\mathbf{x}_{j,M}) = e^{\mathrm{i}(M\phi_{ji})}\mathbf{x}_{j,M}. \tag{6}$$

For every vertex, we construct a parametrization of a local neighborhood of the vertex over the tangent plane using the Riemannian logarithmic map. We represent the map by storing polar coordinates r_{ij} and θ_{ij} with respect to the coordinate system in the tangent plane for every vertex j in the neighborhood of vertex i.

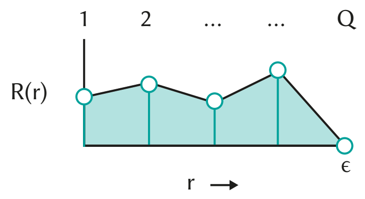

Fig. 4. We parametrize the radial profile by learning the value at Q equally spaced rings and linearly interpolating for values in between.

Convolution. The convolution layers of HSNs combine the rotation-equivariance of circular harmonics with the transport of features along surfaces. We use compactly supported filter kernels to limit the number of neighboring features that are used in a convolutional layer. Let $\mathcal{N}_i$ denote the set of vertices that contribute to the convolution at i. In the case of continuous features on a Euclidean space, the correlation with W_m is given by an integral, see (2). For discrete meshes, we approximate the integral with a sum and use parallel transport and the Riemannian logarithmic map to evaluate the filter:

$$\mathbf{x}^{(l+1)}_{i,M+m} = \sum_{j \in \mathcal{N}_i} w_j \left(R(r_{ij}) e^{\mathrm{i}(m\theta_{ij}+\beta)} P_{j\to i}(\mathbf{x}^{(l)}_{j,M}) \right), \quad (7)$$

where r_{ij} and θ_{ij} are the polar coordinates of point j in $T_i\mathcal{S}$, and $P_{j\to i}$ is the transport of features defined in (6). The integration weights w_j are

$$w_j = \frac{1}{3} \sum_{jkl} A_{jkl} \quad (8)$$

where A_{jkl} is the area of triangle jkl and the sum runs over all triangles of the mesh containing vertex j. The weights can be derived from numerical integration with piecewise linear finite elements, see [Wardetzky et al. 2007].

The radial profile R and the phase offset β are the learned parameters of the filter. To represent the radial profile, we learn values at equally spaced locations in the interval $[0, \epsilon]$, where ϵ provides a bound on the support of the filter. To get a continuous radial profile, the learned values are linearly interpolated as illustrated in Figure 4. At a point with radial coordinate r_{ij}, the radial profile is

$$R(r_{ij}) = \sum_{q}^{Q} \mu_q(r_{ij}) \rho_q, \quad (9)$$

where $\mu_q(r_{ij})$ is a linear interpolation weight and ρ_q a learned weight matrix. We set $\rho_Q = 0$, which bounds the support of the kernel.

To speed up training, we precompute three components: the integration weight, the interpolation weight to each radial profile point, and the rotation by the angle θ_{ij}:

$$\text{precomp}_{ij} = \left(w_j \mu_q(r_{ij}) e^{\mathrm{i}m\theta_{ij}} \right). \quad (10)$$

We precompute the polar coordinates and integration weights with the Vector Heat method [Sharp et al. 2019b]. The precomputation is stored in a [Q x 2] matrix for each (i, j) pair.

Linearities. Linearities are applied to complex features by applying the same linearity to both the real and imaginary component, resulting in a linear combination of the complex features.

Non-Linearities. We follow Harmonic Networks [Worrall et al. 2017] for complex non-linearities: non-linearities are applied to the radial component of the complex features, in order to maintain the rotation-equivariance property. In our experiments, we used a complex version of ReLU

$$\mathbb{C}\text{-ReLU}_b(\mathbf{X}e^{\mathrm{i}\theta}) = \text{ReLU}(\mathbf{X} + b)e^{\mathrm{i}\theta}, \quad (11)$$

where b is a learned bias added to the radial component. If the non-linearity were to be applied without bias, it would be an identity operation, as the radius is always positive.

Pooling and unpooling. Pooling layers downsample the input by aggregating regions to representative points. We need to define an aggregation operation suitable for surfaces and a way to choose representative points and create regions. We use farthest point sampling and cluster all non-sampled points to sampled points using geodesic nearest neighbors.

The aggregation step in pooling is performed with parallel transport, since the complex features of points within a pooling region do not exist in the same coordinate system. Thus, we define the aggregation step for representative point i with points j in its pooling cluster C_i as follows:

$$\mathbf{x}_{i,M} = \square_{j \in C_i} P_{j\to i}(\mathbf{x}_{j,M}), \quad (12)$$

where $\square$ is any aggregation operation, such as sum, max, or mean. In our implementation, we use mean pooling. Pooling happens *within* each rotation order stream, hence the rotation order identifier M for both $\mathbf{x}_i$ and $\mathbf{x}_j$.

The sampling of points per pooling level and construction of corresponding kernel supports are performed as a precomputation step, so we can compute the logarithmic map and parallel transport for each pooling level in precomputation.

Unpooling layers upsample the input by propagating the features from the points sampled in the pooling layer to their nearest neighbors. Parallel transport is again applied to align coordinate systems.

Rotation orders. To maintain rotation-equivariance throughout the network, the output of the filters is separated in streams of rotation orders. The output of a filter applied to $\mathbf{x}_{j,M}$ with rotation order m should end up in rotation order stream $M' = M + m$. The output from two convolutions resulting in the same stream is summed. For example, a convolution on $\mathbf{x}_{j,1}$ with $m = -1$ and a convolution on $\mathbf{x}_{j,0}$ with $m = 0$ both end up in the stream $M' = 0$ and are summed. A visual overview can be found in Figure 6, on the right. We only apply parallel transport to inputs from the $M > 0$ rotation-order streams, as the values in the $M = 0$ stream are rotation-invariant.

Properties of HSNs. In the following, we show that the result of convolutions (7) as well as the layers of HSNs commute with transformations of the coordinate systems in the tangent spaces. This means that we can choose any coordinate system in the tangent

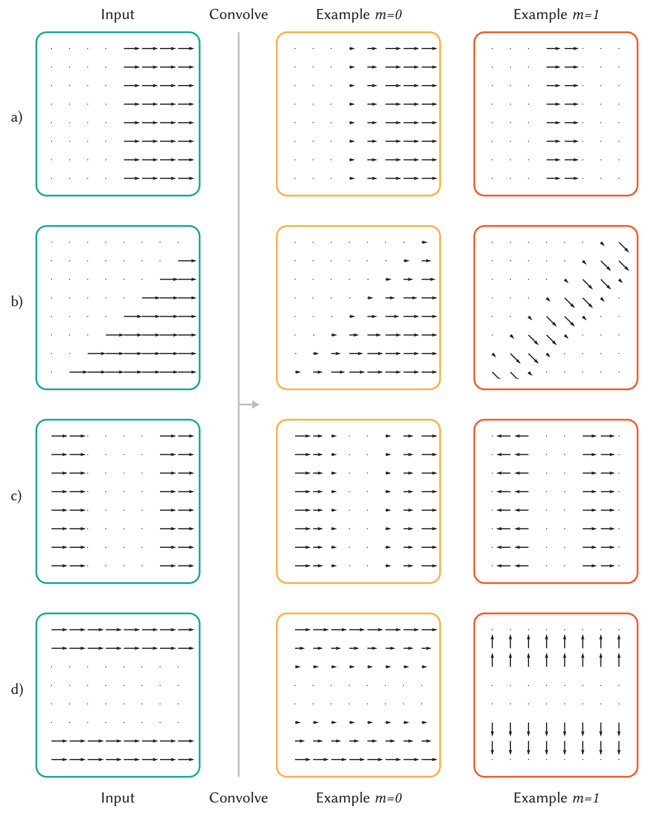

Fig. 5. Examples of simple input vector fields (one feature) and the response of a convolution with $m = 0$ and $m = 1$ kernels.

spaces and use it to build, train, and evaluate our networks. A different choice of the coordinate systems leads to the same network, only with transformed features.

As a consequence, HSNs do not suffer from the rotation ambiguity problem, which we described in the introduction. The rotation ambiguity problem is caused by the fact that due to the curvature of a surface, there is no consistent choice of coordinate systems on a surface. So if a filter that is defined on $\mathbb{R}^2$ is to be applied everywhere on the surface, coordinate systems in the tangent planes must be defined. Since there is no canonical choice of coordinate systems, the coordinate systems at pairs of neighboring points are not aligned. Due to the commutativity property (Lemma 4.1), a convolution layer of our HSNs can use the features that are computed in arbitrary coordinate systems and locally align them when computing the convolutions.

In contrast, a network without that property cannot locally align the features. If one would want to align the coordinate systems' features locally, they would have to recompute the features starting from the first layer. Since this is not feasible, one needs to work with non-aligned coordinate systems. The result is that the same kernel is applied with different coordinate systems at each location. Moreover, as features are combined in later layers, the network will learn from local relations of coordinate systems. This is undesirable, as these relations hold no meaningful information: they arise from arbitrary circumstances and choices of coordinate systems.

To prove that the features of HSNs commute with the coordinate changes, this property must be shown for the individual operations. The non-linearity is invariant to changes of the basis as it only operates on the radial coordinates. Here, we restrict ourselves to convolution, as for pooling, one can proceed analogous to this proof.

Lemma 4.1. *The convolution (7) commutes with changes of coordinate systems in the tangent planes.*

Proof. We represent the coordinate system of a tangent space by specifying the x-axis. Coordinates of points are then given by a complex number. If we rotate the coordinate system at vertex i by an angle of $-\phi$, the features of order M transform to

$$\tilde{\mathbf{x}}_{i,M} = e^{\mathrm{i}M\phi}\mathbf{x}_{i,M}, \tag{13}$$

where $\tilde{\mathbf{x}}_{i,M}$ is the feature in the rotated coordinate system. The change of coordinate system also affects the polar coordinates of any vertex j in the tangent plane of i and the transport of features from any vertex j to i

$$\tilde{\theta}_{ij} = \theta_{ij} + \phi \qquad \tilde{P}_{j\to i}(\mathbf{x}_{j,M}) = e^{\mathrm{i}M\phi}P_{j\to i}(\mathbf{x}_{j,M}), \tag{14}$$

where $\tilde{\theta}_{ij}$ is the angular coordinate of j in the tangent plane of i with respect to the rotated coordinate system and $\tilde{P}$ is the transport of features with respect to the rotated coordinate system. Then, convolution with respect to the rotated coordinate system is given by

$$\tilde{\mathbf{x}}^{(l+1)}_{i,M+m} = \sum_{j\in\mathcal{N}_i} w_j \left(R(r_{ij})e^{\mathrm{i}(m\tilde{\theta}_{ij}+\beta)}\tilde{P}_{j\to i}(\mathbf{x}^{(l)}_{j,M})\right). \tag{15}$$

Plugging (14) into (15), we get

$$\tilde{\mathbf{x}}^{(l+1)}_{i,M+m} = \sum_{j\in\mathcal{N}_i} w_j \left(R(r_{ij})e^{\mathrm{i}m\phi}e^{\mathrm{i}(m\theta_{ij}+\beta)}e^{\mathrm{i}M\phi}P_{j\to i}(\mathbf{x}^{(l)}_{j,M})\right) \tag{16}$$

$$= e^{\mathrm{i}(M+m)\phi}\mathbf{x}^{(l+1)}_{i,M+m}. \tag{17}$$

The latter agrees with the basis transformations of the features, see (13). If the coordinate system at a vertex j that is neighboring i is rotated by some angle, then this affects the transport of features $P_{j\to i}$ from j to i and the feature $\mathbf{x}^{(l)}_{j,M}$. However, since the transport and feature are rotated by the same angle but in opposite directions, the term $P_{j\to i}(\mathbf{x}^{(l)}_{j,M})$ is not affected. □

We visualize the convolution of a simple $m = 0$ feature with $m = 0$ and $m = 1$ filters in Figure 5. For each example, $\beta = 0$ and $R(r) = 1 - r$. The inputs are basic patterns: a) a vertical edge, b) a diagonal edge, c) two vertical edges, and d) two horizontal edges. We observe that $m = 0$ convolutions smooth the signal and $m = 1$ convolutions activate on edge boundaries. We can also see Equation 13 at work: the input in a) and b) differs by a 45°-rotation of the domain. The $m = 0$ outputs therefore are related by the same rotation of the domain. The equivariant $m = 1$ outputs are related by a rotation of the domain and an additional rotation of the features. The same holds for c) and d), now with a rotation by 90°.

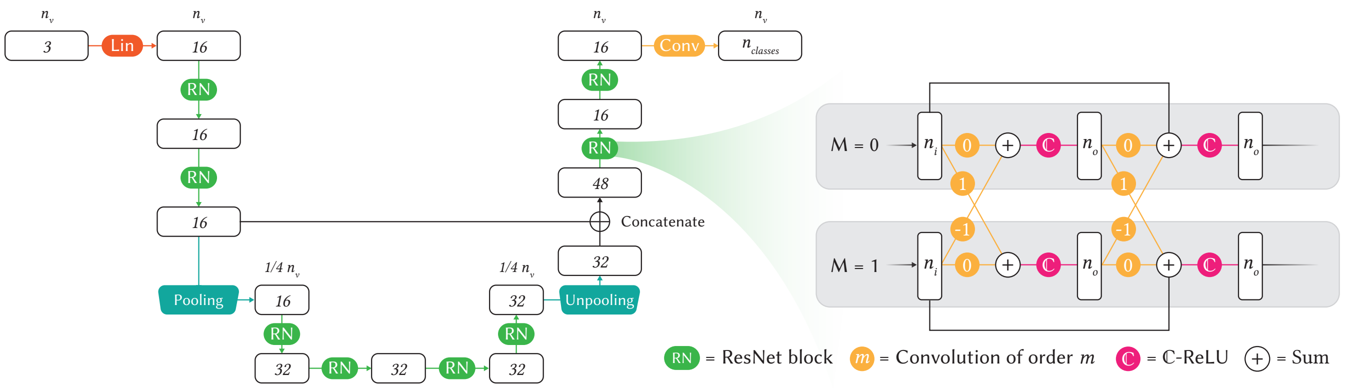

Fig. 6. HSN U-ResNet architecture used for correspondence and shape segmentation. On on the left, the full architecture; on the right, a detail of the ResNet Block.

Discretization. In this paper, we detail Harmonic Surface Networks for meshes, as they are sparse and convenient representations of surfaces. Yet, we have attempted to formulate the building blocks for HSNs as general as possible. Thus, one could replace the use of the words 'mesh' and 'vertex' with 'point cloud' and 'point' and the method would remain largely the same. The main differences would be (1) how to compute the logarithmic map and parallel transport and (2) how to define a weighting per point. The first question has been answered by [Sharp et al. 2019b]: the Vector Heat method can compute a logarithmic map for any discretization of a surface. The latter can be resolved with an appropriate integral approximation scheme.

5 EXPERIMENTS

The goal of our experiments is twofold: we will substantiate our claims about improved performance with comparisons against state-of-the-art methods and we aim to evaluate properties of HSNs in isolation, in particular the benefits of the multi-stream architecture and rotation-equivariance.

5.1 Implementation

In the following paragraphs we discuss the architecture and data processing used in our experiments.

Following recent works in geometric deep learning [Hanocka et al. 2019; Poulenard and Ovsjanikov 2018; Verma et al. 2018], we employ a multi-scale architecture with residual connections. More specifically, we recreate the deep U-ResNet architecture from [Poulenard and Ovsjanikov 2018]. The U-ResNet architecture consists of four stacks of ResNet blocks, organized in a U-Net architecture with pooling and unpooling layers (Figure 6). Each stack consists of two ResNet blocks, which, in turn, consist of two convolutional layers with a residual connection. We use 16 features in the first scale and 32 features in the second scale. Finally, we configure our network with two rotation order streams: $M = 0$ and $M = 1$, learning a rotation-equivariant kernel for every connection between streams.

For pooling operations, we sample a quarter of the points on the surface using farthest point sampling and take the average of the nearest neighbors. These neighbors are computed with the Heat Method [Crane et al. 2013], by diffusing indices from sampled points to all other points with a short diffusion time ($t = 0.0001$). With each pooling step, we grow the filter support by $1/\sqrt{\text{ratio}}$. Thus, we grow the radius with a factor 2. At the unpooling stage, we propagate the features from the points sampled in the pooling stage to the nearest neighbors, using parallel transport. We use ADAM [Kingma and Ba 2015] to minimize the negative log-likelihood and train for 100 epochs on correspondence and 50 epochs for shape segmentation and mesh classification.

For each experiment, we normalize the scale of each shape, such that the total surface area is equal to 1. We then compute local supports for each vertex i by assigning all points within a geodesic disc of radius ϵ to its neighborhood $\mathcal{N}_i$. We normalize the weighting described in Equation 8 by the sum of weights for each neighborhood, to compensate for the different sizes of support of the filters in the different layers, in particular, after pooling and unpooling.

Next, we employ the Vector Heat Method out of the box [Sharp et al. 2019a,b] to compute the logarithmic map and parallel transport for each neighborhood. For our multi-scale architecture, we first compute local supports for each scale and coalesce the resulting graph structures into one multi-scale graph. This way, we can precompute the necessary logarithmic maps in one pass. Wherever we can, we use raw xyz-coordinates as input to our network. To increase the robustness of the network against transformations of the test set, we randomly scale and rotate each shape with a factor sampled from $\mathcal{U}(0.85, 1.15)$ and $\mathcal{U}(-\frac{1}{8}\pi, \frac{1}{8}\pi)$, respectively. For correspondence, we use SHOT descriptors, to provide a clean comparison against previous methods.

The networks are implemented using PyTorch Geometric [Fey and Lenssen 2019]. The implementation can be retrieved from https://github.com/rubenwiersma/hsn.

5.2 Comparisons

Following from the benefits outlined in the introduction, we expect HSNs to show improved results over existing spatial methods, even though fewer parameters are used for each kernel. We experimentally validate these expected benefits by applying HSN on three

Table 1. Results for shape classification on 10 training samples per class of HSN against previous work.

Method	Accuracy
HSN (ours)	**96.1%**
MeshCNN	91.0%
GWCNN	90.3%
GI	88.6%
MDGCNN	82.2%
GCNN	73.9%
SG	62.6%
ACNN	60.8%
SN	52.7%

Table 2. Results for shape segmentation by HSN and related methods.

Method	# Features	Accuracy
HSN (ours)	3	91.14%
MeshCNN	5	**92.30%**
SNGC	3	91.02%
PointNet++	3	90.77%
MDGCNN	64	89.47%
Toric Cover	26	88.00%
DynGraphCNN	64	86.40%
GCNN	64	86.40%
ACNN	3	83.66%

tasks: shape classification on SHREC [Lian et al. 2011], correspondence on FAUST [Bogo et al. 2014], and shape segmentation on the human segmentation dataset proposed by [Maron et al. 2017].

Shape classification. We train an HSN to perform classification on the SHREC dataset [Lian et al. 2011], consisting of 30 classes (Figure 7). We only train on 10 training samples per class. Like [Hanocka et al. 2019], we take multiple random samplings from the dataset to create these training sets and average over the results. We train for 50 epochs, compared to the 200 epochs used in previous works. This task is challenging due to the low number of training samples and labels. Therefore, we reduce the size of our network and consequently, the number of parameters to learn. We only use the first half of the U-ResNet architecture, with only one ResNet block per scale. To obtain a classification, we retrieve the radial components from the last convolutional layer, followed by a global mean pool. The initial radius of our filters is $\epsilon = 0.2$ and the number of rings in the radial profile is 6.

For our comparisons, we cite the results from [Hanocka et al. 2019] and [Ezuz et al. 2017], comparing our method to MeshCNN [Hanocka et al. 2019], SG [Bronstein et al. 2011], SN [Wu et al. 2015], GI [Sinha et al. 2016], and GWCNN [Ezuz et al. 2017]. Additionally, we train MDGCNN [Poulenard and Ovsjanikov 2018], GCNN [Masci et al. 2015], and ACNN [Boscaini et al. 2016] with the exact same architecture as HSN. The results are outlined in Table 1. HSN outperforms all previous methods, demonstrating the effectiveness of our approach, even for lower training times. One explanation for the large gap in performance is the low number of training samples. HSN uses fewer parameters, resulting in fewer required training samples. This is supported by results in [Worrall et al. 2017], who also show higher performance for small datasets compared to other methods. The low number of samples also explains why some non-learning methods outperform learning methods. An additional problem faced by ACNN is the low quality of the meshes, resulting in a disparate principal curvature field. This obstructs the network from correctly relating features at neighboring locations and degrades performance. This same effect is observed when applying HSN without parallel transport, aligned to principal curvature directions (Table 5).

Shape segmentation. Next, we demonstrate HSN on shape segmentation. We train our network to predict a body-part annotation for each point on the mesh. We evaluate our method on the dataset proposed by Maron et al. [2017], which consists of shapes from FAUST ($n = 100$) [Bogo et al. 2014], SCAPE ($n = 71$) [Anguelov et al. 2005], Adobe Mixamo ($n = 41$) [Adobe 2016], and MIT ($n = 169$) [Vlasic et al. 2008] for training and shapes from the human category of SHREC07 ($n = 18$) for testing. The variety in sources provides a variety in mesh structures as well, which tests HSN's robustness to changes in mesh structure.

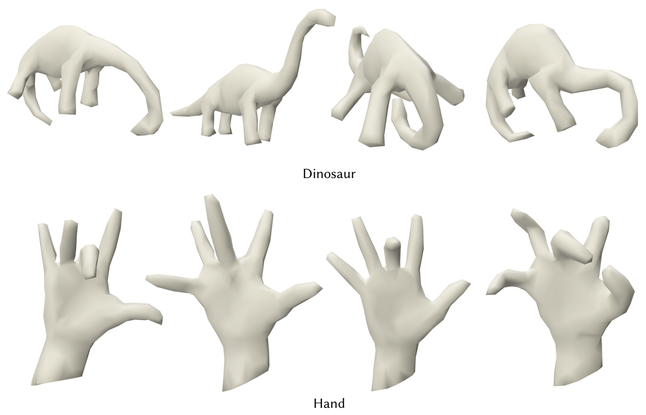

Fig. 7. Two example classes with four shapes each from the SHREC shape classification dataset.

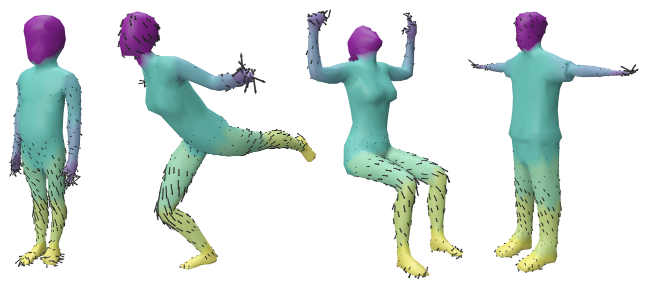

Fig. 8. Vector-valued featuremap and label predictions from our Harmonic Surface Network trained on shape segmentation.

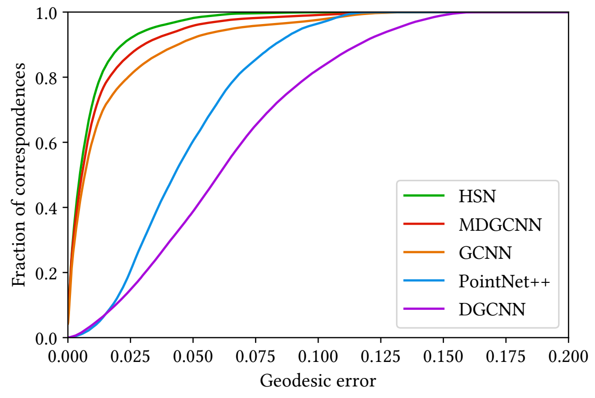

Fig. 9. Fraction of correspondences for a given geodesic error on the remeshed FAUST dataset using HSN (ours), MDGCNN, GCNN, PointNet++, and DGCNN.

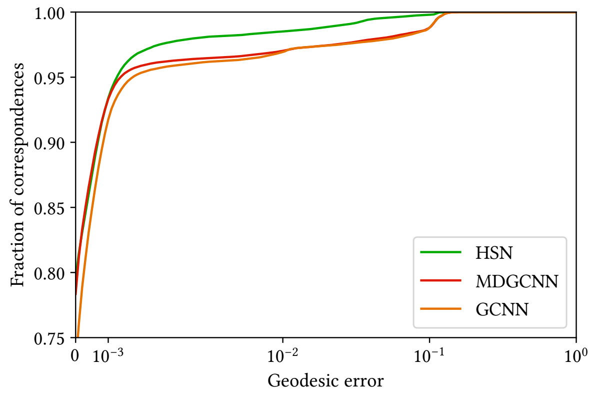

Fig. 10. Fraction of correspondences for a given geodesic error on the original FAUST dataset using HSN (ours), MDGCNN, and GCNN. Note: the x axis is set to logarithmic scale.

We use the U-ResNet architecture, providing xyz-coordinates as input and evaluate the class prediction directly from the final convolutional layer. The logarithmic map and parallel transport are computed using the original meshes from [Maron et al. 2017]. To limit the training time, 1024 vertices are sampled on each mesh using farthest point sampling to be used in training and testing. This type of downsampling is also applied by [Hanocka et al. 2019] (to 750 vertices) for similar reasons. The initial support of our kernels is $\epsilon = 0.2$ and the number of rings in the radial profile is 6.

We report the accuracy as the fraction of vertices that was classified correctly across all test shapes. For the comparisons, we cite the results from [Poulenard and Ovsjanikov 2018], [Haim et al. 2019] and [Hanocka et al. 2019]. HSN produces competitive results compared to state-of-the-art methods, performing only slightly worse than MeshCNN.

To understand the complex features learned by the network, we visualize the features on the model, alongside our predictions for segmentation. We can interpret the complex features as intrinsic vectors on the surface and visualize them as such using Polyscope [Sharp 2019]. Figure 8 shows a single feature in the $M = 1$ stream from the second-to-last layer. We observe high activations on certain bodyparts (legs, hands) and note that the intrinsic vectors are aligned. Our interpretation is that the corresponding filter 'detects' legs and hands. The alignment of features is beneficial to the propagation of these activations through the network; if features are oriented in opposing directions, they can cancel each other out when summed.

Correspondence. Correspondence finds matching points between two similar shapes. We evaluate correspondence on the widely used FAUST dataset [Bogo et al. 2014]. FAUST consists of 100 scans of human bodies in 10 different poses with ground-truth correspondences. We set up the network to predict the index of the corresponding vertex on the ground-truth shape. To this end, we add two fully connected layers (FC256, FCN_{vert}) after the U-ResNet architecture. The initial radius of our filters is $\epsilon = 0.1$ and the number of rings in the radial profile is 2.

We train on the first 80 shapes and report the fraction of correct correspondences for a given geodesic error on the last 20 models (Figures 9 and 10). As input to our network, we provide 64-dimensional SHOT descriptors, computed for 12% of the shape area. Similar to Poulenard and Ovsjanikov [2018], we train our network on a remeshed version of the FAUST dataset as well, since the original FAUST dataset exhibits the same connectivity between shapes. The remeshed dataset is a challenge that is more representative of real applications, where we cannot make any assumptions about the connectivity or discretization of our input. Having recreated the same architecture, with the same input features, we can fairly compare our method to MDGCNN and report the results obtained by Poulenard and Ovsjanikov [2018].

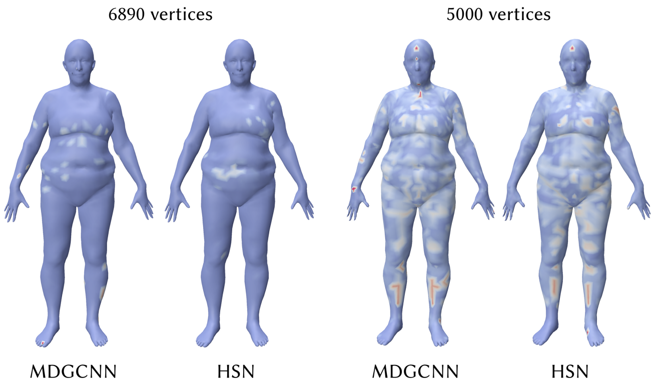

Fig. 11. Geodesic error visualised on the test shapes, shown: the first test shape for MDGCNN and HSN, for both the original and remeshed dataset.

The results in Figures 9, 10, and 11 show that HSN outperforms the compared state-of-the-art methods. More importantly, HSN improves existing results on the remeshed FAUST dataset, demonstrating the robustness of the method to changes in the discretization of the surface.

Parameter count and memory usage. The following calculation shows that HSN achieves this performance using fewer parameters and with less impact on memory during computation. Let n_i and n_o be the number of input and output features, n_ρ and n_θ the number of rings and angles on the polar grid, and n_m the number of rotation order streams. MDGCNN [Poulenard and Ovsjanikov 2018] and GCNN [Masci et al. 2015] learn a weight matrix for every location on a polar grid, resulting in a parameter complexity of $O(n_i\, n_o\, n_\rho\, n_\theta)$. HSN learns the same weight matrix, but only for the radial profile and the phase offset. We chose to learn these weights separately for every connection between streams, resulting in a parameter complexity of $O(n_i\, n_o\, (n_\rho + 1)\, n_m^2)$. If one chooses to learn weights per stream, which the original H-Nets opt for, this is reduced to $O(n_i\, n_o\, (n_\rho + 1)\, n_m)$. Removing the dependency on n_θ has a high impact on the number of parameters: for $n_\rho = 2$ and $n_\theta = 8$, HSN uses 75% of the parameters used by MDGCNN [Poulenard and Ovsjanikov 2018] (122880 vs. 163840), only considering the convolutional layers. If we were to use the same number of rings and angles as used by GCNN [Masci et al. 2015], $n_\rho = 5$ and $n_\theta = 16$, HSN would use only 30% of the parameters used by other spatial methods.

Concerning space complexity, MDGCNN [Poulenard and Ovsjanikov 2018] stores the result from every rotation throughout the network, multiplying the memory complexity of the model by the number of directions used, which tends to be between 8 and 16. In comparison, HSN's complex-valued features only increase the memory consumption for storing the separate streams and complex features. For two streams, this increases the space complexity by a factor of 4. This is important for the ability to scale our method to higher-resolution shapes and larger datasets, necessary for use in applications.

5.3 Evaluation

Aside from comparisons with other methods, we intend to get a deeper understanding of the properties of HSN; specifically, the benefit of the $M = 1$ stream w.r.t. rotation-invariant methods and a single $M = 0$ stream.

To evaluate the benefit of different rotation order streams in our method, we compare two different stream configurations of our method: a single-stream architecture ($M = 0$) and a double-stream architecture ($M = 0$ and $M = 1$). We limit this experiment to only these two configurations, because (1) experiments from [Worrall et al. 2017] demonstrate that rotation order streams of $M > 1$ do not significantly improve the performance of the network and (2) the $M = 0$ and $M = 1$ stream features have an intuitive interpretation on the surface as scalar values and intrinsic vectors, respectively.

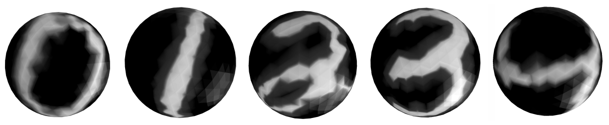
Fig. 12. Rotated MNIST mapped to a sphere

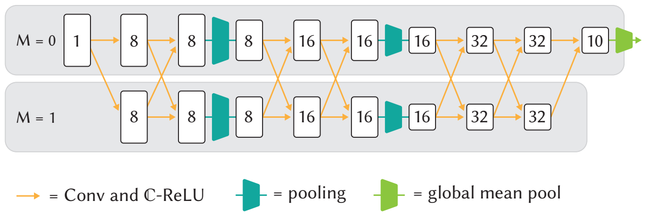

Fig. 13. Architecture for classification of Rotated MNIST.

Table 3. Results of HSN tested on Rotated MNIST mapped to a sphere for a single- and double-stream configuration.

Method	Streams ($M = \ldots$)	Accuracy
HSN	0, 1	**94.10%**
HSN	0	70.68%
HSN (parameters x4)	0	75.57%

We evaluate the two configurations on a new task: classification of digits from the Rotated MNIST mapped to a sphere, as well as the shape segmentation and classification tasks from the comparison experiments.

Rotated MNIST on a sphere. We use an elliptical mapping [Fong 2015] to map the grayscale values from the images of the Rotated MNIST dataset [Larochelle et al. 2007] to the vertices of a unit sphere with 642 vertices. The Rotated MNIST dataset consists of 10000 randomly rotated training samples, 2000 validation samples, and 50000 test images separated in 10 classes. We use an architecture similar to the one in [Worrall et al. 2017]: Conv8, Conv8, Pool0.5, Conv16, Conv16, Pool0.25, Conv32, Conv32, Conv10 (Figure 13). The kernel supports for each scale are the following: 0.3, 0.45, 0.8 (the geodesic diameter of the unit sphere is π).

The two stream architecture demonstrates significant improvement over the single-stream architecture with a higher parameter count to compensate for using only one stream: 94.10% vs. 75.57% (Table 3). This affirms the benefit of rotation-equivariant streams for learning signals on surfaces.

Shape Segmentation and Classification. We repeat our experiments for shape segmentation and classification in four different configurations: the double-stream configuration used in section 5.2; a single-stream configuration; a single-stream configuration with four times the parameters; and the double stream configuration aligned to a smoothed principal curvature field, instead of local alignment with parallel transport. The single-stream configurations aim to provide insight into the benefit of the rotation-equivariant stream and to rule out the sheer increase in parameters as the cause of this performance boost. The last configuration shows the benefit of locally aligning rotation-equivariant features over aligning kernels to

Table 4. Results of HSN tested on shape segmentation for multiple configurations.

Method	Streams ($M = \ldots$)	Accuracy
HSN	0, 1	**91.14%**
HSN	0	88.74%
HSN (parameters ×4)	0	87.25%
HSN (pc aligned)	0, 1	86.22%

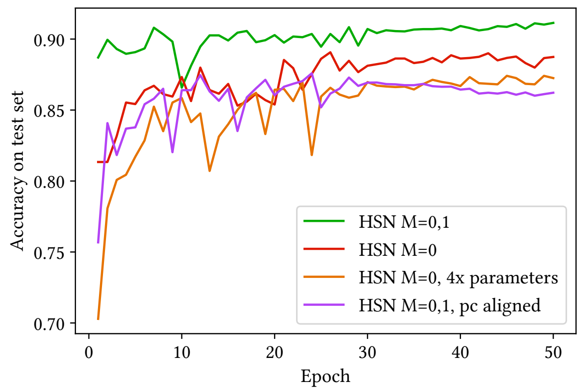

Fig. 14. Validation accuracy per training epoch several configurations of HSN on shape segmentation.

smoothed principal curvature fields, as is done by ACNN [Boscaini et al. 2016].

The results in Table 4 and Table 5 show that the double-stream architecture improves the performance from satisfactory to competitive. Furthermore, an increase in parameters has a negative impact on performance, likely due to the relatively low number of training samples. This becomes even more apparent when comparing the learning curves for each configuration in Figure 14: the double-stream configuration is more stable and performs better than both single-stream configurations. These results support the benefit of the rotation-equivariant stream.

Finally, we find that HSN performs significantly better when using parallel transport than when aligned with a smoothed principal curvature direction (pc aligned): for shape segmentation, the benefit introduced by the $M = 1$ stream is diminished, and for shape classification, we observe a large drop in performance, likely induced by the coarseness of the meshes and the resulting low-quality principal curvature fields.

6 CONCLUSION

We introduce Harmonic Surface Networks, an approach for deep learning on surfaces operating on vector-valued, rotation-equivariant features. This is achieved by learning circular harmonic kernels and separating features in streams of different equivariance classes. The advantage of our approach is that the rotational degree of freedom, arising when a filter kernel is transported along a surface, has no effect on the network. The filters can be evaluated in arbitrarily chosen coordinate systems. Due to the rotation-equivariance of the filters, changes in the coordinate system can be recovered after the convolutions have been computed by transforming the results of the convolution. The convolution operator uses this property and always locally aligns the features.

Table 5. Results of HSN tested on classification for multiple configurations.

Method	Streams ($M = \ldots$)	Accuracy
HSN	0, 1	**96.1%**
HSN	0	86.1%
HSN (pc aligned)	0, 1	49.7%

We implement this concept for triangle meshes and develop convolution filters that have the desired equivariance properties at the discrete level and allow for separating learning of parameters in the radial and angular direction. We demonstrated in our comparisons that HSNs are able to produce competitive or better results with respect to state-of-the-art approaches and analyzed the benefits of rotation-equivariant streams as an addition to rotation-invariant streams and parallel transport for local alignment compared to global alignment.

While we only implemented our method for meshes, every component of HSNs can be transferred to point clouds. We aim to extend our implementation and evaluate the benefits of HSNs for learning on point clouds.

Another promising direction is the application of outputs from the rotation-equivariant stream. We expect that the ability to learn intrinsic vectors on a surface can facilitate new applications in graphics and animation.

ACKNOWLEDGMENTS

We would like to thank Nicholas Sharp and Matthias Fey for their work on *Geometry Central* and *PyTorch Geometric*, Adrian Poulenard for sharing results and insights on MDGCNN, Thomas Kipf and Taco Cohen for helpful discussions on geometric deep learning and group-equivariant learning, and the anonymous reviewers for their constructive comments and suggestions. This work has been supported by Stichting Universiteitsfonds Delft and a gift from Google Cloud.

REFERENCES

Adobe. 2016. Adobe Mixamo 3D characters. www.mixamo.com.
Dragomir Anguelov, Praveen Srinivasan, Daphne Koller, Sebastian Thrun, Jim Rodgers, and James Davis. 2005. SCAPE: Shape Completion and Animation of People. *ACM Trans. Graph.* 24, 3 (2005), 408–416.
Federica Bogo, Javier Romero, Matthew Loper, and Michael J. Black. 2014. FAUST: Dataset and evaluation for 3D mesh registration. In *CVPR*. IEEE.
Davide Boscaini, Jonathan Masci, Emanuele Rodolà, and Michael Bronstein. 2016. Learning shape correspondence with anisotropic convolutional neural networks. In *NeurIPS*. 3189–3197.
Alexander M. Bronstein, Michael M. Bronstein, Leonidas J. Guibas, and Maks Ovsjanikov. 2011. Shape Google: Geometric Words and Expressions for Invariant Shape Retrieval. *ACM Trans. Graph.* 30, 1 (2011).
Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. 2017. Geometric deep learning: going beyond Euclidean data. *IEEE Signal Processing Magazine* 34, 4 (2017), 18–42.
Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann Lecun. 2014. Spectral networks and locally connected networks on graphs. In *ICLR*.
Taco Cohen, Maurice Weiler, Berkay Kicanaoglu, and Max Welling. 2019. Gauge Equivariant Convolutional Networks and the Icosahedral CNN. In *ICML*, Vol. 97.

1321–1330.

Taco Cohen and Max Welling. 2016. Group equivariant convolutional networks. In *ICML*. 2990–2999.

Taco S. Cohen, Mario Geiger, Jonas Köhler, and Max Welling. 2018. Spherical CNNs. In *ICLR*.

Taco S. Cohen and Max Welling. 2017. Steerable CNNs. In *ICLR*.

Keenan Crane, Clarisse Weischedel, and Max Wardetzky. 2013. Geodesics in heat: A new approach to computing distance based on heat flow. *ACM Trans. Graph.* 32, 5 (2013), 152.

Pim de Haan, Maurice Weiler, Taco Cohen, and Max Welling. 2020. Gauge Equivariant Mesh CNNs: Anisotropic convolutions on geometric graphs. arXiv:2003.05425 [cs.LG]

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *NeurIPS*. 3844–3852.

Danielle Ezuz, Justin Solomon, Vladimir G. Kim, and Mirela Ben-Chen. 2017. GWCNN: A Metric Alignment Layer for Deep Shape Analysis. *Comput. Graph. Forum* 36, 5 (2017), 49–57.

Beat Fasel and Daniel Gatica-Perez. 2006. Rotation-invariant neoperceptron. In *ICPR*, Vol. 3. IEEE, 336–339.

Matthias Fey and Jan E. Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

Matthias Fey, Jan Eric Lenssen, Frank Weichert, and Heinrich Müller. 2018. SplineCNN: Fast Geometric Deep Learning With Continuous B-Spline Kernels. In *CVPR*. IEEE, 869–877.

Chamberlain Fong. 2015. Analytical Methods for Squaring the Disc. arXiv:1509.06344 [math.HO]

William T. Freeman and Edward H Adelson. 1991. The design and use of steerable filters. *IEEE Transactions on Pattern Analysis & Machine Intelligence* (1991), 891–906.

Paul Guerrero, Yanir Kleiman, Maks Ovsjanikov, and Niloy J. Mitra. 2018. PCPNet Learning Local Shape Properties from Raw Point Clouds. *Comput. Graph. Forum* 37, 2 (2018), 75–85.

Niv Haim, Nimrod Segol, Heli Ben-Hamu, Haggai Maron, and Yaron Lipman. 2019. Surface Networks via General Covers. In *ICCV*. IEEE, 632–641.

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 (2019), 90.

Philipp Herholz and Marc Alexa. 2019. Efficient Computation of Smoothed Exponential Maps. *Comput. Graph. Forum* 38, 6 (2019), 79–90.

Geoffrey E Hinton, Alex Krizhevsky, and Sida D Wang. 2011. Transforming auto-encoders. In *International Conference on Artificial Neural Networks*. Springer, 44–51.

Jingwei Huang, Haotian Zhang, Li Yi, Thomas A. Funkhouser, Matthias Nießner, and Leonidas J. Guibas. 2019. TextureNet: Consistent Local Parametrizations for Learning From High-Resolution Signals on Meshes. In *CVPR*. IEEE, 4440–4449.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR*.

Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*.

Risi Kondor, Zhen Lin, and Shubhendu Trivedi. 2018. Clebsch-Gordan Nets: a Fully Fourier Space Spherical Convolutional Neural Network. In *NeurIPS 2018*. 10138–10147.

Ilya Kostrikov, Zhongshi Jiang, Daniele Panozzo, Denis Zorin, and Joan Bruna. 2018. Surface networks. In *CVPR*. IEEE, 2540–2548.

Dmitry Laptev, Nikolay Savinov, Joachim M Buhmann, and Marc Pollefeys. 2016. TI-POOLING: Transformation-invariant pooling for feature learning in convolutional neural networks. In *CVPR*. IEEE, 289–297.

Hugo Larochelle, Dumitru Erhan, Aaron Courville, James Bergstra, and Yoshua Bengio. 2007. An empirical evaluation of deep architectures on problems with many factors of variation. In *ICML*. ACM, 473–480.

Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. 2018. PointCNN: Convolution On X-Transformed Points. In *NeurIPS*. 820–830.

Zhouhui Lian, Afzal Godil, Benjamin Bustos, Mohamed Daoudi, Jeroen Hermans, Shun Kawamura, Yukinori Kurita, Guillaume Lavoué, Hien Nguyen, Ryutarou Ohbuchi, Yuki Ohkita, Yuya Ohishi, Fatih Porikli, Martin Reuter, Ivan Sipiran, Dirk Smeets, Paul Suetens, Hedi Tabia, and Dirk Vandermeulen. 2011. SHREC '11 Track: Shape Retrieval on Non-rigid 3D Watertight Meshes. *Eurographics Workshop on 3D Object Retrieval*, 79–88.

Or Litany, Alexander M. Bronstein, Michael M. Bronstein, and Ameesh Makadia. 2018. Deformable Shape Completion with Graph Convolutional Autoencoders. In *CVPR*. IEEE, 1886–1895.

Kun Liu, Qing Wang, Wolfgang Driever, and Olaf Ronneberger. 2012. 2d/3d rotation-invariant detection using equivariant filters and kernel weighted mapping. In *CVPR*. IEEE, 917–924.

Diego Marcos, Michele Volpi, and Devis Tuia. 2016. Learning rotation invariant convolutional filters for texture classification. In *ICPR*. IEEE, 2012–2017.

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 *ICCV*. 37–45.

Eivind Lyche Melvær and Martin Reimers. 2012. Geodesic polar coordinates on polygonal meshes. In *Comput. Graph. Forum*, Vol. 31. 2423–2435.

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 *CVPR*, Vol. 1. IEEE, 3.

Hao Pan, Shilin Liu, Yang Liu, and Xin Tong. 2018. Convolutional Neural Networks on 3D Surfaces Using Parallel Frames. *arXiv: 1808.04952* (2018).

Adrien Poulenard and Maks Ovsjanikov. 2018. Multi-directional geodesic neural networks via equivariant convolution. *ACM Trans. Graph.* 37, 6 (2018), 236:1–236:14.

Adrien Poulenard, Marie-Julie Rakotosaona, Yann Ponty, and Maks Ovsjanikov. 2019. Effective Rotation-Invariant Point CNN with Spherical Harmonics Kernels. In *3DV*. 47–56.

Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. 2017a. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *CVPR*. IEEE, 77–85.

Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. 2017b. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *NeurIPS*. 5099–5108.

Anurag Ranjan, Timo Bolkart, Soubhik Sanyal, and Michael J. Black. 2018. Generating 3D faces using Convolutional Mesh Autoencoders. In *ECCV*. 725–741.

Ryan Schmidt, Cindy Grimm, and Brian Wyvill. 2006. Interactive decal compositing with discrete exponential maps. In *ACM Trans. Graph.*, Vol. 25. 605–613.

Ryan Schmidt and Karan Singh. 2010. Meshmixer: an interface for rapid mesh composition. In *ACM SIGGRAPH 2010 Talks*.

Stefan C Schonsheck, Bin Dong, and Rongjie Lai. 2018. Parallel Transport Convolution: A New Tool for Convolutional Neural Networks on Manifolds. *arXiv:1805.07857* (2018).

Nicholas Sharp. 2019. Polyscope. www.polyscope.run.

Nicholas Sharp, Keenan Crane, et al. 2019a. geometry-central. www.geometry-central.net.

Nicholas Sharp, Yousuf Soliman, and Keenan Crane. 2019b. The Vector Heat Method. *ACM Trans. Graph.* 38, 3 (2019).

Ayan Sinha, Jing Bai, and Karthik Ramani. 2016. Deep Learning 3D Shape Surfaces Using Geometry Images. In *ECCV*. 223–240.

Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik G. Learned-Miller. 2015. Multi-view Convolutional Neural Networks for 3D Shape Recognition. In *ICCV*. IEEE, 945–953.

Zhiyu Sun, Jia Lu, and Stephen Baek. 2018. ZerNet: Convolutional Neural Networks on Arbitrary Surfaces via Zernike Local Tangent Space Estimation. *arXiv:1812.01082* (2018).

Nathaniel Thomas, Tess Smidt, Steven M. Kearnes, Lusann Yang, Li Li, Kai Kohlhoff, and Patrick Riley. 2018. Tensor Field Networks: Rotation- and Translation-Equivariant Neural Networks for 3D Point Clouds. *arXiv:1802.08219* (2018).

Nitika Verma, Edmond Boyer, and Jakob Verbeek. 2018. FeaStNet: Feature-Steered Graph Convolutions for 3D Shape Analysis. In *CVPR*. IEEE, 2598–2606.

Daniel Vlasic, Ilya Baran, Wojciech Matusik, and Jovan Popovic. 2008. Articulated Mesh Animation from Multi-View Silhouettes. *ACM Trans. Graph.* 27, 3 (2008).

Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. 2019. Dynamic Graph CNN for Learning on Point Clouds. *ACM Trans. Graph.* 38, 5 (2019), 146:1–146:12.

Max Wardetzky, Miklós Bergou, David Harmon, Denis Zorin, and Eitan Grinspun. 2007. Discrete quadratic curvature energies. *Computer Aided Geometric Design* 24, 8-9 (2007), 499–518.

Maurice Weiler and Gabriele Cesa. 2019. General E(2)-Equivariant Steerable CNNs. In *NeurIPS*. 14334–14345. http://papers.nips.cc/paper/9580-general-e2-equivariant-steerable-cnns

Daniel E Worrall, Stephan J Garbin, Daniyar Turmukhambetov, and Gabriel J Brostow. 2017. Harmonic networks: Deep translation and rotation equivariance. In *CVPR*. IEEE, 5028–5037.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. 2019. A Comprehensive Survey on Graph Neural Networks. *arXiv:1901.00596* (2019).

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 *CVPR*. IEEE, 1912–1920.

Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. 2018. Graph Neural Networks: A Review of Methods and Applications. *arXiv:abs/1812.08434* (2018).

Received January 2020; revised April 2020; accepted April 2020