Harmonic Surface Networks

MSc Computer Science

Ruben Wiersma

October 2019

TU Delft

HARMONIC SURFACE NETWORKS

A thesis submitted to the Delft University of Technology in partial fulfillment
of the requirements for the degree of

Master of Science in Computer Science

by

Ruben Wiersma

October 2019

Ruben Wiersma: *Harmonic Surface Networks* (2019)

The work in this thesis was made in the:

Computer Graphics and Visualisation
Department of Computer Science
Faculty of EEMCS
Delft University of Technology

Supervisors: Dr. Klaus Hildebrandt
Prof. dr. Elmar Eisemann

ABSTRACT

We present a new approach for deep learning on surfaces, combining geometric convolutional networks with rotationally equivariant networks. Existing work either learns rotationally invariant filters, or learns filters in the tangent plane without correctly relating orientations between different tangent planes (orientation ambiguity). We propose a solution to both problems by applying Harmonic Networks on surfaces in the tangent plane: Harmonic Surface Networks (HSN).

Harmonic Networks constrain their filters to circular harmonics, which output complex-valued, rotatable feature maps. Considering these complex features as vectors inside the tangent plane, we can use parallel transport along shortest geodesics to transport them along the surface in a natural way. Additionally, Harmonic Networks can be configured so that the output is rotationally invariant, while containing rotationally equivariant filters in hidden layers. This property solves the orientation ambiguity problem, while learning directional filters. We evaluate HSN on three different problems: classification on Rotated MNIST in a plane and mapped to a sphere, correspondence on FAUST, and shape segmentation on FAUST. The results suggest that HSN could improve on state of the art approaches.

ACKNOWLEDGEMENTS

Thanks to all who supported me in completing this thesis. Special thanks to my thesis advisor, dr. Klaus Hildebrandt, for his indispensable guidance and countless insightful discussions on geometry; prof. dr. Elmar Eisemann, for his critical remarks and committed mind; Gosia Migut and Remi van der Laan, for their sharp proofreading; and Mathias Fey, Nicholas Sharp, and Thomas Kipf for their amicable support and understandable code.

And last but not least: a huge thanks to my parents, my brothers and sisters, and Joanne who supported me throughout and helped me breathe from time to time.

CONTENTS

ACRONYMS

ACNN	Anisotropic Convolutional Neural Network (Boscaini et al. [2016])	8
CNN	Convolutional Neural Network	1
exp-map	Exponential Map	2
FeastNet	Feature Steered Network (Verma et al. [2018])	9
FPS	Farthest Point Sampling	15
GCN	Graph Convolutional Network (Kipf and Welling [2016])	8
GCNN	Geodesic Convolutional Neural Network (Masci et al. [2015])	8
GDL	Geometric Deep Learning	1
HSN	Harmonic Surface Networks	v
log-map	Logarithmic Map	
MDGCNN	Multi-Directional GCNN (Poulenard and Ovsjanikov [2018])	11
MoNet	Gaussian Mixture Model Network (Monti et al. [2017])	8
VHM	Vector Heat method	6

1 INTRODUCTION

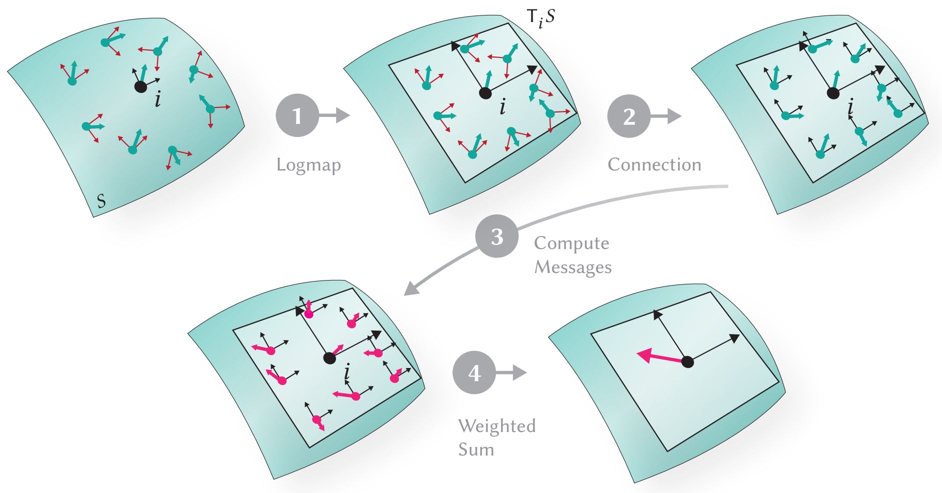


Figure 1.1: An overview of HSN: (1) Map neighbouring points to the tangent plane using the logarithmic map. This introduces orientation ambiguity. (2) Align bases with connection. (3) Apply circular harmonic filters to compute messages for each neighbour. (4) Aggregate messages using a weighted sum. The results is a rotatable complex feature.

The success of Convolutional Neural Networks (CNNs) for learning from images and other data on a Euclidean grid has inspired an effort to generalise CNNs to graphs and manifolds. This effort is referred to as Geometric Deep Learning (GDL) (Bronstein et al. [2017]). GDL unlocks a host of new applications for deep learning: from graph problems and recommender systems, to 3D shape analysis and generation.

In this thesis, we concern ourselves specifically with convolutional networks on surfaces: two-dimensional Riemannian manifolds, embedded in a three-dimensional Euclidean space. CNNs on surfaces have many applications in, for example, medical visualisation, automated driving, and the entertainment industry. Tasks that have already benefited from such methods are correspondence, shape segmentation, object retrieval, and shape generation. We aim to increase the accuracy of convolutional networks on surfaces across most, if not all, of these tasks. To that end, we present Harmonic Surface Networks (HSN) (Figure 1.1).

The goal for GDL on surfaces is to learn a function that maps a signal on a surface $\mathcal{S}$ to a prediction, given a set of examples. Surfaces are typically processed as point clouds or meshes. A point cloud consists of an unordered set of points on the surface. A mesh consists of vertices representing points in space, edges between these vertices, and faces, typically triangles, discretising the surface. The signal on the surface at point i, x_i, can consist of colour values, xyz-coordinates, or shape descriptors, depending on the task.

1.1 GEOMETRIC DEEP LEARNING

As we move from images to surfaces, we are presented with an unordered set of points. This set of points is typically sampled irregularly. Moreover, we have no global coordinate system on the surface. These properties turn out to pose significant challenges which have been faced by previous work. This work can be organised in two main approaches: a graph-based approach, and a spatial approach.

1.1.1 *Graph Approaches*

Graph-based approaches consider the points on a surface as nodes in a graph. Nodes that are within a radius ϵ of each other are connected with edges:
$E = \{(v_i, v_j) | \|v_j - v_i\|_2 < \epsilon\}$. We apply a local 'filter' on the neighbourhood $\mathcal{N}_i$ of each node i: a learned function is applied to the neighbours' feature vectors, which we will refer to as messages. These messages are aggregated using a maximum, average, or sum operator and combined with the feature vector at node i using a learned function (Figure 1.2, Qi et al. [2017a]; Wang et al. [2018]; Gilmer et al. [2017]). By performing these action on all nodes in the graph, we 'convolve' the filter over the input.

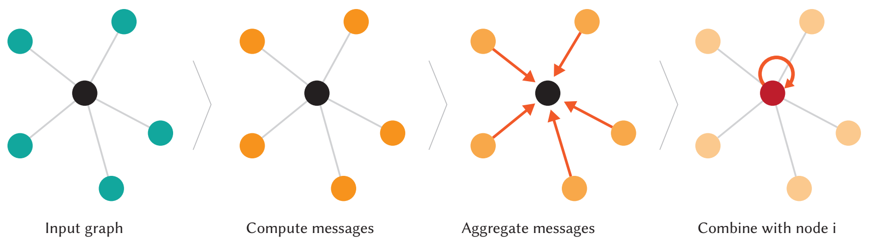


Figure 1.2: Message passing on graphs.

A downside of these methods is that they apply the same message function to each neighbour, irregardless of their orientation to point i. That means we cannot learn filters which detect directional patterns, like edges.

This poses a problem for learning on surfaces. Consider the following illustration (Figure 1.3): when you imagine a simple chair, you are most likely to think of a horizontal plane (the seat) supported by four vertical cylinders (its legs) and a vertical plane attached to the seat (the backrest). Without these directions and relations, the chair might as well be a picket fence or any other object consisting of four cylinders and planes.

1.1.2 *Spatial Approaches*

Unlike graph-based methods, spatial methods can find directional patterns. They learn directional filters in the tangent plane and apply them to points on the surface using the Exponential Map (exp-map) (Figure 1.4a). Spatial methods cope with the irregular sampling of points on the surface by placing a set of Gaussian kernels at fixed or learned locations in the tangent plane and convolving these kernels with the neighbouring points (Boscaini et al. [2016]; Monti et al. [2017]; Masci et al. [2015]).

To apply directional filters consistently, spatial methods have to choose an orientation within the tangent plane using some heuristic, as there is no global orienta-

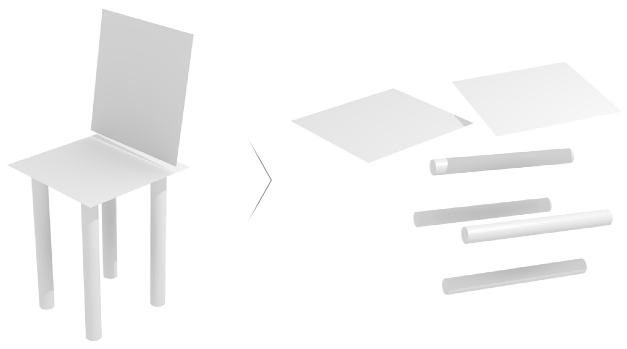
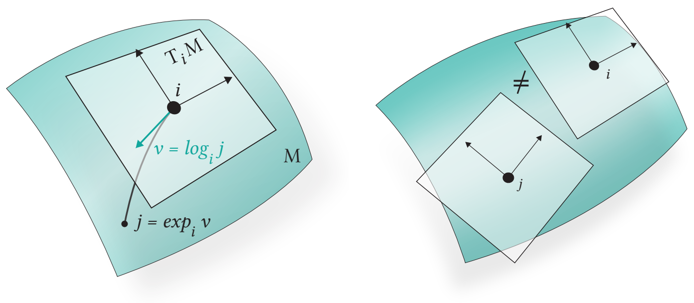


Figure 1.3: Directions are important to recognise shapes.



(a) log-map and exp-map. (b) Orientation ambiguity.

Figure 1.4: Spatial methods learn filters in the tangent plane, using the log-map. They do not yet account for orientation ambiguity.

tion on the surface. Existing spatial methods either choose the maximum curvature direction or apply the filters at multiple directions in the tangent plane and retain the rotation that yields the highest activation. This creates a problem we will refer to as *orientation ambiguity*: at different locations, filters are applied in different orientations (Figure 1.4b). Orientation ambiguity becomes an issue when features propagate through a neural network, because the network combines and relates features that were computed at differing orientations. This is even more problematic when pooling layers are used, because they extend the area that is covered by filters in subsequent layers.

We expect that orientation ambiguity decreases the accuracy of spatial methods, illustrated by the chair example: if a human can hardly recognise a chair from its unaligned components, how could we expect a neural network to do much better?

In this thesis, we aim to solve orientation ambiguity by applying a special kind of convolutional networks, originally designed for images to enable 2D rotational equivariance, called Harmonic Networks.

1.2 HARMONIC NETWORKS

CNNs for images exhibit a property called translational equivariance: if we translate an input image and then compute the feature map, it is equivalent to first computing the feature map and then translating the output of the feature map (Figure 1.5). This is a quintessential ingredient for CNNs' success: it enables the recognition of patterns, independent of their location in the input.

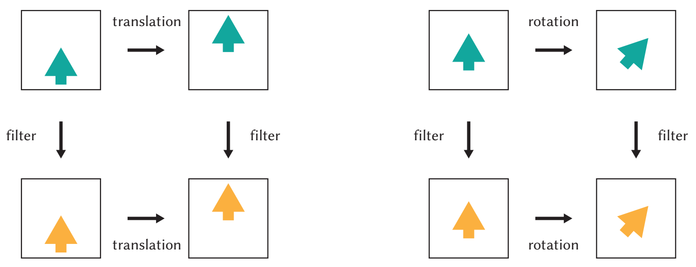


Figure 1.5: Translational equivariance (left) and rotational equivariance (right).

The equivariance property does not hold for rotation in regular CNNs: if we rotate the input and compute a feature map, it is not the same as first computing a feature map and then rotating the output (see Figure 1.5). This inhibits the generalisability of CNNs to different orientations. With Harmonic Networks, Worrall et al. [2017] present a way to guarantee rotational equivariance, by using filters constrained to the family of circular harmonics:

$$\mathbf{W}_m(r, \theta, R, \beta) = R(r)e^{\imath m\theta + \beta}, \tag{1.1}$$

where $R : \mathbb{R}_+ \to \mathbb{R}$ is a learned radial profile, $\beta \in [0, 2\pi)$ is a learned radial offset, and $m \in \mathbb{Z}$ is the rotation order of the circular harmonic. The filter is defined continuously, for all polar coordinates r and θ, and outputs a complex-valued feature vector.

1.2.1 *Rotational Equivariance*

When we rotate an input $\mathbf{X}$ by ϕ and then cross-correlate with a circular harmonic filter, it is equivalent to first convolving the filter and then rotating the output by an angle ϕ:

$$[\mathbf{W}_m \star \mathbf{X}^\phi] = e^{\imath m\phi}[\mathbf{W}_m \star \mathbf{X}], \tag{1.2}$$

where $\mathbf{X}^\phi$ denotes the input $\mathbf{X}$ rotated by the angle ϕ.

Note that the type of rotational equivariance is defined by the rotation order m: for $m = 0$, we achieve rotational invariance:

$$[\mathbf{W}_0 \star \mathbf{X}^\phi] = e^{\imath 0\phi}[\mathbf{W}_0 \star \mathbf{X}] = [\mathbf{W}_0 \star \mathbf{X}]. \tag{1.3}$$

For $m = 1$, we achieve linear rotational equivariance:

$$[\mathbf{W}_1 \star \mathbf{X}^\phi] = e^{\imath 1\phi}[\mathbf{W}_1 \star \mathbf{X}]. \tag{1.4}$$

1.2.2 *Streams and M-equivariance*

It is necessary to separate the output of different rotation orders from each other, to retain the rotational equivariance property. Therefore Harmonic Networks separate feature maps in streams for each rotation order (see Figure 1.6).

Another useful property of circular harmonic filters is that chained cross-correlations of rotation orders m_1 and m_2 lead to a result with rotational equivariance $m_1 + m_2$. Thus, to achieve an output that is M-equivariant to the input rotation, Harmonic Networks require that the sum of rotation orders along any path equals M.

The multi-stream architecture enables us to construct a network that is 0-equivariant at the output, i.e. rotationally invariant, while having higher rotation order streams inside the network.

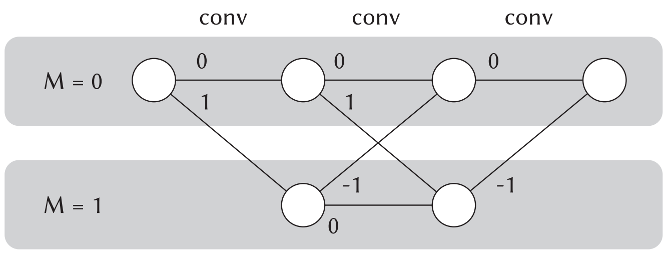


Figure 1.6: In Harmonic Networks, rotation orders are separated in streams.

1.3 HARMONIC NETWORKS IN THE TANGENT PLANE

In this thesis, we aim to apply Harmonic Networks in the tangent plane, to solve the orientation ambiguity problem on surfaces. The benefit is that we can apply a 0-equivariant, i.e. rotationally invariant, network at any orientation in the tangent plane, while guaranteeing the same output. Moreover, inside the network we can learn directional filters, by including streams of rotation orders $M > 0$.

To extend Harmonic Networks to surfaces, we need to face the challenges posed by data on surfaces: it should handle unordered sets of points that are sampled irregularly. To this end, we extend Harmonic Networks to irregular domains and formulate it as a message passing network.

Another challenge is introduced by higher rotation orders *inside* the network: for streams of $M > 0$, the complex features depend on the coordinate system of the tangent plane. If we want to use streams of non-zero rotation orders, we will still encounter the orientation ambiguity problem.

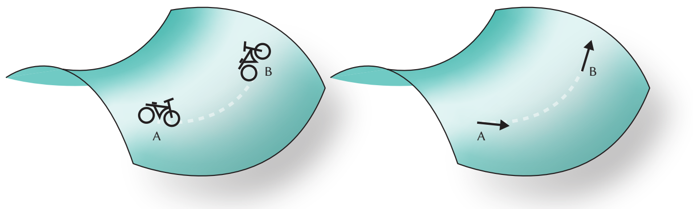


Figure 1.7: Parallel transport: start riding a bike at point A and go to point B without changing your direction.

1.4 PARALLEL TRANSPORT

We use parallel transport along shortest geodesics to approach orientation ambiguity for rotation order streams of $M > 0$. Parallel transport is an intuitive and widely used method to transport vectors along a surface and can be illustrated as follows: Imagine riding a bike along a surface (Figure 1.7). You start riding in a straight line from point A to point B without changing your direction. The vector pointing in your direction at point B is the parallel transport of the vector pointing in your direction at point A.

When we compute parallel transport from point A to point B, we get the direction of the parallel vector in the coordinates of B's tangent basis. On a surface, parallel transport amounts to a 2D rotation, which we will refer to as *connection*. We can simply apply connection to our complex features resulting from harmonic filters, because of the rotational equivariance property.

Without the rotational equivariance property, we could not rotate feature maps without recomputing the convolution for a rotated input. Consequently, convolutional layers would not be able to use outputs from previous layers and thus we would not be able to build neural networks.

Computing global parallel transport along shortest geodesics is a difficult problem, as it involves computing shortest paths and then transporting vectors along these paths. Recently, Sharp et al. [2019c] introduced a new approach to globally approximate parallel transport with vector heat diffusion using the connection Laplacian: the Vector Heat method (VHM). Because the VHM only depends on a set of Laplacian matrices, it can be used on any representation of a surface with a Laplacian and connection Laplacian, including meshes and point clouds. This allows our method to generalise to polygon meshes, point clouds, and voxels as well.

1.5 MAIN CONTRIBUTIONS

Our main contribution is the following: we present a solution to the orientation ambiguity problem that combines geometric convolutional networks and equivariant networks, called Harmonic Surface Networks (HSN). In doing so, we make two technical contributions:

- We introduce a concept for vector-valued convolutional networks on surfaces.
- We extend Harmonic Networks to irregular domains.

With HSN, we can create networks that accurately learn from signals on surfaces, independent of the choice of orientation at each tangent plane. We have found that such networks yield higher accuracy, compared to rotationally invariant graph-based methods, and expect to see improvements on spatial methods with the orientation ambiguity problem.

In the following chapters we survey related work, provide the necessary background, detail our method and implementation, describe our experimental evaluation, and provide a discussion of our results, challenges, and future work.

2

RELATED WORK

This chapter surveys three focus areas to give the reader an overview of the state of the art: geometric deep learning on manifolds, learning with parallel transport, and rotational equivariance for regular CNNs.

2.1 GEOMETRIC DEEP LEARNING ON MANIFOLDS

Our thesis builds on concepts introduced in the field of Geometric Deep Learning (GDL). As 'regular' CNNs gained traction, researchers have attempted to generalise concepts from CNNs to graphs and manifolds. Graphs and manifolds lack a number of properties that regular CNNs build on for learning (Table 2.1), which introduces the need for a new formulation of convolution on graphs and manifolds. In this section, we provide a brief overview of a number of these approaches and discuss how these methods relate to the current work. For a wider survey on the topic, please refer to Bronstein et al. [2017].

Euclidean grid *e.g. images, audio*	**Graphs and manifolds**
Natural ordering *e.g. from left to right*	Unordered
Regular grid *e.g. pixel grid or regular sampling*	Irregularly sampled
Grid used for hierarchy	Clustering used for hierarchy

Table 2.1: Properties of Euclidean data on a grid vs. graphs and manifolds

We will structure our discussion of geometric deep learning in two major streams: spectral methods and spatial methods. As the field progressed, these two streams have merged into the current paradigm of message passing. The message passing paradigm will be detailed in the background section, as this will inform how our convolution operator is defined.

GRAPHS The survey will treat a number of methods that were primarily designed for graphs. Consequently, most of the described works do not directly apply their method on manifolds. We have included these works in the survey as they have informed the design of recent convolutional networks designed for manifolds. Additionally, many of the graph-based methods can be directly applied to manifolds by either building a radius- or k-nearest neighbour graph from the points on the manifold or, if a mesh is available, parsing the mesh as a graph.

2.1.1 *Spectral*

In the spectral domain, convolution of a filter over a signal can be performed with a simple multiplication. The challenge to define convolution for graphs is then shifted to that of transforming a signal on a graph to the spectral domain efficiently.

The idea to use spectral convolution for neural networks on graphs was first introduced in Spectral Networks (Bruna et al. [2014]). Given the graph Laplacian Δ, one can find the bases of the spectral domain by computing the eigenvalue decomposition of $\Delta = \mathbf{\Phi}\mathbf{\Lambda}\mathbf{\Phi}^\mathsf{T}$. The eigenvectors contained in the columns of $\mathbf{\Phi}$ constitute the bases of the spectral domain. The signal is projected onto the these eigenvectors, so that the filter can be applied in the spectral domain and afterwards projected back to the original domain.

The eigenvalue decomposition is a costly operation, especially when it has to be performed multiple times when the graph structure changes due to pooling. To avoid the cost of eigenvalue decomposition, ChebNet (Defferrard et al. [2016]) approximates diffusion using the Chebyshev polynomial. Graph Convolutional Network (Kipf and Welling [2016]) (GCN) further simplifies this design to a 1-hop approximation of the Chebyshev polynomial, resulting in the following simple definition for convolution:

$$\mathbf{X}^{(l+1)} = \sigma\left(\mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}\mathbf{X}^{(l)}\mathbf{W}^{(l)}\right), \tag{2.1}$$

where $\mathbf{X}$ and $\mathbf{X}'$ are the features in the previous and next layer in the network, $\mathbf{A}$ is the adjacency matrix of the input graph, $\mathbf{D}$ is the degree matrix $\mathbf{D}_{ii} = \sum_j \mathbf{A}_{ij}$, and $\mathbf{W}$ is a trainable weight matrix. Often, this formulation is rewritten to give the response per point:

$$\mathbf{x}_i^{(l+1)} = \sigma\left(\mathbf{x}_i^{(l)}\mathbf{W}_0^{(l)} + \sum_{j\in\mathcal{N}} \frac{1}{c_{ij}}\mathbf{x}_j^{(l)}\mathbf{W}_1^{(l)}\right). \tag{2.2}$$

This formulation is fast and effective and has found widespread adoption due to its simplicity. Another benefit of GCN versus the pure spectral methods is that the filters are learned on a local neighbourhood of each node, whereas the filters learned through the spectral transforms depend on the Laplacian for the entire graph, which makes it impossible to transfer learned weights to new graph topologies.

The local support of the filters exhibited by the GCN approach is a nice segue into our next section: One can regard the filter in this convolution as a local template, or a patch, moving across the input. Such a template detects patterns through cross correlation. The template paradigm is exactly the interpretation of CNNs that is explored by spatial approaches.

2.1.2 *Spatial*

When trying to mimic the 'local template' approach, we run into the issues outlined at the beginning of this section: the input points are unordered, irregularly sampled, and, additionally, they lack a meaningful global parametrisation: In CNNs for images, we could define a $k \times k$ patch consisting of a regular grid that maps directly to the structure of the input. Locality in the input tensor was directly related to spatial locality. For graphs and manifolds, our data is unordered. Therefore, we will need to define a local template, or patch, that is explicitly based on the spatial coordinates of each point. This was applied by Geodesic Convolutional Neural Network (Masci et al. [2015]) (GCNN), Anisotropic Convolutional Neural Network (Boscaini et al. [2016]) (ACNN), and Gaussian Mixture Model Network (Monti et al. [2017]) (MoNet).

First, locality. For graphs and manifolds, the neighbourhood is defined by the r-hop neighbourhood around each point, or a radius ball around each point, respectively. The r-hop neighbourhood is the set of points that can be reached within r hops along the graph structure and a radius ball defines the region of space within a certain distance, the radius, around each point.

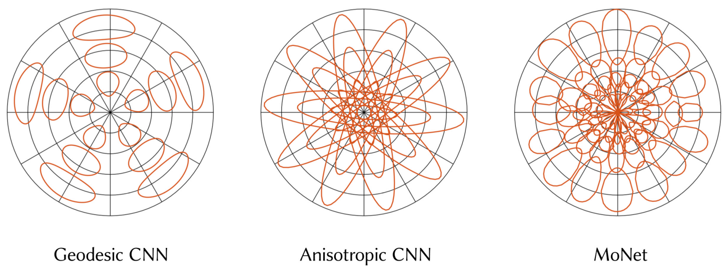


Figure 2.1: Kernels on local patches used in GCNN, ACNN, and MoNet.

Next, local coordinates. For graphs, one can use the degree of each node, as was done by GCN. For manifolds, it is typical to use the logarithmic map to define polar coordinates in the tangent plane for each neighbour (GCNN, ACNN, MoNet).

Finally, the network can only learn a limited amount parameters to represent a filter and our input data is irregularly sampled. GCNN solves this problem by placing k Gaussian kernels at regular intervals in the local patch (Figure 2.1). The product between these kernels and the irregular points is computed to get a new (interpolated) value for each kernel. Next, a filter is applied to each kernel and the resulting values are summed to retrieve the activation in the next layer. ACNN takes a slightly different approach, by using anisotropic kernels instead of Gaussian kernels, hence the name anisotropic CNN.

Our method (HSN) takes a different approach for learning limited parameters for irregular samplings: we parametrise a continuous function, which is evaluated at each point on the surface, instead of learning a value for each location directly. Because HSN evaluates the continuous function at each input point, it avoids the need to set hyperparameters for the number of kernels or locations of kernels, simplifying the construction and optimisation of deep networks.

MoNet and FeastNet

MoNet further generalised and parametrised the Gaussian kernels in the patch. Instead of placing the kernels in pre-defined locations, MoNet also learns the mean and variance of each kernel to increase the ability of the network to fit the data, thereby reaching state of the art performance. Since MoNet allows for any definition of pseudo-coordinates, it can be applied to graph learning problems as well as manifold learning problems.

The abstract definition of pseudo-coordinates gave way for Feature Steered Network (Verma et al. [2018]) (FeastNet) to define pseudo-coordinates as a learned combination of features from the previous layer. This allows the network to optimise even further.

HSN deviates from these last two works, as it explicitly requires polar coordinates. Furthermore, HSN parametrises continuous functions that are evaluated at each input point.

Orientation Ambiguity

A final distinction between spatial methods and this thesis is the orientation of the filter in the tangent plane. GCNN, ACNN and MoNet all use filters that are rotationally variant; the orientation of the filter in the tangent plane influences the output. To achieve consistency across different inputs, GCNN and MoNet compute the filter response on multiple orientations and apply angular max-pooling. ACNN applies the filter in the maximum curvature direction.

Both variations pose a problem that has long gone unsolved, which we refer to as *orientation ambiguity*: filters at different locations use different orientations. Consequently, these networks cannot correctly relate the direction of patterns at different locations, which is likely to influence their performance. Additionally, we expect that angular max-pooling (used by GCNN and MoNet) negatively impacts the stability of training, as the same filter can be applied in different directions for each training iteration.

HSN simplifies the choice of orientation within the tangent plane: 0-equivariant HSN networks result in the same output for any orientation of the tangent plane. Thus, HSN does not require a consistent choice of orientation, further simplifying the optimisation of convolutional networks for surfaces. By using parallel transport on its feature vectors, HSN also accounts for orientation ambiguity between neighbouring points. Overall, HSN provides a simpler and more accurate approach to orientation within the tangent plane.

Other Manifold Approaches

Before our discussion moves to a generalisation covering both spectral and spatial methods, we will cover a number of approaches related to the spatial approach that are typically not included in discussions on geometric deep learning, even though they share many similar problems and solutions. In contrast to MoNet, GCNN, and ACNN, they do not have orientation ambiguity problem.

TORIC COVERS Maron et al. [2017] used a global parametrisation by mapping the input manifold to a toric cover, on which a standard convolution can be computed. Because a global parametrisation is used, the orientation ambiguity problem is dealt with. The limitations of this method are that toric covers are only defined for objects with a sphere-like topology and that the toric cover is dependent on three reference points on the surface. Thus, the learned filters are dependent on the choice of reference points. Additionally, the global cover introduces undesired deformations (e.g. stretching, squeezing) of the surface.

POINTNET AND POINTNET++ PointNet was designed by Qi et al. [2017a] to solve the first problem we encountered: our input is an unordered set of points. First, a neighbourhood is constructed around each point with a radius ball. To retrieve a response for point i, a function is applied to each neighbour of i and the maximum activation across i's neighbours is stored as the new response for i:

$$\mathbf{x}_i^{(l+1)} = \gamma_\Theta \left(\max_{j \in \mathcal{N}(i) \cup \{i\}} h_\Theta(\mathbf{x}_j^{(l)}, \mathbf{p}_j - \mathbf{p}_i) \right). \tag{2.3}$$

This simple formulation is fast and effective for many problems. Qi et al. [2017b] designed PointNet++ as an extension of PointNet with hierarchical functionality. Because PointNet applies the same filter to each neighbour, it is effectively rotationally invariant, and thus does not have the orientation amgibuity problem.

EDGECONV The EdgeConv convolution proposed by Wang et al. [2018] is similar to PointNet. EdgeConv provides additional information to the network by concatenating i's feature vector to its neighbours' feature vectors before feeding it to the learned function and by using a sum operation to aggregate the neighbours' responses instead of a max operation:

$$\mathbf{x}_i^{(l+1)} = \sum_{j \in \mathcal{N}(i)} h_\Theta(\mathbf{x}_i^{(l)} \parallel \mathbf{x}_j^{(l)} - \mathbf{x}_i^{(l)}). \tag{2.4}$$

Additionally, the Wang et al. [2018] propose reconstructing the neighbourhood graph dynamically within the network. This should allow the network to construct and learn from semantic neighbourhoods, instead of mere spatial neighbourhoods.

PointNet and EdgeConv exhibit a familiar structure, one we have seen for GCN as well. Each of these networks applies a learned filter to neighbourhoods and aggregates these values with some aggregator function. In the next section, we will discuss a paradigm that is able to cover all of the previously mentioned approaches.

2.1.3 *Message Passing*

We noticed an overlap between the spectral and spatial approaches. A similarity with another paradigm was observed by Gilmer et al. [2017], who describe message passing networks for learning on molecular structures. After Gilmer et al. [2017], many of the previously discussed methods have been reinterpreted as message passing networks (Fey and Lenssen [2019]). The basic idea of message passing networks is the following: to compute the response for a point i, we consider the features of neighbouring points $j \in \mathcal{N}(i)$ and the point itself. First, a learned function is applied to the feature vectors of the neighbouring points, resulting in a message for each neighbour. These message are then aggregated using some aggregator function (e.g. sum, max, mean) and combined with the feature vector of point i.

2.2 LEARNING WITH PARALLEL TRANSPORT

As described in the introduction and in section Section 2.1.2, current methods either learn filters that are rotationally invariant, or choose an orientation within the tangent plane using some heuristic. None of these methods describe how to deal with orientation ambiguity. Recently, some other works have applied parallel transport with direct connection to solve orientation ambiguity.

Parallel transport, a concept from differential geometry, is a method to transport a vector along the geodesic of a manifold to another location on the manifold, such that the vector is 'parallel' to its original location. One of its applications is vector diffusion (Singer and Wu [2012]), which is analogous to the process performed by message passing networks: in diffusion, a point is iteratively updated with the average of its neighbours, mirroring the aggregation step in message passing networks. For vector diffusion, one needs to average the neighbours' vectors, which requires the vectors to be transformed to the same tangent plane.

One application of parallel transport outside of GDL is the approximation of the exponential map by Sharp et al. [2019c]. Another application is that of parallel transport unfolding of a manifold by Budninskiy et al. [2019].

2.2.1 *Parallel Transport for Geometric Deep Learning*

In GDL, parallel transport has been applied by Multi-Directional GCNN (Poulenard and Ovsjanikov [2018]) (MDGCNN). Their contribution is similar to this thesis: using parallel transport along shortest geodesics, one can transport the filter values from neighbours to the tangent plane of the considered point.

MDGCNN follows GCNN's approach of constructing a patch and computing the cross correlation with this patch. Instead of only storing the activation for the maximum rotation, they store the activation for a number of rotations and apply connection by indexing the correct rotation from a point's feature tensor.

HSN differs from MDGCNN in the type of filters that are learned: instead of learning GCNN-like filters, HSN learns circular harmonic filters that output complex values. These complex values can be rotated continuously, where MDGCNN has to interpolate between a limited amount of rotations. This introduces an inaccuracy for each application of parallel transport that grows as the network deepens. Moreover, HSN only has to store one complex value, encoding all these rotations, saving the network a valuable amount of storage space and complexity.

Another work using parallel transport to define convolution on surfaces is Parallel Transport Convolution Nets (Schonsheck et al. [2018]). This work uses parallel transport to convolve a kernel function over a manifold. Similarly, Pan et al. [2018] use parallel frames, instead of parallel transport, to define convolutions on surfaces. In comparison, HSN uses parallel transport to relate complex feature vectors instead of the convolution kernels. Additionally, PTCNets and Pan et al.'s CNNs have no notion of rotation orders or rotationally equivariant filters.

2.3 ROTATIONAL EQUIVARIANCE

HSN builds on Harmonic Networks (Worrall et al. [2017]). This work exists in a field of research on CNNs, where researchers are attempting to train CNNs for images that can generalise over rotations of the input: rotationally equivariant networks. To reach this goal, researchers have taken a number of different approaches:

- Steerable filters (Freeman and Adelson [1991]; Liu et al. [2012]; Cohen and Welling [2016b]),
- hard-baking transformations in CNNs (Cohen and Welling [2016a,b]; Marcos et al. [2016]; Oyallon and Mallat [2015]; Fasel and Gatica-Perez [2006]; Laptev et al. [2016]; Dieleman et al. [2016]), and
- learning generalised transformations (Hinton et al. [2011]).

Most relevant to HSN are steerable filters, since we desire features that are not only rotationally equivariant, but also 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 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 further developed by Thomas et al. [2018] into 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.

HSN continues in the vein of Harmonic Networks, rather than Tensor field networks, because its filters are designed to operate within the 2D tangent space of each point on the surface. HSN extends Harmonic Networks to irregular samplings and applies it to points mapped to the tangent plane.

3 BACKGROUND

In the following section, we will provide the mathematical and technical background required for our problem statement and method. We set off by describing the input and output to our message passing network, followed by theory on geometry, and harmonic networks. In the next chapter, we will explain how these components fit together to build Harmonic Surface Networks (HSN).

3.1 NOTATION

We will explain each variable and parameter as we go through equations. We use the following convention to denote complex numbers: $\sqrt{-1}$ is denoted by $\imath$, to avoid confusion with the indices i and j.

Feature vectors and matrices are in bold. We do not use boldface for complex values or spatial vectors.

3.2 CNNS AND MESSAGE PASSING NETWORKS

Convolutional Neural Networks (CNNs), are neural networks consisting of three types of layers: convolutional layers, pooling layers, and fully connected layers (LeCun et al. [2015], Figure 3.1). Convolutional layers apply filters through convolution, thereby supporting weight sharing. Pooling layers downsample the input by summarising regions using an aggregator operation, like max, min, or average. In CNNs, pooling layers enable translation invariance and learning multi-scale hierarchies. Fully connected layers are conventional neural network layers (or 1x1 convolution layers), which infer the desired result from the features provided by the convolutional and pooling layers. We aim to build a similar architecture of convolutional and pooling layers, but instead of Euclidean data, we provide point clouds or meshes as input.

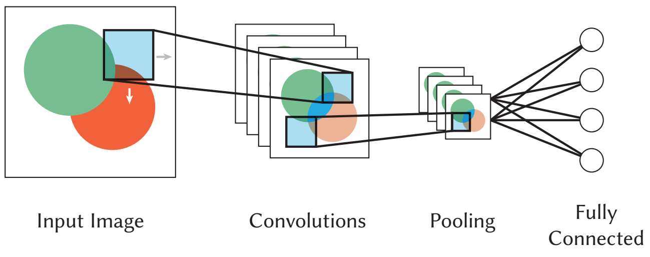


Figure 3.1: A simplified overview of the structure of CNNs.

3.2.1 *Input*

The input to the network is a c-dimensional signal on surface $\mathcal{S}$. The signal can consists of colour values, xyz-coordinates, or shape descriptors. Surface $\mathcal{S}$ is a two-dimensional differentiable manifold. In general, a differentiable manifold $\mathcal{M}$ is a topological space with the property that each point i in the manifold has a neighbourhood that is diffeomorphic to an open ball in d-dimensional Euclidean space. The tangent space at a point i, $T_i\mathcal{M}$ is a d dimensional vector space attached to i. We can think of the tangent space as the vector space of all possible tangent vectors at i of curves in the manifold (Kuehnel [2005]).

HSN is designed to work with any representation of a surface $\mathcal{S}$, irregardless of the discretisation. In this thesis, we limit our explanations to triangle meshes and point clouds.

MESH A mesh is the discretised representation of surface $\mathcal{S}$ and is defined by a set of vertices V, edges between these vertices $E \subseteq V \times V$, and faces, typically triangles: $F \subseteq V \times V \times V$. The following properties hold for surfaces: each interior edge, $\{i, j\} \in E$ is shared by exactly two triangular faces ikj and $jhi \in F$ and boundary edges are part of exactly one triangular face. We can parse the mesh as a graph by using only the vertices and edges.

POINT CLOUD A point cloud consists of an unordered set of points P, where each point is associated with a three-dimensional vector representing its location. The points lie on surface $\mathcal{S}$.

To be able to process a point cloud with a message passing network, a radius-ball graph is computed. The vertices of this graph are the points P and the edges connect points that lie within a given radius r of each other: $E = \{\{i, j\} | \|p_j - p_i\| < r\}$. To work on meshes, we can build a radius-ball graph or simply use the vertices and edges (V, E) from the mesh and discard the set of faces F.

Each vertex in the input graph is associated with the signal on the surface at node i: $\mathbf{x}_i$. The edges in the graph may also be associated with a feature vector, $\mathbf{e}_{ij}$.

3.2.2 *Message Passing Layer*

Most convolutional operations in geometric deep learning can be formulated as message passing layers (Gilmer et al. [2017]; Fey and Lenssen [2019], Figure 3.2). A message passing layer $(l+1)$ takes the feature vector produced by the previous layer (l). To compute a new feature vector for point i, messages are computed for each neighbour with the learned function $\varphi^{(l)}$ and aggregated using an aggregator function $\square$. The aggregation result is combined with $\mathbf{x}_i^{(l)}$ by the learned function $\gamma^{(l)}$. The message passing layer can be formulated as follows:

$$\mathbf{x}_i^{(l+1)} = \gamma^{(l)} \left(\mathbf{x}_i^{(l)}, \square_{j \in \mathcal{N}(i)} \, \varphi^{(l)} \left(\mathbf{x}_i^{(l)}, \mathbf{x}_j^{(l)}, \mathbf{e}_{i,j} \right) \right). \tag{3.1}$$

The new feature vector $\mathbf{x}_i^{(l+1)}$ is of size $c^{(l+1)}$. This is determined by $\gamma^{(l)}$ and $\phi^{(l)}$.

3.2.3 *Pooling*

Pooling for surfaces can either be performed using graph clustering algorithms or by dividing the embedding space. Previous works tend to use the prior method: Spectral Networks (Bruna et al. [2014]) use Naïve Agglomerative clustering, Henaff

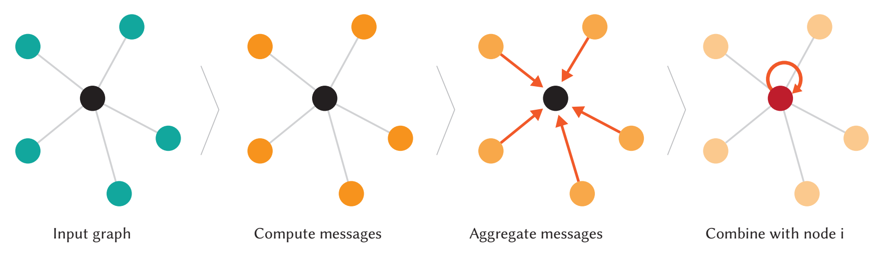


Figure 3.2: An overview of message passing networks.

et al. [2015] opt for Spectral Clustering Von Luxburg [2007], and the technique most used for clustering was first adopted by ChebNet (Defferrard et al. [2016]): Graclus Multi-Level Clustering (Dhillon et al. [2007]). They suggest researching other coarsening techniques as well, like Algebaric Multigrid Techniques (Ron et al. [2011]) and the Kron reduction (Shuman et al. [2016]). The Graclus Multi-Level Clustering technique is also used by MoNet and FeastNet (Monti et al. [2017]; Verma et al. [2018]).

PointNet++ makes use of a technique called Farthest Point Sampling (FPS): FPS repeatedly picks the point that is furthest away from the previously sampled points, until a predefined number of points has been sampled (Qi et al. [2017b]). All non-sampled points are discarded. We will make use FPS, but instead of discarding non-sampled points, we cluster these points to the nearest sampled points.

3.2.4 *Output*

The network can be trained to classify the entire graph, or to classify vertices in the graph. In either setting, a fully connected layer can follow the convolutional and pooling layers.

A **Global classification** is useful for object classification and is derived by a global pooling operation. This operation could be a global average, maximum, or majority vote.

Local classification can be used to segment and label models into semantic (Xu et al. [2016]) or functional (Hu et al. [2018]) parts. Local classification can also be put to use for correspondences in shape analysis. The correspondence problem is then transformed into a labelling problem, where each vertex in one model is labelled with a vertex in another model (Boscaini et al. [2016]; Litany et al. [2017]).

3.3 GEOMETRY

We implement our filters in the tangent plane. For this purpose, we use the Logarithmic Map (log-map). The log-map is the map from a location on the surface to a location in the tangent plane. Moreover, to compute the transformation from coordinates in one tangent plane to those in another tangent plane, we make use of parallel transport along shortest geodesics. We will provide a concise and intuitive description of these concepts. For an in-depth treatment of these subjects, please refer to O'neill [2006], Petersen et al. [2006], or Kuehnel [2005].

3.3.1 *Logarithmic Map*

Spatial geometric deep learning methods use polar coordinates to construct a local patch around each point i on a manifold $\mathcal{M}$. These polar coordinates (r, θ) provide the shortest geodesic from i to a neighbouring point j as the radial coordinate r

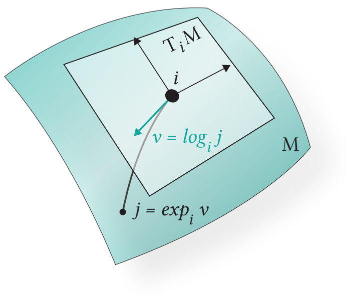


Figure 3.3: The logarithmic map illustrated.

and the direction of that shortest geodesic as the angular coordinate θ, relative to a reference direction $\theta = 0$. This is referred to as the logarithmic map $\log_i(j) = v$, where v is a vector in $T_i\mathcal{M}$ with length r and direction θ, relative to the reference direction (Figure 3.3). One can think of this map as 'unfolding' the curved surface to a flat plane. Most literature on this topic refers to the logarithmic map's inverse, the exponential map: $\exp_i(v) = j$.

Outside of geometric deep learning, the logarithmic map has many useful applications, such as texture decaling (Schmidt et al. [2006]) and interactive shape editing (Schmidt and Singh [2010]).

3.3.2 *Parallel Transport*

Recall the problem of orientation ambiguity: different locations on the surface have different orientations of the tangent plane. This makes it difficult to compare vectors at different locations on a manifold, as their coordinates are defined in different bases. Parallel transport along shortest geodesics provides an intuitive way to compare vectors in different tangent planes. Imagine driving a bike, like our illustration from the introduction (Figure 1.7). If you ride along the surface from j to i without changing your direction, it is sensible to say that your direction at i is parallel to your direction at j. That vector also has same magnitude as the original vector. We will refer to parallel transport along shortest geodesics from j to i as $P_{j \to i}$. If we want to know the parallel vector to x_j in the tangent basis of i, we simply apply parallel transport to x_j: $P_{j \to i}(x_j)$.

On a surface, the tangent plane is a two-dimensional plane. Because parallel transport preserves the magnitude of the vector, we effectively apply a 2D rotation to the each complex value in feature vector $\mathbf{x}_j$. We will refer to this rotation as connection ϕ_{ji}. The knowledge that connection is a 2D rotation can be used when we apply parallel transport to complex features resulting from rotationally equivariant filters, as proposed in Harmonic Networks.

3.3.3 *Computing the Logarithmic Map and Parallel Transport*

Computing parallel transport and the logarithmic map is not as straightforward as it might seem. Nearby points have similar tangent planes, providing a natural way to compute the exponential map by projecting points to the tangent plane. However, the tangent planes of far-away points deviate from each other unpredictably.

A simple approach to compute the exponential map at i for the 1-ring in a mesh is to take the Euclidean distance of the edge between i and its neighbours as r and to either unfold or project these triangles the the tangent plane at i to retrieve the angular coordinate. A similar version of this is used by GCNN and MoNet to compute local exponential maps.

Dijkstra

Schmidt et al. [2006] show how to extend this simple approach to a larger region by using a Dijkstra-like algorithm. The intuition of this approach is as follows: add up the edge lengths along the shortest path from point i to j to retrieve r and consecutively project j to the tangent planes of points along the shortest path to retrieve θ. Another Dijkstra-based approach was proposed by Melvær and Reimers [2012], who reached higher accuracy than the previous work, by inferring geodesic distances to virtual points *within* triangles.

Heat Diffusion

Dijkstra-based methods tend to be quite expensive, motivating research towards close approximations of the logarithmic map (Crane et al. [2013]; Herholz and Alexa [2019]; Sharp et al. [2019c]). The Heat Method (Crane et al. [2013]) uses the observation that the gradients of the heat kernel closely approximate the gradients of the distance field on a surface and exploits this idea to simplify the problem of computing shortest geodesics (the radial coordinate r) to solving sparse linear systems depending on the Laplacian Δ.

The Vector Heat method (VHM) (Sharp et al. [2019c]) extends heat diffusion to vector heat diffusion, using the connection Laplacian Δ^{∇}. This way, parallel transport along shortest geodesics can be computed accurately and globally. Parallel transport along shortest geodesics can then be used to compute the angular component of the logarithmic map θ. Herholz and Alexa [2019] use a similar method to compute the logarithmic map, but do not provide the general theory for vector diffusion.

HSN adopts the Vector Heat method to compute parallel transport and the logarithmic map, because it is the state of the art in computing global and accurate parallel transport. Additionally, because the Vector Heat method only requires a Laplacian and connection Laplacian, we can use it on any surface representation for which these Laplacians are defined. This includes triangle meshes and point clouds, as well as polygonal meshes and voxels.

3.3.4 *Connection for Parallel Transport*

Sharp et al. [2019c] detail how they compute the connection Laplacian for meshes in their paper, but do not provide details for the connection Laplacian on point clouds. In this section, we explain how to compute connection on point clouds, using a method proposed by Singer and Wu [2012] and shortly summarise the Vector Heat method's approach to compute connection on a mesh.

Point Cloud

Given are the orthonormal basis O_i of the tangent plane $T_i\mathcal{S}$ at i and the basis O_j of the tangent plane at j. O_i consists of two column vectors of size three (x, y, z coordinates), spanning $T_i\mathcal{S}$. The column vectors are constructed using local PCA. The connection can be derived from the transformation between O_i and O_j, $O_i^\intercal O_j$. Because of curvature, the subspaces spanned by O_i and O_j are not exactly the same, and thus O_j, $O_i^\intercal O_j$ is not guaranteed to be orthogonal. We therefore define the connection, O_{ij}, as the closest orthogonal matrix:

$$O_{ij} = \text{argmin}_{O \in O(d)} \|O - O_i^\intercal O_j\|_{HS}. \tag{3.2}$$

The minimisation has a simple solution through the SVD of $O_i^\intercal O_j$. Given that the SVD of $O_i^\intercal O_j$ is:

$$O_i^\intercal O_j = U \Sigma V^\intercal, \tag{3.3}$$

the connection is given by:

$$O_{ij} = UV^\intercal, \tag{3.4}$$

which is a two by two rotation matrix.

Mesh

For meshes, we can use another way to compute the connection. Given two neighbouring vertices i and j, we know that the polar coordinates for i to j should be the inverse of the polar coordinates from j to i: $\theta_{ij} = \theta_{ji} + \pi$. Thus, the difference between these values gives us the connection: $\phi_{ij} = (\theta_{ji} + \pi) - \theta_{ij}$ (Figure 3.4).

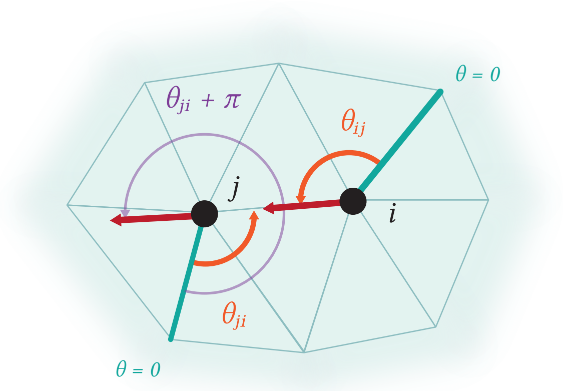


Figure 3.4: Connection can be computed on meshes by comparing θ_{ij} with θ_{ji}.

3.4 HARMONIC NETWORKS

In this thesis, we apply the filters proposed in Harmonic Networks (Worrall et al. [2017]) in the tangent plane. These filters are constrained to the family of circular harmonics, thereby enabling rotational equivariance. A function is rotationally equivariant if we can associate a rotation of the input with an equivalent rotation of the output (Figure 1.5).

Worrall et al. [2017] achieve rotational equivariance by using the following filter definition:

$$\mathbf{W}_m(r, \theta, R, \beta) = R(r) e^{\imath(m\theta + \beta)}, \tag{3.5}$$

where r and θ are the polar coordinates for the point the filter is applied to, $R : \mathbb{R}_+ \to \mathbb{R}$ is the *radial profile*, which controls the shape of the filter, $\beta \in [0, 2\pi)$ is a phase offset term, referred to as the *radial offset*, and $m \in \mathbb{Z}$ is the rotation order of the filter. The output of this filter is a complex value. From hereon, Equation 3.5 will be denoted as $\mathbf{W}_m$.

Given the rotation of feature map $\mathbf{X}$ by ϕ around the origin, denoted as $\mathbf{X}^{\phi} = \mathbf{X}(r, \theta - \phi)$, it follows from the definition of $\mathbf{W}_m$ that[1]:

$$[\mathbf{W}_m \star \mathbf{X}^{\phi}] = e^{\imath m \phi}[\mathbf{W}_m \star \mathbf{X}^{0}]. \tag{3.6}$$

This fulfils the rotational equivariance condition and provides a way to steer the output of convolutional layers, enabling the re-use of activations throughout the network for different rotations of the input.

3.4.1 *m-Equivariance*

Note that the rotation order, m, determines what kind of rotational equivariance is achieved. For $m = 0$, rotational invariance is achieved, since for any rotation ϕ:

$$[\mathbf{W}_0 \star \mathbf{X}^{\phi}] = e^{\imath 0 \phi}[\mathbf{W}_0 \star \mathbf{F}^{0}] = [\mathbf{W}_0 \star \mathbf{F}^{0}]. \tag{3.7}$$

For $m = 1$, linear rotational equivariance is achieved, which is sufficient for parallel transport. We refer to the type of equivariance for rotation order m as m-equivariance.

3.4.2 *Chaining Cross-Correlations*

If we chain cross-correlations of rotation orders m_1 and m_2, we achieve rotational equivariance of order $m_1 + m_2$:

$$[\mathbf{W}_{m_1}[\mathbf{W}_{m_2} \star \mathbf{X}^{\phi}] = e^{\imath m_1 \phi} e^{\imath m_2 \phi}[\mathbf{W}_m \star \mathbf{X}^{0}] \tag{3.8}$$

$$= e^{\imath (m_1 + m_2)\phi}[\mathbf{W}_m \star \mathbf{X}^{0}]. \tag{3.9}$$

3.4.3 *Rotation Order Streams*

It is necessary to separate the outputs of different rotation order convolutions, because the rotational equivariance property breaks when combining different rotation orders. Therefore, Worrall et al. [2017] propose a network architecture structured in separate streams per rotation order (Figure 3.5). For each feature map inside a stream M, we require that the sum of rotation orders m_i along any path reaching that feature map equals M:

$$\sum_{i=1}^{N} m_i = M. \tag{3.10}$$

It is advised to trace the paths in Figure 3.5 to understand what this means.

3.4.4 *M-Equivariant Networks*

With this architecture, we can build networks where the output of the network is M-equivariant to rotations of the input. An obvious application is the construction

1 Harmonic Networks' formulation uses cross-correlation instead of convolution. The authors of Harmonic Networks use cross-correlation as the understanding is easier to follow.

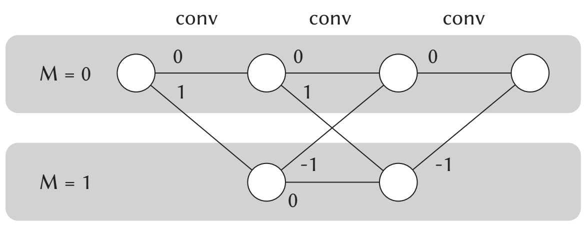


Figure 3.5: Harmonic Networks separate the result of different rotation order convolutions into streams of M-equivariance.

of a 0-equivariant network. Such a network can have streams with $M > 0$, but only outputs the $M = 0$ stream, or a combination of only the magnitudes from each stream. The output of such a network is stable, irregardless of the rotation of the input.

4

METHOD

From the overview on related work and background theory, we point out the following challenges:

1. Many state-of-the-art GDL approaches learn rotationally invariant filters. We expect that this decreases the accuracy of learning on surfaces.

2. Spatial GDL approaches are able to learn directional filters. However, they do not account for the change of basis between points on the surface (orientation ambiguity). We expect that orientation ambiguity negatively impacts accuracy.

We observe the following advantageous properties of Harmonic Networks:

1. Circular harmonic filters allow us to construct rotationally invariant *networks* with more informative rotationally equivariant *streams inside* the network.

2. Circular harmonic filters output complex valued features, which can be rotated.

We propose to apply Harmonic Networks in the tangent plane of surfaces. These networks would be invariant to the rotation of the orientation of the tangent plane, eliminating the need to choose a consistent orientation.

For rotation order streams with $M > 0$ we still need to account for orientation ambiguity, as the features are M-equivariant to any rotation of the input. This is where we apply the second property of Harmonic Networks: we can use parallel transport along shortest geodesics to accurately rotate complex features to other tangent planes.

We expect that the removal of orientation ambiguity, combined with the directional filters in rotation order streams of $M > 0$ will improve the accuracy of learning on surfaces.

In the following sections, we present Harmonic Surface Networks (HSN): a convolutional layer, pooling layer, and set of operations aimed at building more accurate convolutional neural networks for surfaces.

4.1 CONVOLUTION

We start our definition of the equivariant convolution filters within the continuous domain, because the rotational equivariance property is proven in the continuous domain. After the filter is defined in the continuous domain, a discrete approximation of is provided for irregularly sampled points.

Our formulation is influenced by implementation considerations. One example of these considerations is the preference to apply learned parameters on as few values as possible. This reduces the memory overhead for the automatic computation of gradients in PyTorch. Another example is the preference to precompute and cache expensive operations, like computing interpolation weights and the logarithmic map, where possible.

4.1.1 *Notation*

We denote feature vectors with a bold font, e.g. $\mathbf{x}_i$. For HSN, these feature vectors are complex valued. Every operation applied to these vectors is performed element-wise. Thus, when we apply parallel transport to a feature vector, the parallel transport is applied to each complex value, a two-dimensional vector in $\mathbb{C}$, and not to the entire feature vector.

The radial profile $R(r)$ and radial offset $e^{\imath\beta}$ form $\mathbf{W}_m$ implicitly. $\mathbf{W}_m$ is a matrix of size $[N_O \times N_F]$ with complex values, where $N_F = c^{(l)}$, the number of input features, and $N_O = c^{(l+1)}$, the number of output features. Thus, when the radial profile and radial offset are applied to $\mathbf{x}_j$ of size N_F, the output is a complex-valued vector of size N_O.

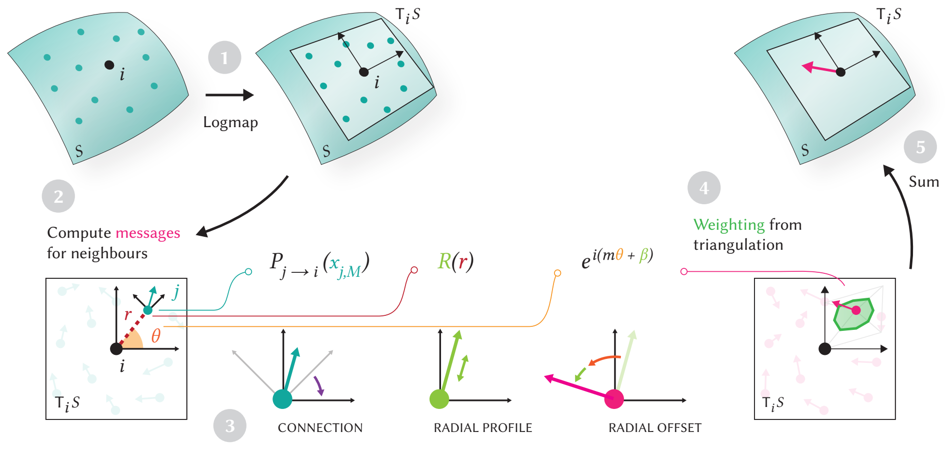


Figure 4.1: Overview of convolution for HSN, given one complex feature $x_{j,M}$: (1) Map neighbours of i from surface $\mathcal{S}$ to $T_i\mathcal{S}$. (2) Evaluate circular harmonic filters for each neighbour to compute messages. (3) Use connection to apply parallel transport. (4) Weight each neighbour with the area of its surrounding triangles. (5) Sum neighbouring messages to get a new feature vector at i in $T_i\mathcal{S}$.

4.1.2 *Continuous Domain*

We define convolution in the continuous domain for each point i and represent the feature vectors of points in i's neighbourhood from stream M and layer l with the function $\mathbf{x}^{(l)}_{(r,\theta),M}$. The value at i in layer $l+1$, using rotation order m, is computed as an integral over the radial neighbourhood within distance ϵ:

$$\mathbf{x}^{(l+1)}_{i,m} = \int_0^{\epsilon}\int_0^{2\pi} R^{(l)}(r)e^{\imath(m\theta+\beta^{(l)})}\mathbf{x}^{(l)}_{(r,\theta),M}\, r\, d\theta\, dr. \tag{4.1}$$

We restructure this formulation to separate the learned components and pull the radial offset out of the integral, motivated by the previously mentioned implementation considerations:

$$\mathbf{x}^{(l+1)}_{i,m} = e^{\imath\beta^{(l)}}\int_0^{\epsilon}\int_0^{2\pi} R^{(l)}(r)e^{\imath m\theta}\mathbf{x}^{(l)}_{(r,\theta),M}\, r\, d\theta\, dr. \tag{4.2}$$

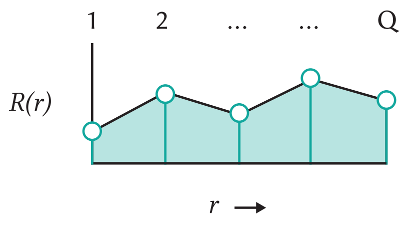


Figure 4.2: We parametrise the radial profile by learning the value at Q equally spaced rings and linearly interpolate for values in between.

4.1.3 *Approximating the Integral on a Surface*

Moving to a discrete setting, our input points are irregularly distributed and exist on a surface. Therefore, we will approximate the integral with a weighted sum over the discrete points within the neighbourhood of i. The neighbourhood of i, $\mathcal{N}_i$ is the set of points within distance ϵ of i. Moreover, we need to push the feature vectors of neighbouring points onto the tangent plane at i with parallel transport:

$$\mathbf{x}_{i,m}^{(l+1)} \approx e^{\imath\beta^{(l)}} \sum_{j\in\mathcal{N}_i} w_j R^{(l)}(r_{ij}) e^{\imath m\theta_{ij}} P_{j\to i}(\mathbf{x}_{j,M}^{(l)}), \tag{4.3}$$

where w_j is a weight for each neighbour, r_{ij} and θ_{ij} are the polar coordinates of point j relative to i, and $P_{j\to i}$ denotes parallel transport along shortest geodesics from the tangent plane of j to that of i.

Radial Profile

For the radial profile R, we learn a limited number of values at equally spaced locations (or rings), which we linearly interpolate to r_{ij} (Figure 4.2):

$$R^{(l)}(r_{ij}) = \sum_{q}^{Q} \mu_q(r_{ij})\rho_q^{(l)}, \tag{4.4}$$

where $\mu_q(r_{ij})$ is a linear interpolation weight. Note that the radial profile could be defined in other ways, which we do not explore in this thesis. Examples are splines or polynomials with learned coefficients.

Complete Formulation

Substituting Equation 4.4 into Equation 4.3, we derive the following formulation of our filter:

$$\mathbf{x}_{i,m}^{(l+1)} \approx e^{\imath\beta^{(l)}} \sum_{j\in\mathcal{N}_i} w_j \sum_{q}^{Q} \mu_q(r_{ij})\rho_q^{(l)} e^{\imath m\theta_{ij}} P_{j\to i}(\mathbf{x}_{j,M}^{(l)}). \tag{4.5}$$

Finally, we separate our filter into a precomputed component and a learned component, denoted by parentheses:

$$\mathbf{x}_{i,m}^{(l+1)} \approx e^{\imath\beta^{(l)}} \sum_{q}^{Q} \rho_q^{(l)} \sum_{j\in\mathcal{N}_i} P_{j\to i}(\mathbf{x}_{j,M}^{(l)}) \left(w_j \mu_q(r_{ij}) e^{\imath m\theta_{ij}}\right). \tag{4.6}$$

A visual overview of this filter can be found in Figure 4.1.

Integration weights

The integration weights w_j are computed using a triangulation of the neighbourhood. If the input is a mesh, we can use the mesh as a triangulation. If the input is a point cloud, a Delaunay triangulation is computed for the points in the tangent plane. For both inputs, the weight is then defined as a third of the sum of areas of all triangles containing j, also known as the vertex lumped mass matrix:

$$w_j = \sum_{ijk \in F} \frac{1}{3} A_{ijk}, \tag{4.7}$$

where A_{ijk} is the area of triangle ijk.

Applying Parallel Transport

We apply parallel transport as a rotation within the tangent plane, referred to as connection. Harmonic Networks guarantee the rotational equivariance property:

$$[\mathbf{W}_m \star \mathbf{X}^{\phi}] = e^{\imath m \phi}[\mathbf{W}_m \star \mathbf{X}^0]. \tag{4.8}$$

Thus, given the angle of the connection, ϕ_{ji}, we can compute the transported feature vector as:

$$P_{j \to i}(\mathbf{x}_{j,M}) = e^{\imath(M\phi_{ji})}\mathbf{x}_{j,M}, \tag{4.9}$$

where M is the rotation order of the source stream of $\mathbf{x}_j$.

4.1.4 *Rotation Orders*

To maintain the rotational equivariance condition throughout the network, we need to keep the output of the filters separated in streams of rotation orders. This concept is explained in the background section on Harmonic Networks. 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: we apply a convolution on $\mathbf{x}_{j,1}$ with $m = -1$ and a convolution on $\mathbf{x}_{j,0}$ with $m = 0$. These convolutions both end up in the stream $M = 0$ and are summed.

We only apply parallel transport to inputs from the $M > 0$ rotation order streams, as the values in the $M = 0$ stream are rotationally invariant.

4.1.5 *Precomputation*

The precomputated part of our convolution layer has three components: the integration weight, the interpolation weight to each radial profile point, and the rotation by the angle θ_{ij}:

$$\text{precomp}_{ij} = \left(w_j \mu_q(r_{ij}) e^{\imath m \theta_{ij}}\right). \tag{4.10}$$

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

We compute the connection angle in precomputation as well, but apply the connection during learning, as it needs to be applied to features inside the network. We use the Vector Heat method out-of-the-box to compute the logarithmic map and connection for each neighbour.

4.2 POOLING

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 nearest neighbours.

The aggregation step in pooling needs to be performed with parallel transport, since the complex features of points within a pooling region do not exist in the same tangent basis. Thus, we define the aggregation step for representative point i with points j in its pooling cluster $\mathcal{C}_i$ as follows:

$$\mathbf{x}_{i,M} = \square_{j \in \mathcal{C}_i} P_{j \to i}(\mathbf{x}_{j,M}). \tag{4.11}$$

Pooling happens *within* each rotation order stream, hence the rotation order identifier M for both $\mathbf{x}_i$ and $\mathbf{x}_j$.

4.2.1 *Precomputation*

The sampling of points per pooling level and construction of corresponding radius-graphs are performed as a precomputation step, so we can compute the logarithmic map and parallel transport for each pooling level using the Vector Heat method Sharp et al. [2019c] in precomputation.

4.3 NON-LINEARITIES

We follow Harmonic Networks (Worrall et al. [2017]) for complex non-linearities: Non-linearities are applied to the magnitudes of the complex features. An example complex version of ReLU:

$$\mathbb{C}\text{-ReLU}(\mathbf{X}e^{\imath\theta}) = \text{ReLU}(\mathbf{X})e^{\imath\theta}. \tag{4.12}$$

4.4 SUMMARY

We propose to apply Harmonic Networks in the tangent plane, solving the orientation ambiguity problem. To accurately compute convolution in streams of $M > 0$, we apply parallel transport to the complex feature maps. We expect that the resulting networks improve in accuracy on existing GDL methods. In the next section, we evaluate these claims and validate our implementation of HSN.

5

IMPLEMENTATION

In this chapter we detail the implementation of HSN in the PyTorch Geometric framework (Fey and Lenssen [2019]). PyTorch Geometric provides implementations for many GDL methods, utilities to process graphs and point clouds, a mini-batch loader, and a multitude of datasets. Furthermore, it provides a way to implement new message passing networks by extending the `MessagePassing` class, which encodes the message passing framework:

$$\mathbf{x}_i^{(l+1)} = \gamma^{(l)} \left(\mathbf{x}_i^{(l)}, \square_{j \in \mathcal{N}(i)} \varphi^{(l)} \left(\mathbf{x}_i^{(l)}, \mathbf{x}_j^{(l)}, \mathbf{e}_{i,j} \right) \right). \tag{5.1}$$

To create a message passing network with the `MessagePassing` class, we define a `message` function, which implements φ, and an `update` function, which implements γ. PyTorch Geometric takes care of the aggregation function $\square$ with fast, GPU-powered scatter and gather operations.

5.1 MESSAGES AND UPDATES

For HSN, the `message` function applies the precomputed filter components to the feature vectors and edge attributes $\mathbf{e}_{i,j} = (r_{ij}, \theta_{ij}, \phi_{ji})$:

$$\varphi^{(l)}(\mathbf{x}_{j,M}^{(l)}, r_{ij}, \theta_{ij}, \phi_{ji}) = P_{j \to i}(\mathbf{x}_{j,M}^{(l)}) \left(w_j \mu_q(r_{ij}) e^{\imath m \theta_{ij}} \right). \tag{5.2}$$

We apply $P_{j \to i}(\mathbf{x}_{j,M}^{(l)})$ as follows:

$$P_{j \to i}(\mathbf{x}_{j,M}^{(l)}) = e^{\imath(M\phi_{ji})} \mathbf{x}_{j,M}^{(l)}, \tag{5.3}$$

where M is the rotation order of the source stream.

The messages are aggregated with an 'add' operation and passed as $\mathbf{x}_{i,M}^{\text{aggr}}$ to the `update` function, which applies the radial profile R and radial offset β for each rotation order m:

$$\mathbf{x}_{i,m}^{(l+1)} = \gamma(\mathbf{x}_{i,M}^{\text{aggr}}) = e^{\imath \beta^{(l)}} \sum_{q}^{Q} \rho_q^{(l)} \mathbf{x}_{i,M}^{\text{aggr}}. \tag{5.4}$$

5.2 NON-LINEARITIES

After each convolution, we apply a non-linearity through the $\mathbb{C}$-ReLu operation, defined in Section 4.3:

$$\mathbb{C}\text{-ReLU}(\mathbf{X} e^{\imath\theta}) = \text{ReLU}(\mathbf{X}) e^{\imath\theta}. \tag{5.5}$$

For every other layer, we apply batch normalisation (Ioffe and Szegedy [2015]) with $\mathbb{C}$-BatchNorm. Like $\mathbb{C}$-ReLu, $\mathbb{C}$-BatchNorm applies batch normalisation to the magnitude of each complex feature and propagates the angle unchanged.

5.3 POOLING

The `MessagePassing` class was also used to implement pooling. We define the message as follows:

$$\varphi(\mathbf{x}_{j,M}, \phi_{ji}) = e^{\imath(M\phi_{ji})}\mathbf{x}_{j,M} \tag{5.6}$$

And let PyTorch Geometric aggregate the messages with a 'mean' operation.

5.4 OUTPUT AND GRADIENT DESCENT

We use the negative log-likelihood to define loss and apply a softmax function followed by a log operation to our predictions before computing the loss. Parameters are optimised through gradient descent with Adam (Kingma and Ba [2014]).

5.5 COMPLEX NUMBER REPRESENTATION

Complex numbers are represented by their real and imaginary components and each rotation order stream is stored separately. Thus, the input and output to each layer is a tensor of size $[N_V, N_M, N_F, N_C]$, where N_V is the number of vertices, N_M is the number of rotation order streams, N_F is the number of features, and N_C is the number of complex components (real and imaginary).

5.6 RADIAL PROFILE AND RADIAL OFFSET

The radial profile and -offset are learned separately for each rotation order by our network. We chose to learn the radial profile by learning parameters for N_R rings. The values in between the ring locations are linearly interpolated.

The radial profile is stored in a tensor of size $[N_M^{\text{in}} * N_M^{\text{out}}, N_R, N_O, N_F]$, where N_M^{in} is the number of input rotation order streams, N_M^{out} is the number of output rotation order streams, N_R is the number of rings, N_O is the number of output features, and N_F is the number of input features. The radial offset is learned as an angle and applied with a rotation matrix. The offset is stored in a tensor of size $[N_M^{\text{in}} * N_M^{\text{out}}, N_O, N_F]$.

To set up a convolutional layer, we have to define the following hyperparameters: the number of rings for the radial profile N_R, the number of rotation order streams N_M, the number of output features per layer N_O, and whether to learn a radial offset as well.

5.7 TRANSFORMS, UTILITIES, AND DATASETS

PyTorch Geometric provides a base class for data transforms. These transforms are applied to the data in each batch before training and can also be applied as a pre-transform to be stored on disk. We wrote a transform that computes the log-map, connection, and vertex lumped mass matrix using the Vector Heat Method (Sharp et al. [2019c]). We wrote a Python binding for Geometry Central (Sharp et al. [2019a]), a C++ library by the authors of the Vector Heat Method, to be able to run the entire pipeline from PyTorch.

Alongside the Vector Heat method, we wrote transforms to compute polar coordinates and connection on small neighbourhoods in point clouds using the method described in Section 3.3.4 (including base estimation). For pooling, we wrote a transformation that precomputes a radius graph for multiple levels and functionality to retrieve the correct radius graph for each pooling level.

Next to these transforms, we wrote a number of utility functions for working with complex numbers and PyTorch modules for complex non-linear functions, like $\mathbb{C}$-ReLU and $\mathbb{C}$-BatchNorm.

Finally, we wrote a number of new `Dataset` classes, that load and clean up data for use in our experiments. One example of such a `Dataset` is the `RotatedMNIST` class, that loads the Rotated MNIST dataset as a graph and constructs the grid graph. Another is the `MNISTSphere` dataset that maps MNIST to a sphere using an elliptical mapping and stores the sphere mesh as the underlying graph.

5.8 REPRODUCIBILITY

The implementation was created with other end-users in mind: it closely follows the conventions of PyTorch Geometric, provides clear documentation, and exhibits a modular design, so that future work could easily build on what has been developed for these experiments.

5.9 TIME AND SPACE COMPLEXITY

We separate the analysis of time and space complexity into two components: precomputation and learning.

5.9.1 *Precomputation*

Precomputation requires the computation of a log map for each vertex, for each model in the dataset. The Vector Heat Method can compute a log map from any source point in linear time, $O(n)$, given a factorisation step that can be precomputed and reused. Repeating this for each vertex on the model results in $O(n^2)$ precomputation time for each model. This can be improved by solving local linear systems, instead of global linear systems, which we leave for future work.

With regards to space, we only store the log-map for local neighbourhoods (k points in each neighbourhood) and for points selected in pooling. This results in the following asymptotic space requirement: $O(nk + \frac{n}{s^1}k + \frac{n}{s^2}k + ...) = O(nk)$, where s is the downscaling factor used in pooling.

5.9.2 *Learning*

Our method requires more computations than most other methods. One reason is the cost of operations on complex numbers, which add a number of multiplications and sine/cosine evaluations. Additionally, the separate rotation order streams require a separate convolution for each input/output rotation order combination. Thus, the number of rotation order streams has a significant impact on performance. It is hard to provide a concrete formulation for the time complexity, as PyTorch adds additional overhead for the computation of gradients. We did find that our method performs significantly slower than other methods: about 30x slower. We argue that this is due to the additional cost of separate rotation order streams and the cost of computing gradients derived from complex operations.

We intend to optimise our code to improve the learning speed by writing the convolution as a C++/CUDA extension for PyTorch, reducing the cost of interpreter and autograd overhead.

We can provide an estimation of space requirements for the network (Table 5.1). Each layer requires $N_M^{\text{in}} * N_M^{\text{out}} * N_R * N_F * N_O$ parameters for the radial profile and $N_M^{\text{in}} * N_M^{\text{out}} * N_F * N_O$ for the radial offset. N_M^{in} and M_{out} are typically lower or equal to 2 and the number of rings range from 2 to 4. Compared to a regular CNN, we have a similar amount of parameters to learn, depending on the values set for N_R and N_M: CNNs require $K * K * N_F * N_O$ parameters, where K is the filter size. K tends to range between 3 and 5. Instead of a separate set of parameters for *each path* between rotation order streams, one can consider learning parameters per rotation order stream. This reduces the number of parameters to $(N_R + 1) * N_M^{\text{out}} * N_F * N_O$.

HSN	**CNN**
$(N_R + 1) * N_M^{\text{in}} * N_M^{\text{out}} * N_F * N_O$	$K * K * N_F * N_O$

Table 5.1: Comparison of number of parameters for CNN and regular CNNs.

6 EXPERIMENTS

In our experimental evaluation we aim to test two hypotheses:

1. **Hypothesis 1 (H1)**: HSNs improve over rotationally invariant methods (like GCN or PointNet) by including rotation order streams of $M > 0$.
2. **Hypothesis 2 (H2)**: HSNs improve over directional methods by accurately relating values from different tangent planes with parallel transport along shortest geodesics.

We test these hypotheses on three different tasks: *classification* on Rotated MNIST and on Rotated MNIST mapped to a sphere, *correspondence* on FAUST, and *shape segmentation* on FAUST.

The hypotheses are tested by setting up an HSN with two streams $(0, 1)$ for each task, and evaluating the performance after 100 epochs of training on that task. We did not tune hyperparameters. We then compare HSN with other configurations of HSN and other methods using the same architecture and hyperparameters. For each hypothesis we use a different set of comparable methods:

1. **(H1)** HSN with only the 0-equivariance stream and GCN, a widely used rotationally invariant network similar to PointNet.
2. **(H2)** HSN without parallel transport and MoNet, a state of the art spatial method.

Additionally, we report on experiments supporting the design choices for transferring Harmonic Networks to the irregular domain.

6.1 ROTATED MNIST

We recreated the experiment performed by Worrall et al. [2017] on Rotated MNIST (Larochelle et al. [2007]). This dataset consists of 10000 training images, 2000 validation images, and 50000 test images and was constructed by randomly rotating images from the MNIST dataset. The small training set, relative to the test set, and $360°$ rotations make this a challenging task for traditional CNNs.

The goal of this experiment was twofold: (1) to validate the implementation of our HSN and (2) to test hypotheses H1 and H2.

6.1.1 *Setup*

Each image from Rotated MNIST was mapped to a graph, in which nodes represent pixels and edges represent adjacencies of pixels (Figure 6.1). These graphs were used to train and test our HSN network. Each node is associated with its position in the pixel grid and we compute relative polar coordinates for each edge in the input graph based on these positions. We add a base offset to each node's polar coordinates to simulate the change of basis on a surface and compute connection as the difference between base offsets: $\phi_{ji} = \text{offset}_j - \text{offset}_i$.

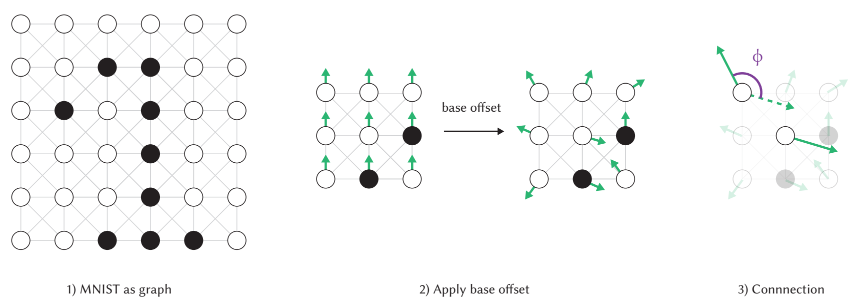


Figure 6.1: Rotated MNIST is converted to a graph. Next, we apply a base offset and compute connection as the difference between base offsets.

Architecture

We recreated the network used by Worrall et al. [2017] for Rotated MNIST with HSN, using two rotation order streams. The network consists of seven convolutional layers (features: 8, 8, 16, 16, 32, 32, 10), interspersed with two pooling layers, and finally a global mean-pooling operation. We set the number of rings for the radial profile to 5. Our filters are applied to a 5×5 region. Please refer to Figure 6.2 for an overview of the architecture.

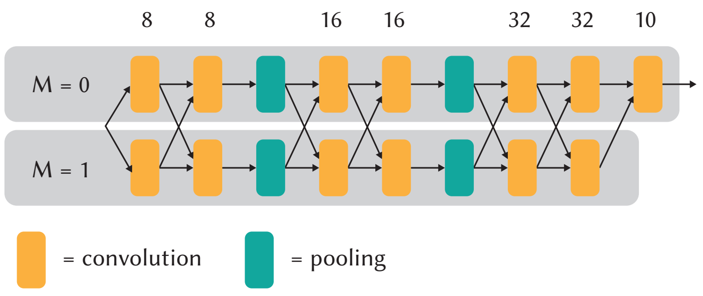


Figure 6.2: Architecture for classification of Rotated MNIST.

6.1.2 *Validation*

The first step is to validate that our test accuracy approximates that of Harmonic Networks. Worrall et al. [2017] report 98.31% accuracy, which we were able to reproduce with their implementation[1]. Although we tried to mimic Harmonic Networks' implementation as closely as possible, we expect that HSN will achieve slightly lower accuracy, due to generalisations required to work on irregular graph structures: we do not use padding, interpolate the radial profile, and approximate the continuous integral in a different way, by using a weighted sum. Therefore, to pass our validation, we allow a small margin of 2%.

1 This result was achieved by training on both the training and validation set. Without the validation set, the results drop by about 0.5%

Method	Rotation order streams	Accuracy
Harmonic Networks	0, 1	98.13%
HSN (ours)	0, 1	96.71%

Table 6.1: Results of HSN and Harmonic Networks on the Rotated MNIST dataset. For HSN, we applied a base offset to each node's polar coordinates.

Method	Rotation order streams	Accuracy
Harmonic Networks	0	93.55%
Harmonic Networks	0, 1	98.13%
Harmonic Networks	0, 1, 2	**98.31%**
Harmonic Networks	0, 1, 2, 4	97.79%
HSN (ours)	0	84.64%
HSN (ours)	0, 1	**96.71%**

Table 6.2: Results of HSN and Harmonic Networks on the Rotated MNIST dataset for different rotation order stream configurations. We observe that the 1-equivariant stream improves the accuracy of our network.

Results

HSN achieves 96.71% accuracy (Table 6.1). This confirms that our implementation matches the performance of Harmonic Networks and that one of the fundamental ideas of this thesis, that we can apply connection with a simple rotation of filter activations, is correct.

6.1.3 *Hypothesis 1*

To test hypothesis 1, we compare a dual-stream HSN ($M = \{0, 1\}$) with a single-stream HSN ($M = \{0\}$). We test the same configurations for the Harmonic Networks implementation as well. We expect that the dual-stream HSN will show improved results, compared to its single-stream counterpart.

Curious about the effect of adding more rotation order streams, we tested Harmonic Networks with additional streams. The authors of Harmonic Networks claim that higher order streams do not add significantly to the performance, as images do not exhibit high frequency patterns, rotationally (Jacobsen et al. [2016]), so we expect to see little additional improvement.

Results

We observe that the 1-equivariant stream improves the accuracy by about 4.58% for Harmonic Networks (Table 6.2). For HSN, the gap is even larger: 12.07%. We also see that higher rotation order streams do not significantly improve our results, supporting our expectation on higher rotation order streams.

MODEL COMPLEXITY The improvement in accuracy might be explained purely by the increased complexity of extra rotation order streams: each stream learns a separate set of filters, adding to the number of parameters to be learned. This would mean that we cannot attribute the improvement in accuracy to the additional rotation orders. To test this, we repeated our experiment with a single 0-equivariant stream for higher feature counts and, additionally, with two 0-equivariant streams (Table 6.3). We observe that additional parameters do not improve performance, most likely because the network is overfitting due to the increased complexity. We

Method	**Rotation order streams**	**# Features**	**Accuracy**
Harmonic Networks	0	$\times 1$	93.55%
Harmonic Networks	0	$\times 4$	93.39%
Harmonic Networks	0, 1	$\times 1$	**98.13%**
HSN (ours)	0	$\times 1$	84.64%
HSN (ours)	0	$\times 4$	84.58%
HSN (ours)	0, 0	$\times 1$	83.96%
HSN (ours)	0, 1	$\times 1$	**96.71%**

Table 6.3: Results of HSN and Harmonic Networks on the Rotated MNIST dataset for a single stream and different parameter counts. Adding parameters does not have the same effect as adding higher rotation order streams.

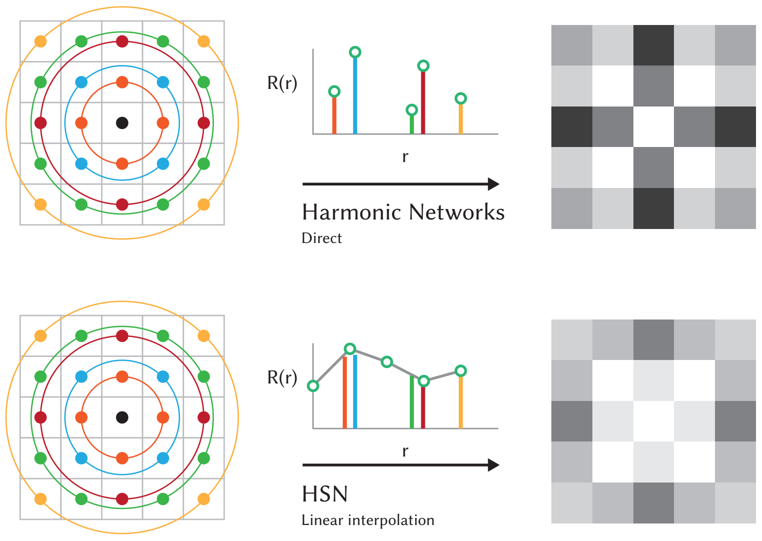


Figure 6.3: The difference between how Harmonic Networks and HSN learn their radial profile could lead to a difference in angular contrast.

also rule out the possibility that the stream architecture is the reason for our observed improvement.

DIFFERENCE FOR SINGLE STREAM The lower performance for $m = 0$ in HSN, compared to Harmonic Networks, requires explanation. Our speculation is that the decreased performance is caused by how we learn the radial profile. Harmonic Networks learn a distinct radial profile value for each ring of pixels, where HSN interpolates the radial profile from radial profile values at fixed locations (Figure 6.3). Harmonic Networks could therefore learn filters with more 'angular contrast' that HSN cannot not construct. One way to solve this, would be to also learn the locations of the radial profile parameters. Because that would increase the complexity of the network, and because our dual-stream network performs well, we chose not to include this.

The experiments on Rotated MNIST support hypothesis 1. Additionally, we found that higher rotation order streams do not improve learning for MNIST and rule out the explanation that the improvement can be attributed to increase in complexity of the model.

Method	**Rotation order streams**	**Accuracy**
HSN (no parallel transport)	0, 1	90.57%
HSN (ours)	0, 1	96.71%

Table 6.4: Results of HSN and HSN without parallel transport. Parallel transport improves HSN's performance.

6.1.4 *Hypothesis 2*

To test hypothesis 2, we compare HSN to a configuration of HSN without parallel transport, i.e. without connection. We argue that Rotated MNIST with random base offsets provides a trustworthy proxy for understanding the effect of parallel transport for real-world data: a random orientation in the tangent plane is the worst case scenario we expect to see for surfaces.

We expect that HSN with parallel transport will improve on HSN without parallel transport.

Results

Where HSN with parallel transport achieved 96.71%, the variant without parallel transport only reached an accuracy of 90.57% (Table 6.4). A drop in performance of 6.14%.

This experiment supports hypothesis 2, demonstrating an improvement by adding parallel transport to an otherwise unchanged HSN.

6.2 ROTATED MNIST ON THE SPHERE

Next, we move from the plane to a curved surface by mapping Rotated MNIST to a sphere. Our goal with this experiment was to test both our hypotheses. To this end, we compare HSN with two competing methods (GCN and MoNet) in a controlled setting with a familiar dataset.

6.2.1 *Setup*

We use an elliptical mapping (Fong [2015]) to map each of the 642 vertices on the sphere to a location in a square grid. The elliptical mapping maps a position in the unit disc (u, v) to a position in the unit square (x, y). In our case, from each half of the sphere to a square image.

The x and y coordinates resulting from this mapping are floats. We bilinearly interpolate from the pixel locations to x and y, resulting in a value for each vertex on the sphere (Figure 6.4).

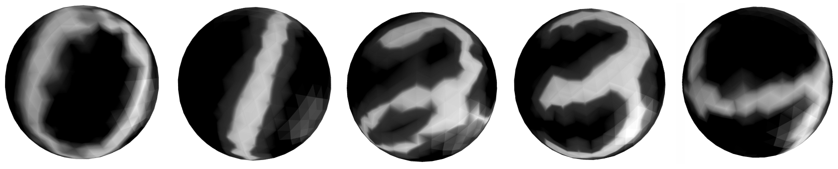


Figure 6.4: Rotated MNIST mapped to a sphere

Architecture

We use the exact same architecture as was used for Rotated MNIST (Figure 6.2): seven convolutional layers with two pooling layers in between.

Pooling

We use mean-pooling on clusters based on Farthest Point Sampling (Section 4.2). The pooling layers use the following downsampling ratios: $0.5, 0.25$. We use the following radii to construct a radius graph for each pooling level, starting at the full input: $0.3, 0.45, 0.8$[2].

Orientation within Tangent Plane

To construct polar coordinates for each neighbouring vertex, we need to choose an orientation within the tangent plane. The Vector Heat method's implementation uses the first edge connected to a vertex as the orientation within the tangent plane. Because the same sphere mesh is used for each training sample, the orientation within the tangent plane at each vertex is consistent for each training sample. Yet, the orientation within the tangent plane is different for different vertices in the same model.

Our choice of orientation within the tangent plane has the following consequences for our experiments:

1. We can test our solution for the orientation ambiguity problem, as different vertices use different orientations.

2 Recall that the input is a unit sphere.

2. MoNet can be applied to the dataset without angular max-pooling, as the choice of orientation is consistent across each input model. This change is in favor of MoNet, as it stabilises the learning process compared to angular max-pooling. We prefer using MoNet without angular max-pooling, as angular max-pooling is a costly operation: each filter response has to be computed multiple times. In our current testing framework this also resulted in a large memory overhead that could not be resolved within the desired time frame.

3. Rotationally invariant methods, and thus our experiments on hypothesis 1, are not affected.

For future work, we suggest adding a random offset the the polar coordinates of each point, like we did for Rotated MNIST in the plane. This will greatly simplify interpreting experiments for hypothesis 2 and align more with what we expect to see for real-world datasets.

6.2.2 *Hypothesis 1*

To test hypothesis 1, we compare a dual-stream HSN with a single-stream configuration and GCN, a rotationally invariant network. GCN is a widely used method for learning on graphs that is similar to works that learn specifically on point clouds or meshes (e.g. PointNet or EdgeConv).

We also tested a version of GCN that takes the maximum of its neighbours, instead of the average. We refer to this version as GCN (max). GCN with a max operation is similar to PointNet, giving us an impression of how our network would compare with PointNet.

For single-stream HSN, GCN, and GCN (max), we increase the number of features to compensate for the extra stream of dual-stream HSN.

Because the single 0-equivariant stream achieves rotational invariance, we expect that the single-stream HSN and GCN achieve similar results. Additionally, we expect that the additional 1-equivariant stream in the dual-stream HSN improves the performance.

Results

We observe that HSN with two streams clearly outperforms all other configurations: 94.10% against 79.97% by GCN (max) (Table 6.5). This is an increase in performance of 14.13%. Note that we did not tune any hyperparameters and used the exact same number of parameters (accounting for streams) and setup for each method[3] We also see that the single-stream HSN resembles the accuracy of GCN, following our expectation. This leads us to interpret this result in support of hypothesis 1.

Method	**Rotation order streams**	**Accuracy**
GCN	0 (equivalent to)	77.49%
GCN (max)	0 (equivalent to)	79.97%
HSN (ours)	0	76.52%
HSN (ours)	0, 1	**94.10%**

Table 6.5: Results of GCN and HSN tested on Rotated MNIST mapped to a sphere for different stream configurations.

3 We also tested the single-stream configurations with the same number of features. This resulted in a lower accuracy. Therefore, we only report on the experiments with higher parameter counts.

6.2.3 *Hypothesis 2*

To test hypothesis 2, we compare our dual-stream HSN ($M = 0, 1$) network with MoNet. Again, we use the exact same architecture for all methods and use the same number of parameters. We do not tune any hyperparameters and use the same logarithmic map for MoNet and HSN, derived from the Vector Heat method.

We expect MoNet to do well, because it is a state of the art spatial method. We also expect that HSN can improve on MoNet, because it accounts for the orientation ambiguity problem, where MoNet does not.

Results

We observe that HSN has a slight advantage over MoNet (94.10% vs. 92.19%, Table 6.6). We suggest the following explanations for this low improvement: (1) MoNet filters have a higher degree of freedom, giving MoNet an initial edge over HSN. (2) We did not use angular max-pooling for MoNet, enabling a stabler learning process that we would not see in real-world datasets.

Nonetheless, HSN has improved over MoNet, demonstrating the importance of correctly relating different tangent plane orientations.

Method	**Rotation order streams**	**# Features**	**Accuracy**
MoNet	-	$\times 1$	87.29%
MoNet	-	$\times 4$	92.19%
HSN (ours)	0, 1	$\times 1$	**94.10%**

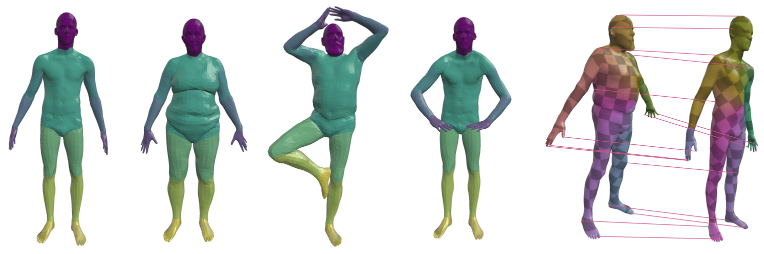
Table 6.6: Results of MoNet and HSN tested on Rotated MNIST mapped to a sphere.

6.3 SEGMENTATION AND CORRESPONDENCE ON FAUST

Finally, we apply HSN to a dataset derived from real-world scans: FAUST (Bogo et al. [2014]). FAUST contains 100 scans of bodies in different configurations. We split the dataset in a train set of 80 scans and a test set of 20 scans. Each model contains 6890 vertices.

The first task is segmentation: labelling each vertex of the model with one of eight segment labels (Figure 6.5a). The second task is correspondence: labelling each vertex with its index (Figure 6.5b). This means that each vertex has to be assigned the correct label out of 6890 possibilities.

We aim to test our two hypotheses by comparing a dual-stream HSN with several configurations of HSN as well as GCN and MoNet.



(a) Segmentation (b) Correspondence

Figure 6.5: Segmentation and correspondence on FAUST: 8 or 6890 class labels for 6890 vertices.

6.3.1 *Setup*

We use the meshes from FAUST as input to our network. The input features consist of the raw xyz-coordinates of the vertices. The accuracy of each task is reported as the percentage of vertices that were labelled correctly. For segmentation, this is a natural performance measure. For correspondence, researchers tend to use a more lenient performance measure: the percentage of correct matches, given a certain margin of geodesic distance. Because FAUST uses the same vertex topology for each model, we can use the exact accuracy as a performance measure.

Architecture

We set up a simple architecture to perform both tasks (Figure 6.6): six convolutional layers with the following number of output features: $32, 32, 64, 64, 64, 64$. This is followed by a fully connected layer of 128 features and a final fully connected layers with the number of classes ($N_{classes}$) as output. For segmentation $N_{classes} = 8$, for correspondence $N_{classes} = 6890$.

We do not apply pooling and have not tuned our hyperparameters. Therefore, we do not expect to surpass state of the art approaches that use ResNet architectures or additional tuning measures.

Orientation within Tangent Plane

Like our sphere dataset, FAUST has a consistent mesh structure for each model in the dataset. Consequently, the orientation in the tangent plane for each vertex is consistent for each model, enabling us to apply MoNet without angular max-pooling.

The consistent choice of orientation has made it difficult to perform trustworthy experiments on correspondence for MoNet and HSN without parallel transport.

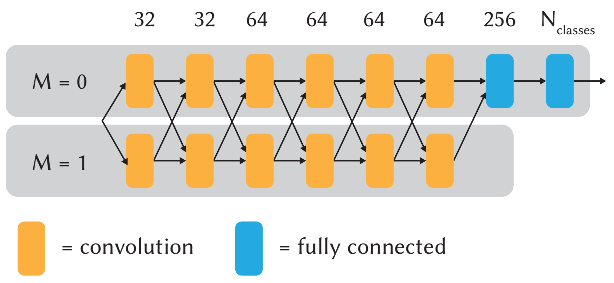


Figure 6.6: Network architecture for learning on FAUST.

Because the orientation at each vertex is consistent, the difference of orientations between neighbouring vertices is consistent. In addition, the difference of orientations is unique for each vertex pair. Therefore, orientation ambiguity becomes a highly effective *feature* to predict the index of each vertex.

In real-world scans, we do not have such consistent orientations and MoNet has to resort to angular max-pooling to choose an orientation. Due to time constraints, we were not able to implement this within our testing framework for a fair comparison. Therefore, we omit the experiments on correspondence for hypothesis 2.

6.3.2 *Hypothesis 1*

We compare a dual-stream HSN with a single-stream HSN, GCN, and GCN (max). We expect that the dual-stream HSN improves on all of these methods.

Segmentation Results

We observe that HSN performs well on the segmentation task: 97.54% (Table 6.7). GCN and single-stream HSN show about a 9% drop in accuracy and demonstrate similar accuracy to each other. GCN (max) does worse than any of the other methods.

Method	**Rotation order streams**	**Accuracy**
GCN	0	88.67%
GCN (max)	0	82.30%
HSN (ours)	0	87.34%
HSN (ours)	0, 1	**97.54%**

Table 6.7: Results of segmentation on FAUST with HSN and GCN.

Correspondence Results

This task is much harder than segmentation, attested to by the low accuracy of GCN and single stream HSN: only 19.67% (Table 6.8). The additional 1-equivariant stream for dual-stream HSN improves the accuracy by 50%, clearly demonstrating the added benefit of the 1-equivariant stream. Again, we see a high similarity between GCN and single-stream HSN, confirming our expectation that these methods exhibit similar properties for learning.

To place the performance of HSN into perspective: GCNN achieves results in the range of $60 - 70\%$, PointNet and EdgeConv achieve $< 10\%$ for exact accuracy (Poulenard and Ovsjanikov [2018]).

Method	**Rotation order streams**	**Accuracy**
GCN	0	19.67%
GCN (max)	0	4.36%
HSN (ours)	0	19.66%
HSN (ours)	0, 1	**68.42%**

Table 6.8: Results of correspondence on FAUST with HSN and GCN. HSN improves on GCN and its single-stream counterpart.

6.3.3 *Hypothesis 2*

We compare a dual-stream HSN with a MoNet setup for segmentation. We have mixed expectations for this hypothesis: with its higher degree of freedom, MoNet could produce more salient features. This might outweigh the loss of accuracy from orientation ambiguity.

Method	**Rotation order streams**	**Accuracy**
MoNet	-	84.52
HSN (ours)	0, 1	**97.54%**

Table 6.9: Results of segmentation on FAUST with HSN and MoNet.

Results

For segmentation, HSN improves over MoNet by about 13.02% (Table 6.9).

We are hesitant, however, to derive any hard conclusions from these experiments, because our comparison methods were not up to par: MoNet was implemented by its authors with only three layers and angular max-pooling. We used MoNet in a network with seven layers and a different number of kernels.

For future work, we intend to exactly replicate the experiments of Monti et al. [2017] and compare with the results as reported by the authors using the same performance metric based on geodesic distance-error.

6.4 ROTATIONAL EQUIVARIANCE FOR IRREGULAR SAMPLING

Circular Harmonic filters and their rotational equivariance property are designed in the continuous domain. To apply these filters on discrete surface representations, we have to approximate the continuous convolution somehow. We performed the following experiments to find an interpolation or weighting method that would best preserve the rotational equivariance property.

6.4.1 *Interpolation and Weighting*

These experiments were performed at an early stage of the thesis and use a step that was later found to be unnecessary: in these experiments we first interpolate to a number of grid points and then sum over these grid points. From these experiments, we concluded that interpolation based on a Delaunay triangulation would be most suitable for our method.

If we increase the number of grid points to infinity, we find that this method is equivalent to weighting each irregular grid point with a third of the sum of its surrounding triangles. This makes the triangulation weighting more accurate than any of the surveyed methods. Additionally, it removes the need to set a number of grid points and reduces the computational complexity of our method.

Even with this change, the results are informative for rotational equivariance on irregular grids.

6.4.2 *Irregular Samples and Rotation*

Motivation and questions

A crucial ingredient in our method is rotational equivariance. To apply connection to activations, we should be able to associate a rotation in the input with a rotation in the output:

$$[\mathbf{W}_m \star \mathbf{F}^{\phi}] = e^{\imath m\phi}[\mathbf{W}_m \star \mathbf{F}^{0}]. \tag{6.1}$$

The main question for this experiment is the following: does the rotational equivariance property hold for irregularly sampled functions? In the process, we aim to decide on which interpolation method to use, and find out more about the limits of the rotational equivariance property when applied to real images.

Method

The following experiment was set up to answer these questions: k unique samples were taken from an image using a uniform distribution within a circle. These k samples were interpolated to a polar grid with n rings and m angles (Figure 6.7) to form input x.

Next, a circular harmonic filter with radial profile $R(r) = 1 - r$ and phase offset $\beta = 0$ was sampled at the same polar grid to form filter W. The cross correlation for one patch was then computed, resulting in activation a.

This process was repeated for the same input image rotated around its centre by ϕ, resulting in a rotated activation a^{ϕ}.

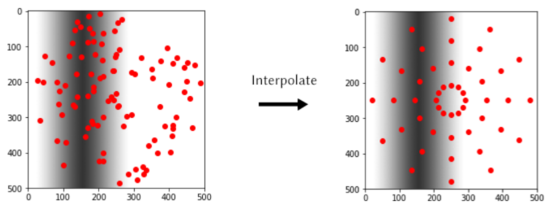


Figure 6.7: Left: Samples taken from a test image. Rejection sampling was used to sample k unique points from a uniform distribution within a circle. Right: interpolated to a regular polar grid with n rings and m angles.

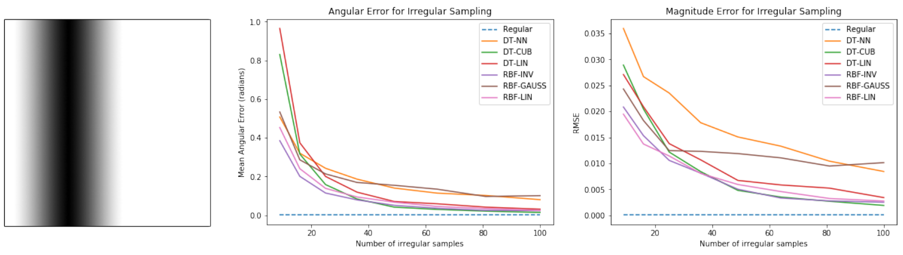


Figure 6.8: Angular and Magnitude error plotted against the number of irregular samples.

We set $a^{\phi'} = e^{i\phi}a$. It follows from Equation 6.1 that $a^{\phi} = a^{\phi'}$. To validate this, we repeat the experiment for 8 rotations and 100 repetitions and observed the RMSE of the magnitude and the mean of the angular error L_{ϕ}:

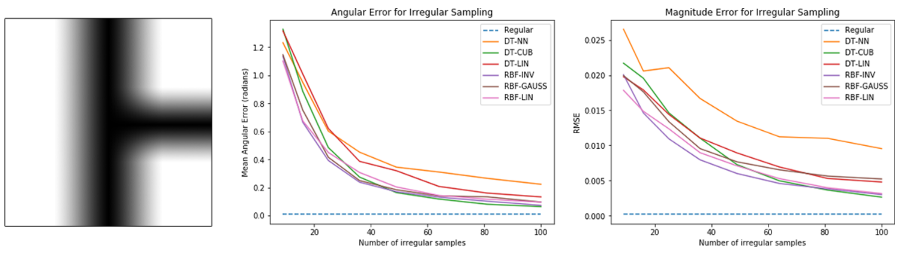
$$L_{\phi} = \cos^{-1}\left(\frac{a^{\phi'}_{Re}a^{\phi}_{Re} + a^{\phi'}_{Im}a^{\phi}_{Im}}{|a_{\phi'}||a_{\phi}|}\right). \tag{6.2}$$

We use a polar grid, consisting of the same number of points used in Harmonic Networks: 4 rings and 12 points on each ring. The number of rings is limited, as it directly impacts the number of weights used in the network. We report the angular error and magnitude error for the following interpolation methods alongside a zero measurement of direct sampling on the polar grid (Figure 6.8):

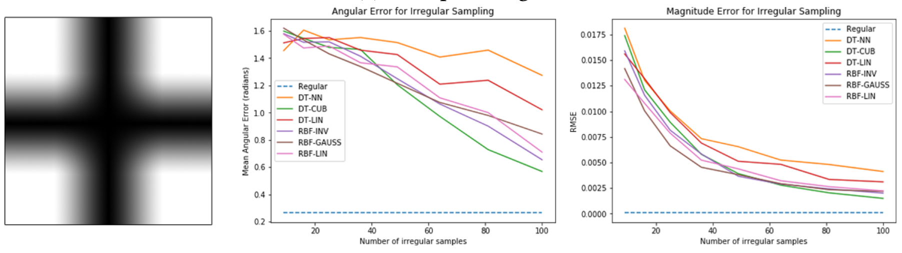
- Delaunay triangulation
  - Nearest Neighbour (DT-NN)
  - Linear (DT-LIN)
  - Cubic (DT-CUB)
- Radial Basis Functions
  - Inverse Multiquadric (RBF-INV)
  - Gaussian (RBF-GAUSS)
  - Linear (RBF-LIN)

Discussion

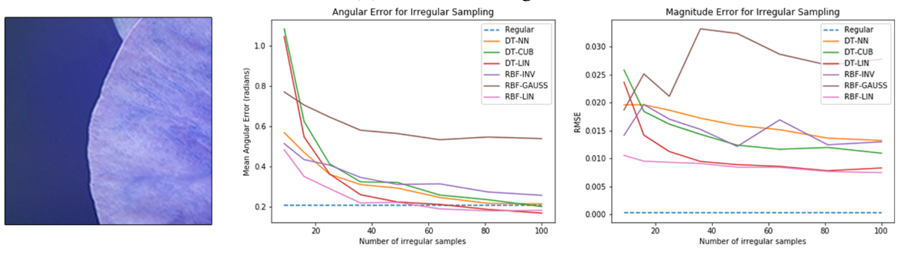
The resulting plots show that RBF-INV and DT-CUB perform best for the provided input image. From $k = 25$ we see that the error flattens to an acceptable low error, showing that interpolation can provide an approximation that is useful for rotational equivariance. One problem with DT-CUB is that it first computes a convex



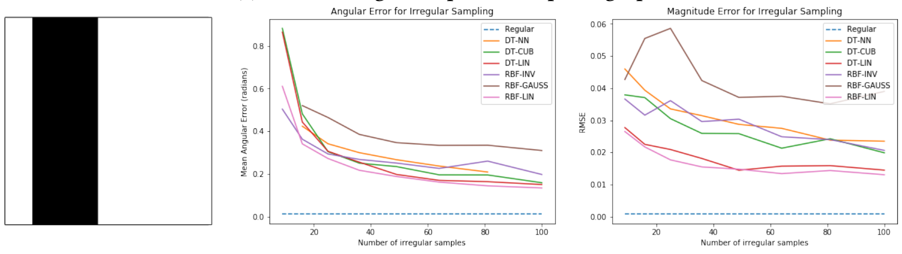
(a) T shaped image.



(b) Crosshair image.



(c) Flower image, sampled from photograph.



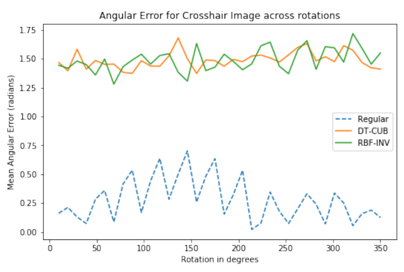
(d) Sharp edge image.

Figure 6.9: Angular and Magnitude error plotted against the number of irregular samples for several images.

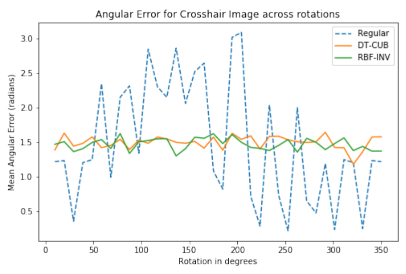
hull and interpolates values within the convex hull of the sampled points, leaving some points without a value, whereas RBF-based methods have infinite support.

Next, we repeat the same experiment for different images. Shown are the images besides their corresponding plots (Figure 6.9a, Figure 6.9b, Figure 6.9c, Figure 6.9a). The images that were used are smooth, differentiable functions, as we expect to see on a manifold. From experiments we find that the interpolation fails for high-frequency images or images with sharp edges (Figure 6.9d).

We also find that asymmetric images perform best, especially the image sampled from a real photograph (Figure 6.9c). The crosshair image fails in restoring the correct rotation. We investigate this specific case by plotting the rotation angle against the mean angular error (Figure 6.10a). The angular error is the same across all rotations, so there must be some other effect than rotational symmetry at play. We suggest the following explanation: the gradient of the crosshair, combined with interpolation gives inconsistent values for the polar grid. We investigated this by plotting the values on the polar grid (not shown in this report) and comparing the re-



(a) Gradient crosshair.



(b) Sharp-edged crosshair.

Figure 6.10: Rotation vs Mean Angular Error for the crosshair image.

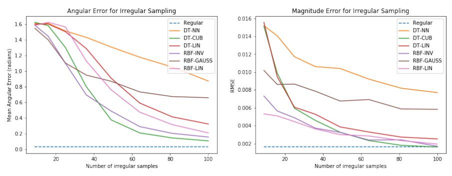


Figure 6.11: Angular and Magnitude error plotted against the number of irregular samples for rotation order $m = 2$.

sults with a sharp-edged crosshair (Figure 6.10b). This plot supports our suggestion: with sharp edges, the errors for interpolated values correspond to those for regular sampling. Both errors are high, which could be attributed to rotational symmetry, as can be seen in the regular sampling plot.

A final analysis is concerned with the effect of the rotation order m. We repeat the above experiment for $m = 2$ and create the same plots (Figure 6.11. We find that irregular sampling performs worse for lower sample numbers, but catches up with $m = 1$ at $k = 64$. Again, DT-CUB is the best interpolation method.

Conclusion

From the above experiments, we conclude that interpolation gives a good approximation for $k >= 20$, especially for smooth, low-frequency images. We find that cubic interpolation based on a Delaunay triangulation is the best method to use, outperforming the other methods across the board.

7 DISCUSSION AND CONCLUSION

With Harmonic Surface Networks (HSN), we present a solution to the orientation ambiguity problem that combines geometric convolutional networks and equivariant networks. In doing so, we have (1) introduced a concept for vector-valued convolutional networks on surfaces and (2) extended Harmonic Networks to irregular domains.

We evaluated our approach, testing two main hypotheses about our method, and supported important design decisions experimentally. We will now discuss our results, generalisability and potential applications, limitations and practical considerations, and future work.

7.1 RESULTS

HYPOTHESIS 1 (H1) We are able to confidently state that hypothesis 1, on the benefit of higher rotation order streams, holds true. Across every experiment, and with a great margin, the two-stream HSN demonstrated improved performance compared to a single-stream HSN and GCN.

HYPOTHESIS 2 (H2) We grew our confidence in hypothesis 2 and the experimental results are promising. We can conclude that parallel transport is beneficial to learning on surfaces for HSN, based on our experiments with random base offsets. We cannot yet make a solid claim about the benefit of parallel transport compared to state of the art approaches, like MoNet, due to limited comparison experiments.

7.2 GENERALISABILITY

HSN can be applied to many learning problems on surfaces, and potentially on higher-dimensional manifolds. While we demonstrate the effectiveness of HSN on triangle meshes, the basic building blocks of our method also work on point clouds and other discretisations of surfaces, such as polygonal meshes and voxels. In this section, we highlight a number of potential applications, as well as the generalisation to point clouds and higher-dimensional manifolds.

7.2.1 *Applications*

Spatial geometric deep learning methods have a wide array of applications. Major applications within the area of computer graphic are shape correspondences, segment labelling, and shape completion and generation. Other applications described in the literature are learning shape descriptors, normal prediction, keypoint prediction, style transfer, and shape generation. A thorough review of these applications can be found in Bronstein et al. [2017].

Our method provides an additional 'hook' for applications: the output of our layers is complex valued, representing vectors in the tangent plane. We currently only use the magnitude of these activations to derive classifications and discard the orientation. The orientation could be of use for applications with vectors as input or output.

7.2.2 *Point Clouds*

Our implementation can simply be extended to point clouds: we compute parallel transport and the logarithmic map with the Vector Heat method, which generalises to any structure with a Laplacian and connection Laplacian. The connection Laplacian for a point cloud can be computed by the method described by Singer and Wu [2012]. The integration weights can be derived from a Delaunay triangulation in the tangent plane, instead of the vertex lumped mass matrix.

7.2.3 *Higher-Dimensional Manifolds*

HSN might generalise to higher-dimensional manifolds. This will require the following advancements: higher-dimensional harmonic networks, a higher-dimensional logarithmic map and higher-dimensional parallel transport. Thomas et al. [2018] show a generalisation of Harmonic Networks to 3D rotations, which could be a pointer for work into higher dimensional harmonic networks. Singer and Wu [2012] present their connection Laplacian approach for any number of dimensions and the Vector Heat method could use this to generalise to higher dimensions. In the near future, we will remain focused on improving HSN for surfaces. In the far future, we could see an application of HSN in higher-dimensional manifolds.

7.3 LIMITATIONS AND PRACTICAL CONSIDERATIONS

Our method comes with a number of limitations and practical considerations, which we discuss in the following section. We discuss our considerations on intrinsic learning, the expressiveness and understanding of circular harmonic filters, and the limitations posed by irregular inputs.

7.3.1 *Intrinsic vs. Extrinsic*

HSN learns filters in the two-dimensional tangent plane. This is often referred to as intrinsic learning, as the filters are applied 'inside' the surface. Methods that apply convolutions on the three-dimensional embedding space are referred to as extrinsic approaches. Examples of extrinsic approaches are multi-view CNNs (Huang et al. [2018]) and volumetric CNNs (Wu et al. [2015]).

Although intrinsic learning has many advantages over extrinsic learning (Boscaini et al. [2016]), it also comes at a cost. Examples of problematic cases for intrinsic representations are shapes that seem to merge into one shape but are strictly separate, such as feathers on a bird (Poulenard and Ovsjanikov [2018]). An extrinsic representation will 'see' that points on separate feathers are close to each other in the embedding space, while the intrinsic representation will perceive a much larger distance. This can be a problem for tasks such as shape segmentation.

Our approach is also limited to polar coordinates as parametrisations of the tangent plane. Verma et al. [2018] show with FeastNet that other local parametrisations, for example those learned from features inside the network, could improve the accuracy of networks.

7.3.2 *Expressiveness of the Filters*

The filters used in the network are constrained to the circular harmonics family. This strictly reduces the number of configurations our filters can take. We believe that radial offset plays an important role in constructing more informative features. Without it, the filters simply act as an averaging operation over surrounding features, resulting in an activation that points in the direction of the highest activations. By adding the radial offset parameter, the network can influence how to relate different rotations, which increases the expressiveness of these filters.

We expect that the reduced expressiveness of these features is most apparent in shallow neural architectures; in deeper architectures, where filters in later layers cover a wide surface area, the advantages of accurate diffusion with parallel transport may outweigh the disadvantages of a constrained filter.

Understanding of Harmonic Filters

This brings us to another limitation of HSN: it is hard to get a good grasp of the types of filters learned by the network. Although we can make some statement about the magnitude and angle of activations in early layers, the radial offset and input data makes it hard to understand what our network is doing. We can design networks that function with high accuracy, but we need to understand better what our filters are doing to improve their accuracy. We propose investigating this by visualising our filters and visualising what our filters 'look for' by generating shapes that maximise filter activations.

7.3.3 *Irregularity*

HSN uses a weighted sum to approximate a continuous integral. The weights used for this are derived from the area of triangles surrounding each point. Consequently, HSN's accuracy depends on the quality of the triangulation. We suggest using an intrinsic Delaunay triangulation to minimise the error introduced by bad triangulations (Sharp et al. [2019b]).

There are other ways to approximate a continuous integral, such as radial basis functions. We have used a weighting based on triangulations, as it demonstrated high accuracy and reduces the number of hyperparameters compared to other interpolation methods. Nevertheless, further numerical optimisations could improve this component of HSN.

7.4 FUTURE WORK

We have demonstrated a new concept for vector-valued convolutional networks on surfaces. Although the experiments performed in this thesis are promising and the concept is sound, we need to evaluate our method on a wider range of applications and experiment with other network architectures to finalise the concept. Ideally, this involves an implementation for point clouds.

We intend to share our implementation as a contribution to PyTorch Geometric, so other researchers can study and replicate our work. Such research could move in the direction of applications for 1-equivariant networks or, in the future, higher-dimensional manifolds.

BIBLIOGRAPHY

Bogo, F., Romero, J., Loper, M., and Black, M. J. (2014). FAUST: Dataset and evaluation for 3D mesh registration. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, Piscataway, NJ, USA. IEEE.

Boscaini, D., Masci, J., Rodolà, E., and Bronstein, M. (2016). Learning shape correspondence with anisotropic convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 3189–3197.

Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., and Vandergheynst, P. (2017). Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42.

Bruna, J., Zaremba, W., Szlam, A., and Lecun, Y. (2014). Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations (ICLR2014), CBLS, April 2014*.

Budninskiy, M., Yin, G., Feng, L., Tong, Y., and Desbrun, M. (2019). Parallel transport unfolding: A connection-based manifold learning approach. *SIAM Journal on Applied Algebra and Geometry*, 3(2):266–291.

Cohen, T. and Welling, M. (2016a). Group equivariant convolutional networks. In *International conference on machine learning*, pages 2990–2999.

Cohen, T. S. and Welling, M. (2016b). Steerable cnns. *arXiv preprint arXiv:1612.08498*.

Crane, K., Weischedel, C., and Wardetzky, M. (2013). Geodesics in heat: A new approach to computing distance based on heat flow. *ACM Transactions on Graphics (TOG)*, 32(5):152.

Defferrard, M., Bresson, X., and Vandergheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852.

Dhillon, I. S., Guan, Y., and Kulis, B. (2007). Weighted graph cuts without eigenvectors a multilevel approach. *IEEE transactions on pattern analysis and machine intelligence*, 29(11).

Dieleman, S., De Fauw, J., and Kavukcuoglu, K. (2016). Exploiting cyclic symmetry in convolutional neural networks. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 1889–1898. JMLR.org.

Fasel, B. and Gatica-Perez, D. (2006). Rotation-invariant neoperceptron. In *18th International Conference on Pattern Recognition (ICPR'06)*, volume 3, pages 336–339. IEEE.

Fey, M. and Lenssen, J. E. (2019). Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

Fong, C. (2015). Analytical methods for squaring the disc.

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

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural message passing for quantum chemistry. *CoRR*, abs/1704.01212.

Henaff, M., Bruna, J., and LeCun, Y. (2015). Deep convolutional networks on graph-structured data. *CoRR*, abs/1506.05163.

Herholz, P. and Alexa, M. (2019). Efficient computation of smoothed exponential maps. In *Computer Graphics Forum*. Wiley Online Library.

Hinton, G. E., Krizhevsky, A., and Wang, S. D. (2011). Transforming auto-encoders. In *International Conference on Artificial Neural Networks*, pages 44–51. Springer.

Hu, R., Savva, M., and van Kaick, O. (2018). Functionality representations and applications for shape analysis. In *Computer Graphics Forum*, volume 37, pages 603–624. Wiley Online Library.

Huang, H., Kalogerakis, E., Chaudhuri, S., Ceylan, D., Kim, V. G., and Yumer, E. (2018). Learning local shape descriptors from part correspondences with multiview convolutional networks. *ACM Transactions on Graphics (TOG)*, 37(1):6.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167.

Jacobsen, J.-H., van Gemert, J., Lou, Z., and Smeulders, A. W. (2016). Structured receptive fields in cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2610–2619.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Kipf, T. N. and Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907.

Kuehnel, W. (2005). *Differential Geometry: Curves - Surfaces - Manifolds*. AMS.

Laptev, D., Savinov, N., Buhmann, J. M., and Pollefeys, M. (2016). Ti-pooling: transformation-invariant pooling for feature learning in convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 289–297.

Larochelle, H., Erhan, D., Courville, A., Bergstra, J., and Bengio, Y. (2007). An empirical evaluation of deep architectures on problems with many factors of variation. In *Proceedings of the 24th international conference on Machine learning*, pages 473–480. ACM.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436.

Litany, O., Remez, T., Rodolà, E., Bronstein, A. M., and Bronstein, M. M. (2017). Deep functional maps: Structured prediction for dense shape correspondence. In *ICCV*, pages 5660–5668.

Liu, K., Wang, Q., Driever, W., and Ronneberger, O. (2012). 2d/3d rotation-invariant detection using equivariant filters and kernel weighted mapping. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 917–924. IEEE.

Marcos, D., Volpi, M., and Tuia, D. (2016). Learning rotation invariant convolutional filters for texture classification. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2012–2017. IEEE.

Maron, H., Galun, M., Aigerman, N., Trope, M., Dym, N., Yumer, E., Kim, V. G., and Lipman, Y. (2017). Convolutional neural networks on surfaces via seamless toric covers. *ACM Trans. Graph*, 36(4):71.

Masci, J., Boscaini, D., Bronstein, M., and Vandergheynst, P. (2015). Geodesic convolutional neural networks on riemannian manifolds. In *Proceedings of the IEEE international conference on computer vision workshops*, pages 37–45.

Melvær, E. L. and Reimers, M. (2012). Geodesic polar coordinates on polygonal meshes. In *Computer Graphics Forum*, volume 31, pages 2423–2435. Wiley Online Library.

Monti, F., Boscaini, D., Masci, J., Rodola, E., Svoboda, J., and Bronstein, M. M. (2017). Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proc. CVPR*, volume 1, page 3.

O'neill, B. (2006). *Elementary differential geometry*. Elsevier.

Oyallon, E. and Mallat, S. (2015). Deep roto-translation scattering for object classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2865–2873.

Pan, H., Liu, S., Liu, Y., and Tong, X. (2018). Convolutional neural networks on 3d surfaces using parallel frames. *CoRR*, abs/1808.04952.

Petersen, P., Axler, S., and Ribet, K. (2006). *Riemannian geometry*, volume 171. Springer.

Poulenard, A. and Ovsjanikov, M. (2018). Multi-directional geodesic neural networks via equivariant convolution. In *SIGGRAPH Asia 2018 Technical Papers*, page 236. ACM.

Qi, C. R., Su, H., Mo, K., and Guibas, L. J. (2017a). Pointnet: Deep learning on point sets for 3d classification and segmentation. *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 1(2):4.

Qi, C. R., Yi, L., Su, H., and Guibas, L. J. (2017b). Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems*, pages 5099–5108.

Ron, D., Safro, I., and Brandt, A. (2011). Relaxation-based coarsening and multiscale graph organization. *Multiscale Modeling & Simulation*, 9(1):407–423.

Schmidt, R., Grimm, C., and Wyvill, B. (2006). Interactive decal compositing with discrete exponential maps. In *ACM Transactions on Graphics (TOG)*, volume 25, pages 605–613. ACM.

Schmidt, R. and Singh, K. (2010). Meshmixer: an interface for rapid mesh composition. In *ACM SIGGRAPH 2010 Talks*, page 6. ACM.

Schonsheck, S. C., Dong, B., and Lai, R. (2018). Parallel transport convolution: A new tool for convolutional neural networks on manifolds. *arXiv preprint arXiv:1805.07857*.

Sharp, N., Crane, K., et al. (2019a). geometry-central. www.geometry-central.net.

Sharp, N., Soliman, Y., and Crane, K. (2019b). Navigating intrinsic triangulations. *ACM Trans. Graph.*, 38(4).

Sharp, N., Soliman, Y., and Crane, K. (2019c). The vector heat method. *ACM Trans. Graph.*, 38(3).

Shuman, D. I., Faraji, M. J., and Vandergheynst, P. (2016). A multiscale pyramid transform for graph signals. *IEEE Transactions on Signal Processing*, 64(8):2119–2134.

Singer, A. and Wu, H.-T. (2012). Vector diffusion maps and the connection laplacian. *Communications on pure and applied mathematics*, 65(8):1067–1144.

Thomas, N., Smidt, T., Kearnes, S. M., Yang, L., Li, L., Kohlhoff, K., and Riley, P. (2018). Tensor field networks: Rotation- and translation-equivariant neural networks for 3d point clouds. *CoRR*, abs/1802.08219.

Verma, N., Boyer, E., and Verbeek, J. (2018). Feastnet: Feature-steered graph convolutions for 3d shape analysis. In *CVPR 2018-IEEE Conference on Computer Vision & Pattern Recognition*.

Von Luxburg, U. (2007). A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416.

Wang, Y., Sun, Y., Liu, Z., Sarma, S. E., Bronstein, M. M., and Solomon, J. M. (2018). Dynamic graph CNN for learning on point clouds. *CoRR*, abs/1801.07829.

Worrall, D. E., Garbin, S. J., Turmukhambetov, D., and Brostow, G. J. (2017). Harmonic networks: Deep translation and rotation equivariance. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5028–5037.

Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., and Xiao, J. (2015). 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920.

Xu, K., Kim, V. G., Huang, Q., Mitra, N., and Kalogerakis, E. (2016). Data-driven shape analysis and processing. In *SIGGRAPH ASIA 2016 Courses*, page 4. ACM.

TU Delft