A Journey Through Surface Convolutions

Anonymous authors
Paper under double-blind review

Abstract

Surface data appears in a wide range of machine learning tasks. Among other objectives, Geometric deep learning generalizes Euclidean CNNs to two-dimensional Riemannian manifolds, allowing to learn patterns of data along curved surfaces. While the literature presents numerous surface convolutions, existing comparisons often vary both the convolution and the underlying surface charting algorithm, complicating the isolation of the source of performance differences. Furthermore, unlike in Euclidean CNNs, the size of the convolution neighborhood is not implied by the template size but manually set in the form of a chart radius. Finding appropriate chart radii has received little attention and thus remains poorly understood. To address these challenges, we (i) provide a structured survey of surface convolutions and (ii) reformulate several approaches into a unified discretization notation. Using this notation, we (iii) develop a charting-method agnostic surface CNN framework, allowing to isolate downstream effects of charting algorithms from those of the convolution itself, and to systematically study the impact of chart radii. We provide a Python library and (iv) use it to conduct an ablation study encompassing 420 surface CNN configurations. We find that varying the chart radius gives rise to statistically significant performance changes in nearly all of our experiments. Additionally, we find statistically significant effects of charting algorithms in local prediction tasks. These findings suggest that both the choice of the charting algorithm and chart radius constitute important experimental factors. Future comparisons among surface CNNs should thus control charting effects by using a common charting algorithm and treat chart radii as tunable hyperparameters.

1 Introduction

As a subcategory of machine learning, geometric deep learning (Bronstein et al., 2021) studies how to leverage underlying domain information on which data is distributed. One particular problem domain is represented by learning from data that is distributed along curved 2D manifolds in a 3D ambient space.

This kind of 3D data plays an important role in a variety of applications across many interdisciplinary subjects: *Computer graphics* tasks include estimating point-correspondences among 3D shapes (Bogo et al., 2014) or finding shapes in 3D retrieval tasks (Yacoub et al., 2025). *Machine learning-based weather predictions* are often modeled on top of a discretized sphere that represents the earth (Lam et al., 2023). Here, learning tasks include estimating the temperature (Esteves et al., 2023) or more generally atmospheric dynamics (Bonev et al., 2023; 2025). Learning protein functions from their surface geometry represents a task in the *biological domain* (Sverrisson et al., 2021). The *medical domain* can benefit from learning from surfaces by, e.g., estimating wall shear stress on three-dimensional artery models for cardiovascular-disease diagnosis and prognosis (Suk et al., 2024) or for cortical parcellation (Zhu et al., 2025). *Self-driving cars* need to be able to understand their surroundings via semantic scene segmentation (Behley et al., 2019). *Robotic applications* include tasks such as learning to reconstruct incomplete surfaces from tactile sensing (Comi et al., 2024).

Approaches to learning from surface data, whether one uses volumetric approaches, graph neural networks, point set approaches or networks specifically designed for surfaces, can vary substantially among existing work. In this work, we put our focus on *isometry invariant, spatial surface convolutions*. These convolutions generalize the Euclidean convolution to two-dimensional Riemannian manifolds. Here, the main theoretical task is to describe how local surface features are extracted from the manifold and correlated with a trainable

template in a common integration domain. Depending on what aspects should be emphasized, existing work varies in its emphasis on practical implementation and theoretical rigor. To provide an overview, we present a survey on surface CNNs which (a) structures them by how they tackle the so-called *gauge-ambiguity problem*, a fundamental problem in surface learning arising from the fact that no global coordinate system is available, and (b) how they compute local coordinate systems for discretized surfaces in practice.

Different surface convolutions learn different sets of weights and thus imply distinct learnable function classes. From a machine learning perspective, differences in downstream task performances should primarily be attributed to changes of these learnable functions. However, the way in which features are extracted from the surface and carried into the convolution *also* varies among proposed approaches. In this matter, the vast majority of existing surface convolutions relies on surface charting algorithms, i.e., methods to compute local coordinate systems on the surface. *Existing comparisons among surface CNNs often change both the charting algorithm and the convolution simultaneously.* This is problematic because different charting algorithms can introduce distinct confounding factors into model training due to varying geometric distortions caused by dissimilar equations for computing geodesic distances and directions. Hence, reported performance differences among surface CNNs are not necessarily caused by differences in the convolutions alone.

To isolate performance changes caused by varying convolution definitions from effects of charting algorithms, we reformulate existing surface convolutions into a *unified notation*. Within this notation, all considered convolutions rely on the same surface feature extraction mechanism, independent of the choice of a particular charting algorithm, and thus differ only in their trainable parameters. The benefits of this notation are two-fold. (a) By highlighting the defining differences among existing surface convolutions, the unified notation reveals an opportunity to combine ideas from two prior approaches. The resulting convolution outperforms all other considered convolutions in one of our benchmarks (cf. Section 7.1). (b) By abstracting from these defining differences, we define a *charting-algorithm agnostic surface CNN framework*. This framework enables us to conduct a systematic ablation study on the downstream effects of commonly used charting algorithms by training a fixed set of surface CNNs once per charting algorithm and comparing their performance.

In addition to selecting a charting algorithm, one must specify a maximum allowed chart radius per surface point. It describes how large the neighborhood into which we position the template of the surface CNN can become. Choosing it too small can omit a substantial amount of helpful information, resulting in underperforming surface CNNs. On the other hand, choosing it too large causes geometric instabilities during chart computation, since directions for neighboring surface points cannot be assigned uniquely anymore. The theoretical upper bound that describes this is called the *injectivity radius*. Despite being a fundamental design choice, the impact of the chart radius on the downstream performance of surface CNNs has received little attention in the literature. Thus, we additionally analyze the effects of varying chart radii side-by-side with our ablations on the downstream effects of charting algorithms. Altogether, we train 420 surface CNNs covering different convolutions and charting algorithm configurations in our ablation study.

In summary, our contributions encompass the following:

1. We conduct a survey to structure existing work on surface convolutions by how they approach key-problems in surface learning.

2. We reformulate existing surface convolution definitions into a unified notation, based on which we:

   (a) propose a novel surface convolution that emerges from the comparison of two existing surface convolutions.

   (b) provide a charting-method agnostic surface CNN framework, isolating convolution effects from effects originating from the mechanism that extracts surface features

3. We provide a Python library for the introduced surface CNN framework to foster future research on surface convolutions.

4. Using our framework, we conduct an ablation study on the downstream effects of different charting algorithms and varying chart radii. We find that both the selected charting algorithm and the template radius are critical parameters in surface learning.

In Section 2, we provide a structured survey on the literature of surface convolutions. Based on that survey, we phrase three research questions in Section 3 in alignment with our contributions. At the end of Section 3, we provide an overview of the remaining sections of this manuscript, describing our approach to answering the research questions.

2 A Structured Overview of Approaches to Surface Convolutions

Surface convolutions. The subject of surface convolutions is a broad field with many diverse approaches to learn patterns from data distributed along surfaces. Thematically related to this work are approaches that implement *isometry-invariant, spatial surface convolutions on two-dimensional Riemannian manifolds* whose definitions are independent of the surface's discretization into polygonal meshes or point clouds. For their definition, we consider a two-dimensional Riemannian manifold M, i.e., a surface, as well as a surface signal function $s : M \to \mathbb{R}^n$ whose patterns we aim to learn. Examples for s include shape texture, heatmaps, climate data or local shape descriptors. In order to learn these patterns, we make use of intrinsic properties of M. That is, information that can be derived with the Riemannian metric, in particular geodesic distances and angular directions along M. These quantities are invariant under isometries, i.e., metric-preserving transformations such as rotations and translations of M in an ambient space, and therefore ideal for describing relative locations between features on the surface, which is essential for implementing the correlation-with-template idea.

Convolutions that do not explicitly define their operation on two-dimensional Riemannian manifolds are outside the scope of this work. For example, *volumetric approaches* (Maturana & Scherer, 2015; Wu et al., 2015; Qi et al., 2016) leverage the Euclidean ambient space, *unordered sets of points* approaches (point clouds) focus on permutation-invariant operations on sets (Qi et al., 2017a;b; Li et al., 2018; Wu et al., 2019) rather than interpreting the surface as a manifold, and *spectral approaches* (Yi et al., 2017; Kostrikov et al., 2018; Defferrard et al., 2016) operate in the frequency domain instead of spatially on the manifold. Lastly, *mesh-based approaches* (Bouritsas et al., 2019; Gong et al., 2019; Hanocka et al., 2019; Wang et al., 2022; Hu et al., 2022; Ludwig et al., 2023) define their convolution by specifically relying on a fixed surface discretization and are thus also not considered in the course of this work.

Preliminary: the gauge-ambiguity problem. In analogy to convolutions on images, the majority of surface convolutions implements the correlation-with-template idea. In difference to convolutions on images, surface convolutions cannot rely on a global coordinate system along which the template can be aligned. Instead, we compute one local coordinate system per point on the surface which, in terms of polar coordinates, involves selecting a $0°$-direction. This direction together with its orthogonal direction determine one possible basis for the local coordinate system in the considered surface point, also referred to as the *gauge*. However, note that multiple possible gauges exist as the selection of the $0°$-direction is arbitrary.

The gauge determines how weights and surface features are paired during correlation. Given that these pairings change with varying gauges, the convolution yields different results depending on the selected gauge. This makes the convolution gauge-dependent. Considering that infinitely many equally valid gauges exist, each of which yields different convolution results, it is unclear which gauge should be selected. This issue is referred to as the *gauge-ambiguity problem*.

In the following, we distinguish related work between two classes of approaches (cf. Figure 1): *gauge-selective surface convolutions*, which attempt to resolve the gauge-ambiguity problem by selecting a preferred gauge, and *gauge-equivariant surface convolutions*, which ensure that the convolution result rotates according to the orientation of the gauge.

Gauge selective surface convolutions. In practice, continuous surfaces are typically discretized into polygonal meshes or point clouds. Signals along these discretized surfaces are only available at a finite set of vertices. Consequently, a fundamental design choice in the implementation of any surface convolution concerns the decision of whether the implementation interpolates existing signals in between of available surface points or merely takes signals into account that are located at available surface points.

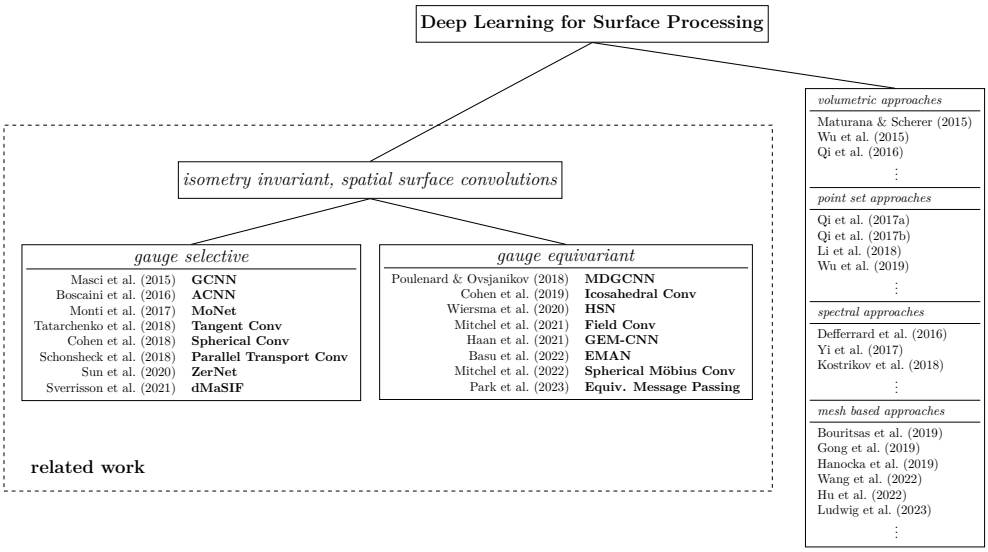


Figure 1: A schematic overview of deep learning approaches for two-dimensional surfaces.

The first set of approaches that we consider follows the former idea: create interpolated surface signals to obtain signal information at any surface location, potentially in between surface points.

Early approaches to this implementation idea are surface convolutions that build on the so-called *patch operator* (Masci et al., 2015; Boscaini et al., 2016; Monti et al., 2017). Motivated by local shape descriptors (Kokkinos et al., 2012), the patch operator extracts surface features and interpolates these in a polar grid of soft-bins, referred to as a *surface patch*. Binning weights are either defined to be Gaussian weights (Masci et al., 2015), heat-kernels (Boscaini et al., 2016) or learned (Monti et al., 2017). By aligning a trainable template with the surface patch, learnable weights are paired with patch bins. These are multiplied and aggregated to effectively implement the correlation-with-template idea on curved surfaces. This approach *selects a gauge* during patch construction: the binning operation requires a preceding charting algorithm to compute local coordinates around the convolution center which inevitably selects a $0°$-direction. The gauge-ambiguity problem can then be resolved in different ways: Masci et al. (2015) compute multiple convolutions using different template orientations and subsequently select the maximum result. This mechanism is called *angular max-pooling*. Boscaini et al. (2016) suggest to align the gauge of a local chart with the principal curvature at the convolution center and compute convolutions only with respect to this gauge. Monti et al. (2017) propose a framework that allows to obtain the approaches of Masci et al. (2015) and Boscaini et al. (2016) as particular instances and thus leaves the choice of angular max-pooling or alignment with principal curvature open.

Similarly to patch operator approaches, Tatarchenko et al. (2018) interpolate given signals at available surface points to create a continuous signal. To this end, they project neighboring surface points into tangent planes and interpolate their carried signals into so-called *tangent images* using either nearest neighbor estimations or Gaussian mixture models. Tatarchenko et al. (2018) use local PCA to estimate tangent planes, where the gauge is determined by the Eigenvector associated to the largest Eigenvalue. Their convolution results are thus always given with respect to the direction of the largest Eigenvalue.

Another alternative for creating a continuous signal from the given signals at available surface points is proposed by Sun et al. (2020). They define their surface convolution to learn coefficients for a linear combination of normalized Zernike polynomials. The Zernike polynomials describe the learnable weights and the surface signals as continuous functions parameterized in local coordinates. The Zernike decomposition of the surface signal from available surface points requires surface charts, which Sun et al. (2020) compute using the DGPC algorithm (Melvær & Reimers, 2012). This algorithm randomly *selects a gauge* for each surface point by construction. The gauge-ambiguity problem is resolved by using angular max-pooling.

Other gauge-selective approaches implement surface convolutions by only relying on signals at available surface points. Schonsheck et al. (2018) move a trainable template into each vertex of the discretized surface via parallel transport (cf. Section 4.1). Their surface convolution then computes the mass-matrix-weighted dot-products between transported weights and surface features from locally available points. To prepare their convolution, Schonsheck et al. (2018) determine local reference frames in all mesh vertices, whose gauges are aligned with the gradient field over precomputed geodesic distances. Their convolution results are hence given with respect to this gradient direction.

In the context of learning protein functions, Sverrisson et al. (2021) define their surface convolution to be the weighted sum of chemical features. Weights depend on their relative position to the convolution center. These positions are described using local reference frames whose gauges are aligned with the gradient of estimated electrostatic potentials along the protein, thereby determining the local gauge in any point of the surface. Thus, convolution results are given with respect to the direction of electrostatic potentials.

Exemplarily for methods that circumvent the explicit definition of local gauges, Cohen et al. (2018) propose surface convolutions specifically for spheres as the integral over dot-products of weights and features, using a rotation invariant integration measure. This convolution behaves equivariant under rotations of the sphere. Regarding template alignment, the integration measure allows to directly associate points $\boldsymbol{x} \in S^2 \subset \mathbb{R}^3$ from the sphere to features and weights. Hence, no local gauges need to be defined for associating weights to features. This convolution relies on using the *global* coordinate system of the $\mathbb{R}^3$ ambient space, which is why we categorize it as a gauge-selective approach.

Gauge equivariant surface convolutions. Instead of returning an embedding vector w.r.t. a preferred gauge, *gauge-equivariant surface CNNs* rotate their output embedding vector according to the orientation of an arbitrary gauge. This is a consequence of satisfying the *gauge equivariance* constraint, which we describe more formally in Section 4.

Multi-directional geodesic convolution neural networks (MDGCNNs) are chronologically one of the first surface convolution approaches that recognize the gauge-ambiguity problem. To counteract this ambiguity, they extend geodesic convolutions (Masci et al., 2015) to map *directional functions* to directional functions. These return different signals depending on the direction of neighboring surface points. Directions are described w.r.t. a local gauge, which Poulenard & Ovsjanikov (2018) determine using the DGPC algorithm (Melvær & Reimers, 2012). In practice, directional functions are parameterized by an angular coordinate. Poulenard & Ovsjanikov (2018) show that rotating the gauge yields the same convolution result as adding the rotation onto the input angle. However, since MDGCNNs return directional functions rather than embedding vectors whose orientation depends on the local gauge, they are not gauge-equivariant in the sense described above.

Contrary to this, *Harmonic Surface Networks* (HSNs) introduced by Wiersma et al. (2020), are gauge-equivariant in the sense described above. They define their convolution as dot a product over complex features. Wiersma et al. (2020) algebraically proof gauge-equivariance by factorizing gauge rotations, expressed as the multiplication of two exponential terms, outside the convolution. This leaves a rotation term multiplied onto the convolution that uses the original local gauge, showing that the convolution commutes with gauge-rotations. Thus, it can be considered gauge-equivariant. While not explicitly shown, the same argument holds true for *field convolutions* (Mitchel et al., 2021) which also implement complex-valued surface convolutions. However, field convolutions differ from HSNs in that they learn sums of angular frequencies instead of radial profile matrices and a phase offset.

Cohen et al. (2019) introduce the theory of equivariance under symmetry transformations to surface convolutions (cf. Section 4.1). In summary, previously assumed stacks of scalar signals are reconsidered to be multi-dimensional feature vectors that (a) are represented in local fiber bases and (b) transform according to a group representation under gauge rotations (Cohen & Welling, 2017). Due to varying bases among local fibers, neighboring feature vectors have to be parallel transported to the convolution center. This ensures that all features are represented in the same basis prior to feature aggregation. During aggregation, the feature vectors are multiplied with a kernel that satisfies the *rotation-steerable kernel constraint* (Weiler et al., 2018). Due to satisfying this constraint, the surface convolution commutes with gauge rotations and hence is gauge-equivariant. Cohen et al. (2019) implement this surface convolution using an Icosahedron as

Table 1: Charting methods used in surface CNNs to compute local surface coordinates.

		FMM	DGPC	HDM	Tangent Plane Projections	Predefined
Masci et al. (2015)	**GCNN**	✓				
Monti et al. (2017)	**MoNet**	✓				
Schonsheck et al. (2018)	**Parallel Transport Conv**	✓				
Poulenard & Ovsjanikov (2018)	**MDGCNN**		✓			
Sun et al. (2020)	**ZerNet**		✓			
Boscaini et al. (2016)	**ACNN**			✓		
Wiersma et al. (2020)	**HSN**			✓		
Mitchel et al. (2021)	**Field Convolutions**			✓		
Tatarchenko et al. (2018)	**Tangent Convolutions**				✓	
Haan et al. (2021)	**GEM-CNN**				✓	
Sverrisson et al. (2021)	**dMaSIF**				✓	
Basu et al. (2022)	**EMAN**				✓	
Park et al. (2023)	**Equiv. Message Passing**				✓	
Cohen et al. (2018)	**Spherical Conv**					✓
Cohen et al. (2019)	**Icosahedral Conv**					✓
Mitchel et al. (2022)	**Spherical Möbius Conv**					✓

a specific example surface. Haan et al. (2021) implement it for arbitrary surfaces as an extension of graph neural networks. Their network-design is referred to as *gauge-equivariant mesh CNN* (GEM-CNN). Traditional isotropic kernels used by graph neural networks are replaced with directionally sensitive, anisotropic kernels that satisfy the rotation-steerable kernel constraint. The features associated to the one-hop neighbors are parallel transported to the convolution center and subsequently aggregated in a self-connection & neighborhood aggregation scheme.

Basu et al. (2022) extend GEM-CNNs by attention coefficients which are invariant under gauge transformations and thereby introduce *equivariant mesh attention networks* (EMANs). Park et al. (2023) proposes gauge-equivariant message passing as an extension to regular message passing in graph neural networks. To this end, they embed node and edge features using the convolution of GEM-CNNs.

Mitchel et al. (2022) propose spherical convolutions that are equivariant towards Möbius transformations of the signal on the sphere. As Möbius transformations can represent rotations of the features defined on the sphere, this convolution can be considered gauge-equivariant.

Surface charting methods. A common methodological requirement among the vast majority of discussed surface convolutions is the computation of local surface charts. A surface chart describes local surface coordinates around the convolution center w.r.t. a selected gauge. The choice of the charting algorithm, however, varies across approaches.

Some approaches, like Cohen et al. (2018), Cohen et al. (2019) or Mitchel et al. (2022), define their convolution *solely for one particular shape*. This either allows for using a fixed set of predefined charts, a closed form formulation of the exponential map, i.e., a function associating tangent vectors to manifold points and thereby allowing to locally associate coordinates to the surface, or to use an integration measure for the considered shape. *Shape-agnostic approaches* use one out of the following four surface charting algorithms.

Masci et al. (2015) and Monti et al. (2017) compute local surface charts using the *Fast Marching Method* (FMM) (Kimmel & Sethian, 1998) and outward-ray shooting (Kokkinos et al., 2012) to determine radial and angular bins. Schonsheck et al. (2018) also use FMM to compute distances on the shape. However, instead of outward-ray shooting, they combine gradient fields over the geodesic distances provided by FMM, mesh triangle-normals and orthonormal vector fields to create local reference frames which parameterize the surface locally.

The *Heat Diffusion Method* (HDM) (Crane et al., 2017) leverages the relation between distance functions and shortly diffused heat kernels (Varadhan, 1967). The surface convolutions of Wiersma et al. (2020) and Mitchel et al. (2021) use the Vector Heat Method (Sharp et al., 2019) which completes the geodesic distance computation of HDM to an approach that additionally yields angular coordinates. Boscaini et al. (2016)

combine shortly diffused anisotropic heat kernels and local reference frames, created from mesh triangle-normals and principal curvature directions, to parameterize the surface locally.

The *Discrete Geodesic Polar Coordinates* algorithm (DGPC) of Melvær & Reimers (2012) is used by Poulenard & Ovsjanikov (2018) and Sun et al. (2020). In contrast to FMM and HDM, DGPC computes directional coordinates via angular interpolation in addition to geodesic distances. Thus, unlike in FMM and HDM, no additional step is required.

Approximating local surface charts by entirely relying on *Tangent Plane Projections* (TP) represents another popular approach (Tatarchenko et al., 2018; Haan et al., 2021; Basu et al., 2022; Park et al., 2023). For this, local reference frames on surface points are estimated by, e.g., local distance weighted PCA as done by local shape descriptors (Salti et al., 2014). While the Eigenvector associated to the smallest Eigenvalue determines the point normal, the tangent planes are spanned by the Eigenvectors associated to the two largest Eigenvalues. These tangent planes are then used to compute tangent plane projections of neighboring surface points. Sverrisson et al. (2021) additionally weigh the Euclidean distance between points by differences in point normal orientations to improve geodesic distances estimation.

Recently, new work on how to efficiently compute geodesic distances on triangle meshes has emerged (Verninas et al., 2026). However, it has not yet been adopted by existing surface convolution approaches.

We summarize our literature findings on how surface convolution approaches compute local surface coordinates in Table 1 according to their main methodological characteristics. The diversity of charting method usage and their combination with various possible approaches to implement surface convolutions motivates our following research questions for this manuscript.

3 Research questions

Table 1 shows that each surface CNN has been evaluated using only a single charting algorithm. While theoretically any discussed algorithm aims to provide local charts, they differ in their approach to compute distances and angles in practice, which may introduce different geometric distortions. This raises the question whether the performance of surface CNNs is invariant towards the choice of the charting algorithm. Specifically, we suspect that reported performance differences among surface CNNs may partly originate from differently selected charting algorithms. This motivates our first research question: **Are the downstream performances of surface CNNs affected by their chosen charting algorithm? (RQ1)**

In image CNNs, a fundamental design choice concerns the size of the template. It is typically set to some $n \times n$ window that directly determines (a) the size of the template and (b) the size of the considered neighborhood on the image. These $n \times n$ windows are possible due to the regular grid-like structure of images, which is not present on discretized surfaces. That is, while most surface CNNs adopt the correlation-with-template idea, the size of the template is decoupled from the size of the neighborhood it is placed into.

Instead of implying the extent of neighborhoods via the template size, we explicitly compute them using charting algorithms that require a maximum radius to be know beforehand. Once we know the neighborhood sizes, we can configure template sizes relative to all available neighborhood sizes, or more specifically, in terms of all chart radii. But how do we determine how large each chart radius should be?

Theoretically, the radius of a valid neighborhood around the convolution center has an upper bound, called the *injectivity radius*. This injectivity radius depends on the local shape geometry and thus varies for different surface points. Efficiently estimating accurate injectivity radii for all vertices in a set of arbitrary shapes represents a problem that has not yet been discussed in the surface convolution literature. So far, the literature tends to use small neighborhoods such as the one-hop neighborhood of a mesh vertex. However, this can omit a substantial amount of meaningful information for downstream tasks and thus leaves room for improvement. In this work, we want to systematically study the effect of chart radii and thus raise the research question: **How does the chart radius affect the performance of surface CNNs? (RQ2)**

The majority of existing surface convolutions aim to implement the correlation-with-template idea. That is, the convolution puts a template into a surface chart and computes an inner product between template weights and locally available surface features. The discrete implementation of that inner product, however, can differ

substantially across approaches. Most notably, once all charts are computed, the surface convolution needs to gather surface features using these charts. To this end, some approaches use feature vectors directly associated with surface points. Others rely on interpolations of surface features in between surface points. Varying confounding factors introduced by different implementations make it difficult to isolate whether performance differences are caused by theoretical considerations or practical implementations. This motivates our last research question for this work: **How can existing surface convolutions be reformulated into a unified, charting-algorithm-agnostic discretization framework? (RQ3)**

Having outlined the relevant literature on isometry-invariant, spatial surface convolutions and highlighted the research questions addressed in this manuscript, we now provide an overview over the remaining parts of this work.

To establish a common theoretical basis on surface convolutions, we present an introduction to the theory of surface convolutions in the next section (Section 4). Subsequent to discussing the theory on surface convolutions, we shortly point out the theoretical common denominators and differences among the charting algorithms commonly used for computing surface convolutions in Section 5. Both theoretical introduction sections (Section 4 and 5) represent the theoretical foundation for our work on RQ3, exploring a unified implementation framework for surface convolutions in Section 6. Using this framework as a means to create a controlled discretization setting, we then conduct an ablation study in Section 7 to investigate both RQ1 and RQ2. Limitations of this work are discussed in Section 8. We draw conclusion regarding our research questions in Section 9.

4 An Introduction to Surface Convolutions

From Euclidean spaces to manifolds. In a Euclidean domain, convolutions are defined as an inner product between a signal $s : \mathbb{R}^m \to \mathbb{R}$ and a template $t : \mathbb{R}^m \to \mathbb{R}$ in a point $\boldsymbol{u} \in \mathbb{R}^m$:

$$(s * t)(\boldsymbol{u}) = \int_{\mathbb{R}^m} s(\boldsymbol{v}) t(\boldsymbol{u} - \boldsymbol{v}) d\boldsymbol{v} \ . \tag{1}$$

To compute a convolution, a template t is centered at $\boldsymbol{u}$, and relative directions $\boldsymbol{u} - \boldsymbol{v}$ determine how template weights are paired with signal values. In a Euclidean domain, such relative directions are described by translation vectors $\boldsymbol{u} - \boldsymbol{v}$ that are defined by pairs of locations $\boldsymbol{u}$ and $\boldsymbol{v}$. That is, we induce a direction by a pair of locations. In general Riemannian manifolds M^n, however, local directions cannot be inferred from pairs of locations on the manifold. Instead, we rely on *tangent vectors* $\boldsymbol{y} \in T_{\boldsymbol{u}} M^n$ to describe directions locally around the convolution center $\boldsymbol{u} \in M^n$.

Since tangent vectors $\boldsymbol{y}$ are coordinate-free in general Bredon (1993), a *gauge* $\omega_{\boldsymbol{u}} : \mathbb{R}^m \to T_u M^n$ must be chosen for the tangent space to represent $\boldsymbol{y}$ in local coordinates. Hence, the gauge can intuitively be thought of as a coordinate frame put in a tangent space $T_{\boldsymbol{u}} M^n$ (Cohen et al., 2019).

Although tangent vectors represent a helpful tool to describe a local notion of direction, they do not represent the elements of the manifold on which the signal $s : M^n \to \mathbb{R}$ is defined that shall be learned. The *exponential map* $\exp_{\boldsymbol{u}} : T_{\boldsymbol{u}} M^n \to M^n$ offers a way to describe the target point on the manifold when moving into the direction of a tangent vector. It is a *local* diffeomorphism, i.e., a differentiable bijection with differentiable inverse, limited by the *injectivity radius* around $\boldsymbol{u}$, that maps tangent vectors onto points of the manifold.

Cohen et al. (2019) combine the gauge, the exponential map and the signal function into the *signal-pullback* $[s \circ \exp_{\boldsymbol{u}} \circ \omega_{\boldsymbol{u}}]$ *at* $\boldsymbol{u}$ (cf. Figure 2), which allows to read s when moving relative to a base point $\boldsymbol{u}$ in order to define the surface convolution:

Definition 4.1 (Cohen et al. (2019): Surface convolution)**.** *Let M be a 2-dimensional Riemannian manifold, $B_R(\mathbf{0})$ the disk around $\mathbf{0}$ with radius R, $\omega_{\boldsymbol{u}} : B_R(\mathbf{0}) \to T_{\boldsymbol{u}} M$ a selected gauge for $T_{\boldsymbol{u}} M$, and $exp_{\boldsymbol{u}} : T_{\boldsymbol{u}} M \to M$ the exponential map at $\boldsymbol{u}$. The* surface convolution *of a signal $s : M \to \mathbb{R}$ on M with a template $t : \mathbb{R}^m \to \mathbb{R}$ in point $\boldsymbol{u} \in M$ is defined as:*

$$(s * t)(\boldsymbol{u}; \omega_{\boldsymbol{u}}) = \int_{B_R(\mathbf{0})} t(\boldsymbol{v}) \ [s \circ exp_{\boldsymbol{u}} \circ \omega_{\boldsymbol{u}}](\boldsymbol{v}) \ d\boldsymbol{v} \ .$$

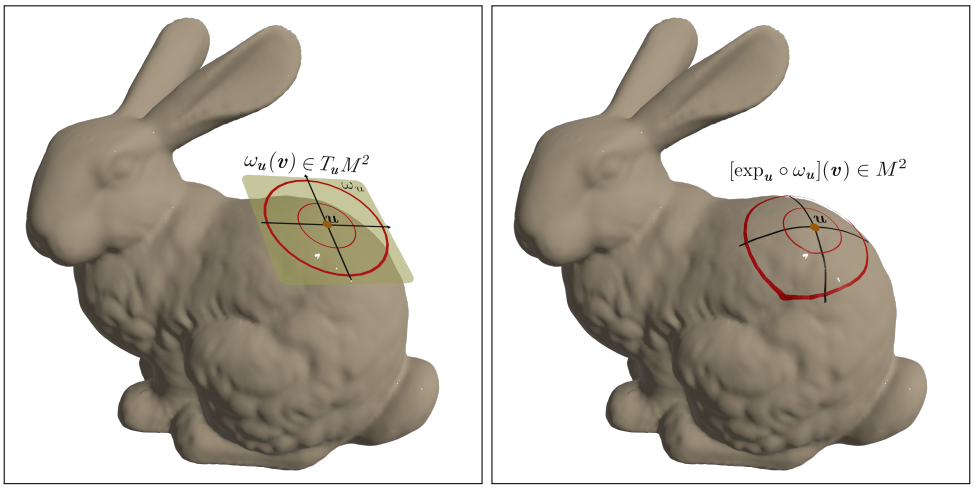


Figure 2: Exemplary illustration of the signal-pullback at $\boldsymbol{u}$. **[Left]** To describe relative positions around $\boldsymbol{u} \in M$ we choose a gauge $\omega_{\boldsymbol{u}} : B_R(\mathbf{0}) \to T_{\boldsymbol{u}}M$ and retrieve tangent vectors $\boldsymbol{y} \in T_{\boldsymbol{u}}M$. **[Right]** Tangent vectors $\omega_{\boldsymbol{u}}(\boldsymbol{v}) \in T_{\boldsymbol{u}}M$ are mapped into the surface via the exponential map $\exp_{\boldsymbol{u}}(\omega_{\boldsymbol{u}}(\boldsymbol{v})) \in M$. Eventually, a neighboring signal to $\boldsymbol{u}$ is extracted using the signal-pullback $[s \circ \exp_u \circ \omega_u](\boldsymbol{v})$.

The gauge-ambiguity problem. In our survey, we introduced the gauge-ambiguity problem at a conceptual level to categorize existing surface convolutions according to how they resolve it (cf. Section 2). Using Definition 4.1, we can explain more formally why it arises.

In order to compute a surface convolution, we need to associate coordinates from $B_R(\mathbf{0})$ with tangent vectors in $T_{\boldsymbol{u}}M$ via a gauge $\omega_{\boldsymbol{u}}$. The gauge $\omega_{\boldsymbol{u}}$ represents a basis for the local tangent space $T_{\boldsymbol{u}}M$ and thus can be interpreted as a *local coordinate system*. It determines the orientation, i.e., the $0°$-direction, on the manifold locally around $\boldsymbol{u}$. While multiple equally valid $\omega_{\boldsymbol{u}}$ exist, the result of the surface convolution changes depending on what $0°$-direction is selected. This is because the pairings of weights $t(\boldsymbol{v})$ and surface features $[s \circ \exp_{\boldsymbol{u}} \circ \omega_{\boldsymbol{u}}](\boldsymbol{v})$, and thus their products, change when $\omega_{\boldsymbol{u}}$ is rotated. In summary, a result of the surface convolution from Definition 4.1 is *not gauge-invariant* and we understand gauge-ambiguity as the following problem:

Definition 4.2 (Gauge-ambiguity problem)**.** *The* gauge-ambiguity problem *refers to the ambiguity of selecting a gauge for a gauge-dependent surface convolution from an infinite set of equally valid gauges.*

Considering that we aim to explore whether surface CNNs, i.e., both non gauge-equivariant and gauge-equivariant, can be phrased in a common notation (RQ3), we need to additionally discuss how gauge-equivariant surface CNNs are defined in theory. Hence, in the following, we summarize the underlying theory of gauge-equivariant surface CNNs to subsequently discuss the gauge-equivariant extension of Definition 4.1 from Cohen et al. (2019).

4.1 Gauge-equivariant surface convolution

Gauge-equivariance. Equivariance describes a property of a function: if we alter the input to the function, its result alters accordingly in a predictable way (Bronstein et al., 2021). In the context of surface convolutions, it specifically states that the resulting *embedding vector rotates according to the orientation of the underlying gauge*. We can describe gauge rotations using a *symmetry group* $\mathbb{G}$, whose elements $g \in \mathbb{G}$ act on local gauges $\omega_{\boldsymbol{u}}$ from the left via $(g, \omega_{\boldsymbol{u}}) \mapsto g\omega_{\boldsymbol{u}}$. But how do n-dimensional embeddings rotate?

The rotation behavior of an n-dimensional convolution output w.r.t. a gauge rotation $g \in \mathbb{G}$ can be described using a *group representation* for $\mathbb{G}$. This group representation is a function $\rho : \mathbb{G} \to \mathrm{GL}(\mathbb{R}^n)$ which assigns an invertible linear map $\rho(g)$ to each gauge rotation $g \in \mathbb{G}$, while satisfying $\forall g, h \in \mathbb{G} : \rho(gh) = \rho(g)\rho(h)$ (Serre, 1977; Bronstein et al., 2021). In the context of surface convolutions, a group representation ρ therefore represents a mathematical tool to describe how gauge rotations g act on n-dimensional quantities, even

though the rotations g themselves are initially only defined for two-dimensional quantities in some tangent space of the surface.

Using symmetry groups and their representations, a gauge-equivariance constraint for surface convolutions can formally be defined as follows:

Definition 4.3 (Bronstein et al. (2021): Gauge-equivariance constraint)**.** *A surface convolution is said to be* gauge-equivariant *if it satisfies:*

$$\exists \rho \, \forall g \in \mathbb{G} : \rho(g) \, (s * t) \, (\boldsymbol{u}; \omega_{\boldsymbol{u}}) = (s * t) \left(\boldsymbol{u}; g^{-1} \, \omega_{\boldsymbol{u}}\right) \ .$$

This constraint formalizes the concept that we have discussed before: if we alter the input to the convolution via a gauge rotation g, its n-dimensional result alters accordingly in a predictable way. In particular, it changes according to g's representation $\rho(g)$.

Fibers and vector bundles. A standard Euclidean convolution (cf. equation 1) returns a scalar feature for each point in the Euclidean domain. A feature map is then represented by all scalars over the domain. Multiple scalar feature maps can then stacked, allowing us to associate an n-dimensional, more descriptive feature vector to each point in the domain.

Expecting a scalar-valued surface convolution to be gauge-equivariant is limiting, since the equivariance constraint would only allow to train restrictive isotropic (i.e., directionally insensitive) templates (Cohen et al., 2019).

Training anisotropic (i.e., directionally sensitive) templates thus requires us to interpret the output of a gauge-equivariant surface convolution as an n-dimensional quantity which can be associated to meaningful transformation behavior. This inevitably leads to a perspective change on embedding spaces: instead of stacking feature maps, we create a bundle of n-dimensional feature vectors over the entire surface (Cohen & Welling, 2017). Formally, each surface point is now associated to an n-dimensional vector space, called a *fiber*. The collection of all fibers together with the underlying manifold constitute a *vector bundle*.

Parallel transport. Fibers $F_{\boldsymbol{u}}$ and $F_{\boldsymbol{v}}$ at different points $\boldsymbol{u}, \boldsymbol{v} \in M$ cannot be assumed to share the same basis, considering that these spaces describe embedding vectors obtained using different gauges $\omega_{\boldsymbol{u}}$ and $\omega_{\boldsymbol{v}}$ (Cohen et al., 2019). As a result, a direct comparison or aggregation of feature vectors originating from different fibers, as required by surface convolutions, is not well-defined. Therefore, neighboring feature vectors have to be expressed in a common basis prior to calculating gauge-equivariant surface convolutions. This change of basis amounts to a *parallel transport* $g_{\boldsymbol{v} \to \boldsymbol{u}}$ into the fiber of the convolution center.

Gauge equivariant surface convolution. Gauge-equivariant surface convolutions need to satisfy the constraint presented in Definition 4.3. Cohen et al. (2019) adapt Definition 4.1 by (a) parallel transporting neighboring feature vectors into the convolution center using a representation $\rho(g_{\boldsymbol{v} \to \boldsymbol{u}})$ and (b) using a trainable template t that satisfies the rotation-steerable kernel constraint (Weiler et al., 2018):

$$\forall g \in \mathbb{G} : t(g^{-1} \boldsymbol{v}) = \rho_{\text{out}}(g^{-1}) \, t(\boldsymbol{v}) \, \rho_{\text{in}}(g) \ . \tag{2}$$

Overall, this results in a surface convolution that satisfies the constraint from Definition 4.3 and therefore is gauge-equivariant:

Definition 4.4 (Cohen et al. (2019): Gauge-equivariant surface convolution)**.** *Let M be a 2-dimensional Riemannian manifold, $B_R(\mathbf{0})$ the disk around $\mathbf{0}$ with radius R, $\omega_{\boldsymbol{u}} : B_R(\mathbf{0}) \to T_{\boldsymbol{u}}M$ a selected gauge for $T_{\boldsymbol{u}}M$, and $exp_{\boldsymbol{u}} : T_{\boldsymbol{u}}M \to M$ the exponential map at $\boldsymbol{u}$. The* gauge equivariant surface convolution *of a signal s on M with a template t in point $\boldsymbol{u} \in M$ is defined as:*

$$(s * t)(\boldsymbol{u}; \omega_{\boldsymbol{u}}) = \int_{B_R(\mathbf{0})} t(\boldsymbol{v}) \ \rho(g_{\boldsymbol{q}_v \to \boldsymbol{u}}) \ [s \circ exp_{\boldsymbol{u}} \circ \omega_{\boldsymbol{u}}](\boldsymbol{v}) \ d\boldsymbol{v} \ ,$$

where we write $\boldsymbol{q}_v = [exp_{\boldsymbol{u}} \circ \omega_{\boldsymbol{u}}](\boldsymbol{v}) \in M$ and t satisfies equation 2.

5 An Introduction to Surface Charting

The surface convolution requires a gauge $\omega_{\boldsymbol{u}} : B_R(\mathbf{0}) \to T_{\boldsymbol{u}}M$ and the exponential map $\exp_{\boldsymbol{u}} : T_{\boldsymbol{u}}M \to M$ to extract local surface features. In the context of differentiable manifolds, the composition of those functions $(\exp_{\boldsymbol{u}} \circ \omega_{\boldsymbol{u}}) : B_R(\mathbf{0}) \to M$ can be interpreted as a *local chart* (Bredon, 1993). Given that arbitrary shapes do not necessarily come with closed form solutions that yield local charts, we use surface charting algorithms to numerically estimate these $(\exp_{\boldsymbol{u}} \circ \omega_{\boldsymbol{u}})$ around each convolution center $\boldsymbol{u}$. This inevitably makes surface charting algorithms a key component of surface convolutions.

In the following, we outline the algorithmic ideas of the charting algorithms FMM, DGPC, HDM and TP, which are commonly used by surface convolution approaches (cf. Table 1). We start by motivating the *Eikonal equation*, a partial differential equation that characterizes distance functions. FMM, DGPC and HDM approximate an optimal distance function Φ that satisfies this PDE, but differ in their approach to build Φ. These differences induce varying geometric distortions, which can lead to distinct downstream learning effects in surface CNNs (RQ1). We also outline how SHOT descriptors (Salti et al., 2014) robustly estimate local reference frames on shapes, enabling the computation of tangent plane projections. However, projections can only be considered approximations to surface charts as they are computed independently of the Eikonal equation. Naturally, distortions should be expected.

5.1 Optimal distance calculation by solving the Eikonal equation

A core motivation for distance calculation in Euclidean or non-Euclidean domains can be given by taking a look at *wave-front propagation*. The dynamics of propagating wave fronts can be described with the implicit level set equation from Osher & Sethian (1988):

$$\frac{\partial \Phi(\boldsymbol{x}, t)}{\partial t} + F(\boldsymbol{x}) \; ||\nabla_{\boldsymbol{x}} \Phi(\boldsymbol{x}, t)||_2 = 0 \; . \tag{3}$$

Here, $\Phi(\boldsymbol{x}, t)$ defines a distance function that takes any point $\boldsymbol{x} \in \mathbb{R}^2$ in your domain and returns the signed minimal distance $\pm d$ to a wave front $\Gamma_t = \{\boldsymbol{y} \in \mathbb{R}^2 \mid \Phi(\boldsymbol{y}, t) = 0\}$ at a time step t. The term $\partial\Phi(\boldsymbol{x}, t)/\partial t$ thus represents a time-derivative of a distance, making $\partial\Phi(\boldsymbol{x}, t)/\partial t$ the velocity of the wave front approaching $\boldsymbol{x}$. We can rephrase equation 3 as follows:

$$\frac{\partial \Phi(\boldsymbol{x}, t)}{\partial t} = -F(\boldsymbol{x}) \; ||\nabla_{\boldsymbol{x}} \Phi(\boldsymbol{x}, t)||_2 \; . \tag{4}$$

Here we can see that the approach velocity is described by two factors: (i) An altering effect of the domain on the wave speed $F(\boldsymbol{x})$ and (ii) the magnitude of the direction vector $||\nabla_{\boldsymbol{x}} \Phi(\boldsymbol{x}, t)||_2$ that points towards the direction into which we would need to shift $\boldsymbol{x}$ to *increase* the distance from the wave front at time step t the fastest. However, the negative sign in equation 4 negates the direction of the gradient $-||\nabla_{\boldsymbol{x}} \Phi(\boldsymbol{x}, t)||_2$, making it point towards the direction that would *decrease* the distance the fastest. Thus, it points along the shortest path from $\boldsymbol{x}$ to the wave front or, in other words, it points along the *geodesic* that connects $\boldsymbol{x}$ with the wave front. Assuming a constant approach velocity of $\partial\Phi(\boldsymbol{x}, t)/\partial t = -1$, that is, the wave closes the distance to a point $\boldsymbol{x}$ by 1 unit per time step, allows us to rephrase equation 4 into the following:

$$\begin{aligned} -1 &= - \;\; F(\boldsymbol{x}) \; ||\nabla_{\boldsymbol{x}} \Phi(\boldsymbol{x}, t)||_2 \\ \Leftrightarrow \; 1 &= \;\; F(\boldsymbol{x}) \; ||\nabla_{\boldsymbol{x}} \Phi(\boldsymbol{x}, t)||_2 \; . \end{aligned} \tag{5}$$

When we assume that no wave-speed-altering effects exist anywhere in the domain, that is, for all $\boldsymbol{x} \in \mathbb{R}^2$ we have $F(\boldsymbol{x}) = 1$, we end up with the **Eikonal equation** that is typically referred to in work about distance calculations (Sethian, 1996; Kimmel & Sethian, 1998; Melvær & Reimers, 2012; Crane et al., 2017):

$$1 = ||\nabla_{\boldsymbol{x}} \Phi(\boldsymbol{x}, t)||_2 \; . \tag{6}$$

Interestingly, the Eikonal equation constrains changes in distance to be equal regardless of the origin and the approached point or the direction of movement. Hence, it states that *distance spreads isotropically through the domain*. Algorithms that aim to approximate a distance function try to solve the Eikonal equation by finding a distance function Φ that satisfies the isotropy property described by equation 6. In the following, we outline how FMM, DGPC and HDM try to solve the Eikonal equation.

Fast Marching Method (FMM). Among the charting methods discussed in Section 2, the *Fast Marching Method* (FMM) (Sethian, 1996) chronologically emerged as the first algorithm for geodesic distance computation. Motivated by Dijkstra's algorithm (Dijkstra, 1959), FMM employs a narrow band technique, categorizing domain points as *alive*, *narrow*, or *far*. Distances of *narrow* points $\boldsymbol{x}_{i,j}$ are updated from neighboring *alive* points by solving the upwind-difference structure for $\Phi(\boldsymbol{x}_{i,j})$, a discretization of the squared Eikonal equation:

$$\begin{aligned} 1 = ||\nabla\Phi(\boldsymbol{x}_{i,j})||_2^2 = \min\left(\frac{\Phi(\boldsymbol{x}_{i+1,j}) - \Phi(\boldsymbol{x}_{i,j})}{\Delta x}, 0\right)^2 + \max\left(\frac{\Phi(\boldsymbol{x}_{i,j}) - \Phi(\boldsymbol{x}_{i-1,j})}{\Delta x}, 0\right)^2 + \\ \min\left(\frac{\Phi(\boldsymbol{x}_{i,j+1}) - \Phi(\boldsymbol{x}_{i,j})}{\Delta y}, 0\right)^2 + \max\left(\frac{\Phi(\boldsymbol{x}_{i,j}) - \Phi(\boldsymbol{x}_{i,j-1})}{\Delta y}, 0\right)^2 . \end{aligned} \tag{7}$$

Points considered to be *far* become *narrow* points once they get close enough to *alive* points. *Narrow* points are accepted as *alive* points once we computed a distance for them. The algorithm iterates until all points are accepted, resulting in an approximation of the true discrete function Φ. Kimmel & Sethian (1998) explain how to use FMM on triangle meshes.

Discrete Geodesic Polar Coordinates (DGPC). Similar to FMM, the DGPC algorithm (Melvær & Reimers, 2012) numerically approximates Φ by employing a narrow band technique, as used in Dijkstra's algorithm (Dijkstra, 1959). During an update step, the chart origin and a triangle consisting of two known and one unknown point are assumed to lie in a plane spanned by the triangle. The distance to the unknown point is then updated by solving two quadratic equations for coefficients c_i and c_j of a linear combination of the triangle's edges. This linear combination returns a vector that points from the unknown point to the chart's origin, whose norm estimates the distance to the unknown point:

$$\Phi(\boldsymbol{x}) = \begin{cases} ||c_i\boldsymbol{e}_i + c_j\boldsymbol{e}_j|| & \text{if } c_i, c_j > 0, \\ \min_{l=i,j}\{\, \Phi(\boldsymbol{x}_l) \,+\, ||\boldsymbol{e}_l||\,\} & \text{else. (Dijkstra Fallback)} \end{cases} \tag{8}$$

Here, $\boldsymbol{e}_i$ and $\boldsymbol{e}_j$ represent the triangle edges. Unlike FMM, which is a one-pass *label setting* algorithm, the DGPC algorithm represents a *label correcting* algorithm, potentially updating already existing distances. Also, additionally to distances, the DGPC algorithm computes angular coordinates by linearly interpolating previously determined polar angles, forming complete geodesic polar coordinate charts.

Heat diffusion method (HDM). Crane et al. (2017) describe a method to compute distance fields along manifolds. Their method is based on the idea of Varadhan (1967), who related the heat kernel $k_{t,x}(y)$ to distance calculations $\Phi(x, y)$:

$$\Phi(x, y) = \lim_{t\to 0}\sqrt{-4t\,\log\, k_{t,x}(y)}\ . \tag{9}$$

This equation is very sensitive to errors in approximations for $k_{t,x}$, rendering it unfit for use in practice. A key observation, however, is that the spatial gradient of a heat kernel is in parallel to the gradient of a distance function, but has a wrong magnitude and sign.

To estimate geodesic distances, Crane et al. (2017) compute a shortly diffused heat scalar field and its gradient field X. Then, they negate X and, in alignment to the Eikonal equation, normalize each gradient to have a magnitude of one (cf. equation 6). Eventually, they search for a Φ whose gradient minimizes a least-squares problem:

$$\int_M |\nabla\phi - X|^2\ . \tag{10}$$

Thus, by construction, the found Φ approximately satisfies the Eikonal equation. Sharp et al. (2019) additionally explain how to retrieve angular coordinates by applying heat diffusion to vectors to obtain a complete logarithmic map for surface charting.

We have summarized our descriptions of optimal surface charting algorithms in Table 2. Another popular method to compute charts for surface convolutions is to use *tangent plane projections* (henceforth, abbreviated TP). However, the construction of tangent plane projections does not explicitly try find a Φ that

Table 2: The algorithmic ideas to estimate geodesic distance functions Φ and local angles of commonly used charting algorithms for surface convolutions. Min- and max-operations in the FMM distance update have been left away for readability.

	FMM (Sethian, 1996)	**DGPC** (Melvær & Reimers, 2012)	**HDM** (Crane et al., 2017)
algorithm	Dijkstra variant	Dijkstra variant	least squares
distance updates	solve for $\Phi(\boldsymbol{x}_{i,j})$: $\left(\frac{\Phi_{i+1,j}-\Phi_{i,j}}{\Delta x}\right)^2 + \left(\frac{\Phi_{i,j}-\Phi_{i-1,j}}{\Delta x}\right)^2 + \left(\frac{\Phi_{i,j+1}-\Phi_{i,j}}{\Delta y}\right)^2 + \left(\frac{\Phi_{i,j}-\Phi_{i,j-1}}{\Delta y}\right)^2$	solve for c_i and c_j and compute: $\Phi(\boldsymbol{x}) = \begin{cases} \|c_i\boldsymbol{e}_i + c_j\boldsymbol{e}_j\| \text{ if } c_i, c_j > 0, \\ \min_{l=i,j}\{\, \Phi(\boldsymbol{x}_l) \,+\, \|\boldsymbol{e}_l\| \,\} \end{cases}$	find a Φ that minimizes: $\int_M \lvert\nabla\Phi - X\rvert^2$
angle updates	none	angular interpolation	none

satisfies the Eikonal equation. Therefore, we consider them to be an approximate instead of an "optimal" approach.

5.2 Approximate surface charts using tangent plane projections

A common method to robustly estimate tangent planes on point cloud or mesh data is described by Salti et al. (2014). They use a least squares estimation of the normal vector which is determined by the Eigenvector associated to the smallest Eigenvalue of the distance weighted local covariance matrix. That is, let $\boldsymbol{p}$ be the point for which we want to calculate the tangent plane, R a radius around $\boldsymbol{p}$, $\boldsymbol{p}_i$ points of the discretized surface and d_i their distances to $\boldsymbol{p}$. Then, Salti et al. (2014) calculates the Eigenvectors of:

$$\boldsymbol{M}_{\boldsymbol{p}} = \frac{1}{\sum_{d_i \leq R}(R - d_i)} \sum_{d_i \leq R} (R - d_i)(\boldsymbol{p}_i - \boldsymbol{p})(\boldsymbol{p}_i - \boldsymbol{p})^T \ . \tag{11}$$

The signs of the Eigenvectors associated to the smallest and largest Eigenvalue are then disambiguated according to a set of rules given in (Salti et al., 2014). Eventually, the last axis of the reference frame is calculated as the cross product of the two disambiguated Eigenvectors. The local coordinates of the surface points around $\boldsymbol{p}$ can subsequently be calculated by the means of orthogonal projections into the tangent plane at $\boldsymbol{p}$, i.e., the x-y-plane of the computed reference frame.

5.3 A note regarding complete surface charts from FMM and HDM

Table 2 shows that among the discussed optimal approaches only the DGPC algorithm also returns angular coordinates additionally to geodesic distances. Henceforth, when we refer to surface charts that were calculated using FMM or HDM in our experiments, we imply that we use their geodesic distances in combination with angular coordinates estimated using tangent plane projections.

We note that the vector heat method (Sharp et al., 2019) builds on HDM (Crane et al., 2017) and can compute complete surface charts including geodesic distances and angular directions. However, our preliminary experiments comparing TP-completed HDM and the vector heat method typically resulted in more robust surface charts created by the former approach, leading to better downstream performance of surface CNNs in our benchmarks.

6 A Unified Implementation Framework for Surface CNNs

Varying confounding factors introduced by different implementations of surface convolutions complicate the isolation of the source for performance differences among existing surface CNNs. In the following, we explore how implementations of existing surface CNNs can be reformulated into a charting-algorithm-agnostic and unified discretization (RQ3). To this end, we aim to implement a *charting-method agnostic surface CNN framework* that builds on a *shared discretized signal-pullback* across all considered surface CNN implementations, leaving only the trainable parameters, i.e. the template, to be specified by a concrete convolution definition.

Existing surface CNNs can then be instantiated by selecting their respective weights within a self-connection & neighborhood aggregation scheme, while the underlying charting algorithm and template radius remain interchangeable. That is, charting algorithm and template radius can be treated like hyperparameters for a selected surface convolution, enabling a controlled discretization setting to systematically investigate our research questions about the impact of charting algorithms (RQ1) and varying chart radii (RQ2).

Discretization overview. We start by *discretizing* the theoretically continuous *template*. Then, we describe how surface signals are sampled from the manifold using signal interpolation with *barycentric coordinates* in precomputed local charts. This constitutes the discrete realization of the *signal-pullback* from the theoretical surface convolution. Based on the signal-pullback, we define the *patch operator* which has been adapted by early surface CNNs (Masci et al., 2015; Boscaini et al., 2016; Monti et al., 2017) from the construction of local shape descriptors (Kokkinos et al., 2012) to read surface signals robustly. The patch operator enables us to implement the discrete, *non gauge-equivariant* surface convolution (cf. Definition 4.1) which we can use in together with *angular max-pooling* layers to design non-gauge-equivariant surface CNNs.

Then, we explain how we discretize the parallel transport. This enables the implementation of gauge-equivariant surface convolutions (cf. Definition 4.4). We exemplarily discuss how we can implement HSNs (Wiersma et al., 2020), GEM-CNNs (Haan et al., 2021) and EMANs (Basu et al., 2022) using our discretized signal-pullback with preceding parallel transport.

Given that any implementation relies on the same discretized signal-pullback at this point (with potentially preceding parallel transport), non gauge equivariant and gauge equivariant surface convolutions merely vary in their sets of trainable parameters that construct the template. A comparison of HSNs and GEM-CNNs within this unified discretization suggests a natural extension of GEM-CNNs by learnable phase offsets and radially dependent weights inspired by HSNs. We refer to this extension as GEM-CNN+. This can similarly be applied to EMANs, creating EMAN+.

Overall, our surface CNN framework can be summarized in one equation and a table of parameter choices (cf. equation 24 and Table 4). This unified discretization (RQ3) ensures that all considered surface convolutions follow the same implementation idea and thereby supports a systematic analysis of differences between existing surface CNNs (RQ1, RQ2).

6.1 Preprocessing

Template vertices. Our discretization of the surface convolution builds on the idea of Poulenard & Ovsjanikov (2018), who compute surface convolutions using *barycentric coordinates* to interpolate signal values at fixed positions within local surface charts. To this end, we start by defining the *template vertices*:

Definition 6.1 (Template Vertices)**.** *We discretize a template into $N_\tau \times N_\theta$* template vertices *by defining fixed interpolation coordinates in the local charts on the surface:*

$$\mathbb{T} = \{(\tau_r,\, \theta_a) \,|\, r = 1, ..., N_\tau,\ a = 1, ..., N_\theta\} \text{ with } \tau_r = \frac{r\tau_R}{N_\tau} \text{ and } \theta_a = \frac{(a-1)2\pi}{N_\theta}\ ,$$

where τ_R defines the template radius*. Each template vertex $(\tau_r, \theta_a) \in \mathbb{T}$ defines a spatially fixed position at which trainable weights can be placed. We define the origin of any discrete template at $(\tau_0, \theta_0) := (0, 0)$. It is* not *included in $\mathbb{T}$.*

The discrete template is thus given by the cartesian product of geodesic polar coordinates. Its origin corresponds to the convolution center and will be handled separately in the discretized surface convolution that we will structure into a self-connection vs. neighborhood aggregation scheme. An illustration of a discrete template in a local chart is provided in Figure 3 on the left. The middle and right illustrations in Figure 3 show how the chart in which the template was placed looks on the 3D shape.

Barycentric coordinates. Barycentric coordinates provide a convenient way to parameterize a plane spanned by a triangle or to check whether a point is located within a triangle (Ericson, 2004). Specifically, given triangle corner points $\boldsymbol{t}_1$, $\boldsymbol{t}_2$, $\boldsymbol{t}_3$ and a query point $\boldsymbol{q}$ in the same plane, one can express $\boldsymbol{q}$ as a linear

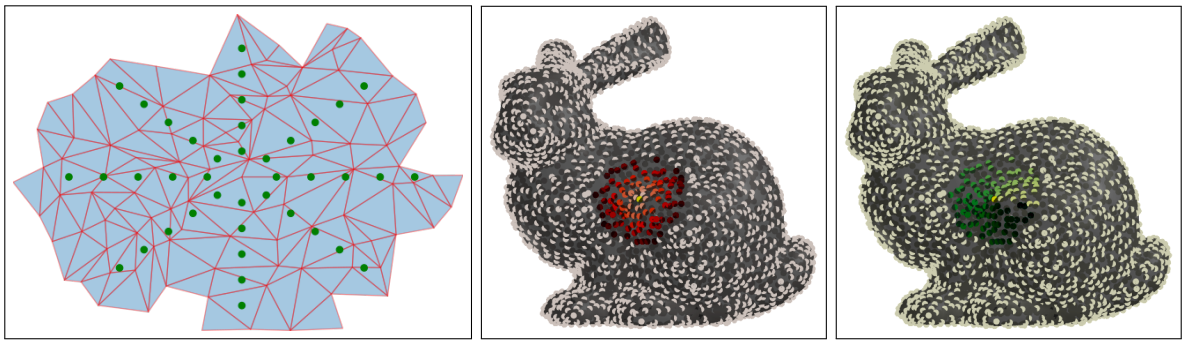


Figure 3: **[Left]** The template vertices $\mathbb{T}$ of a discretized template of size $N_\tau = 5$ and $N_\theta = 8$ (green dots) put on top of a surface chart (blue triangles with red edges). **[Middle and Right]** The radial and angular coordinates of the same chart illustrated on the Stanford Bunny, the shape on which the chart was computed. Red colors represent geodesic distances (radial coordinates). Green colors show directions (angular coordinates). The darker the color, the larger its value. The yellow dot represents the chart's origin.

combination of the triangle points:

$$\boldsymbol{q} = \sum_{i=1}^{3} b_i \boldsymbol{t}_i \text{ with } \sum_{i=1}^{3} b_i = 1 \ . \tag{12}$$

The coefficients b_i represent barycentric coordinates for $\boldsymbol{q}$. We know that $\boldsymbol{q}$ is within the triangle if for all b_i holds that $0 \leq b_i \leq 1$. As explained by Ericson (2004), given a triangle, barycentric coordinates b_i can be found efficiently for $\boldsymbol{q}$ by solving a 2×2 system of linear equations.

Preprocessing pipeline. Overall, our surface convolution requires a preprocessing stage that consists of three steps: (i) Compute local charts around every point of the input shape, (ii) define the template vertices following Definition 6.1 and (iii) determine barycentric coordinates for the specified template vertices within all local charts. In terms of our barycentric coordinates description, query points are $\boldsymbol{q} \in \mathbb{T}$, and the triangles are taken from the local neighborhood around the convolution center, whereby their corner points $\boldsymbol{t}_1, \boldsymbol{t}_2$ and $\boldsymbol{t}_3$ are represented in geodesic coordinates of the computed local chart.

Together with a predefined surface signal, the barycentric coordinates are passed to the surface convolution as input, which uses them to interpolate signal values at its template vertices on the surface to eventually compute the convolution. Details of the discretized surface convolution are described in the next section.

6.2 Implementing non gauge-equivariant surface convolutions

Signal-pullback. While surface convolutions are theoretically defined on continuous manifolds, in practice, we can use meshes consisting of finite sets of vertices and triangles which piecewise linearly approximate the real surface. Consequently, the template vertices typically fall into triangles in between mesh vertices when we place the discrete template onto the triangle mesh at a specific position. This is problematic since feature vectors are only defined for the mesh's vertices. Therefore, we use linear signal interpolation at the template vertices based on existing feature vectors provided by surrounding shape vertices. To this end, our *discretized signal-pullback* uses the precomputed barycentric coordinates:

Definition 6.2 (Poulenard & Ovsjanikov (2018): Signal-pullback)**.** *Let $\boldsymbol{S}^{(r,a)} \in \mathbb{R}^{d_{in} \times 3}$ be a matrix whose columns contain the d_{in}-dimensional feature vectors of the three mesh vertices enclosing the template vertex (τ_r, θ_a). Furthermore, let $\boldsymbol{b}^{(r,a)} \in \mathbb{R}^3$ describe their associated barycentric coordinates for interpolating at template vertex (τ_r, θ_a). We approximate the* signal-pullback *using the linear map:*

$$\boldsymbol{x}^{(r,a)} = \boldsymbol{S}^{(r,a)} \boldsymbol{b}^{(r,a)} \ .$$

In the following, we use the interpolations at all template vertices $(\tau_r, \theta_a) \in \mathbb{T}$. Therefore, we stack the interpolations $\boldsymbol{x}^{(r,a)}$ in a matrix $\boldsymbol{X} \in \mathbb{R}^{d_{\text{in}} \times N_\tau N_\theta}$ with $\boldsymbol{X} = [\boldsymbol{x}^{(1,1)}, \cdots, \boldsymbol{x}^{(N_\tau, N_\theta)}]$.

Patch-operator. One of the earliest surface convolutions (Masci et al., 2015) adapted the idea of Kokkinos et al. (2012) to use a weighting function for reading surface information. Kokkinos et al. (2012) aimed to create robust local shape descriptors by spatially arranging local bins, each equipped with a soft-membership function, i.e. a weighting function, to determine how much neighboring shape vertices attribute to a bin. Monti et al. (2017) leave the choice of the weighting function for soft-binning open, to create a parametric patch operator which allows to instantiate prior convolutions as particular instances of their framework:

Definition 6.3 (Monti et al. (2017): Patch operator)**.** *Let $\boldsymbol{X} \in \mathbb{R}^{d_{in} \times N_\tau N_\theta}$ be the matrix that contains the interpolated feature vectors at all template vertices. Furthermore, let $\boldsymbol{w}^{(r,a)} \in \mathbb{R}^{N_\tau N_\theta}$ be the pre-defined weighting function localized in template vertex (τ_r, θ_a) which assigns scalar weights to all template vertices. The* patch operator *is defined as:*

$$\boldsymbol{p}^{(r,a)} = \boldsymbol{X}\boldsymbol{w}^{(r,a)} \ .$$

In contrast to existing literature (Kokkinos et al., 2012; Masci et al., 2015; Boscaini et al., 2016; Monti et al., 2017), which pre-computes the attribution of all mesh vertices included in the local chart to the bins, we only consider the attribution of interpolated feature vectors to the bins. At the cost of precision, this allows flexibility: since we remember the barycentric coordinates for interpolation, the patch-operator results $\boldsymbol{p}^{(r,a)}$ do not need to be preprocessed for each shape individually.

Surface convolution. We can now formulate the discrete surface convolution using a neighborhood aggregation term that uses the patch-operator and a self-connection term for the convolution center:

Definition 6.4 (Discrete surface convolution)**.** *Let $\boldsymbol{t}_m \in \mathbb{R}^{d_{in}}$ describe a trainable weight vector for the self-connection, $\boldsymbol{t}_m^{(r,a)} \in \mathbb{R}^{d_{in}}$ the trainable weight vector at template vertex (τ_r, θ_a) and $\beta \in \mathbb{R}$ a trainable bias. Furthermore, let $\boldsymbol{s} \in \mathbb{R}^{d_{in}}$ be the feature vector at the convolution center and $\boldsymbol{p}^{(r,a)} \in \mathbb{R}^{d_{in}}$ the feature vector at template vertex (τ_r, θ_a) computed using the patch-operator. We define the* surface convolution *as:*

$$\boldsymbol{x}_m = \boldsymbol{t}_m^T \boldsymbol{s} + \sum_{r=1}^{N_\tau} \sum_{a=1}^{N_\theta} \boldsymbol{t}_m^{(r,a)^T} \boldsymbol{p}^{(r,a)} + \beta, \ \textit{for } m = 1, \ldots, d_{out} \ .$$

Figure 4 illustrates the surface convolution algorithm, ranging from the definition of template vertices until final weighted aggregation. The resulting surface convolution conceptually lies between classical image and graph convolutions. In analogy to graph convolutions, the implementation follows the self-connection plus neighborhood aggregation scheme. In contrast to graph convolutions, the neighborhood aggregation is not calculated using isotropic message passing along graph edges. Instead, aggregation pairs directionally dependent (i.e., anisotropic) weights with interpolated feature vectors at template vertices that have fixed, relative locations. This aligns more with the concept of fixed-sized templates used in image CNNs, and yields a discretization that naturally leverages the underlying continuous structure of surfaces.

Dirac and Geodesic CNNs. In the course of this work, we include two non gauge-equivariant surface convolutions baselines into our ablations that differ in their predefined weighting function $\boldsymbol{w}^{(r,a)}$ used in the patch operator. First, we consider the *Dirac* weighting function

$$\boldsymbol{w}^{(r,a)}(\tau, \theta) = \begin{cases} 1 & \text{if } (\tau, \theta) = (\tau_r, \theta_a) \\ 0 & \text{else.} \end{cases} \tag{13}$$

Inserting it into Definition 6.3 yields $\boldsymbol{x}^{(r,a)} = \boldsymbol{X}\boldsymbol{w}^{(r,a)}$. That is, the weighting function merely selects the interpolated feature vector at template vertex (τ_r, θ_a) and therefore simplifies the patch operator back to the discrete signal-pullback. We refer to convolutions that use Dirac weighting as *intrinsic surface convolutions* (ISC). Alternatively, we consider the *Gaussian* weighting function:

$$\boldsymbol{w}^{(r,a)}(\tau, \theta) = \frac{1}{\sqrt{(2\pi)^2 \sigma_\tau^2 \sigma_\theta^2}} \exp\left(-\frac{1}{2} \begin{pmatrix} \tau - \tau_r \\ d(\theta, \theta_a) \end{pmatrix}^T \begin{pmatrix} 1/\sigma_\tau^2 & 0 \\ 0 & 1/\sigma_\theta^2 \end{pmatrix} \begin{pmatrix} \tau - \tau_r \\ d(\theta, \theta_a) \end{pmatrix} \right) \tag{14}$$

with $d(\theta, \theta_a)$ representing the shortest angular distance

$$d(\theta, \theta_a) = \min(|\theta - \theta_a|, 2\pi - |\theta - \theta_a|) \ . \tag{15}$$

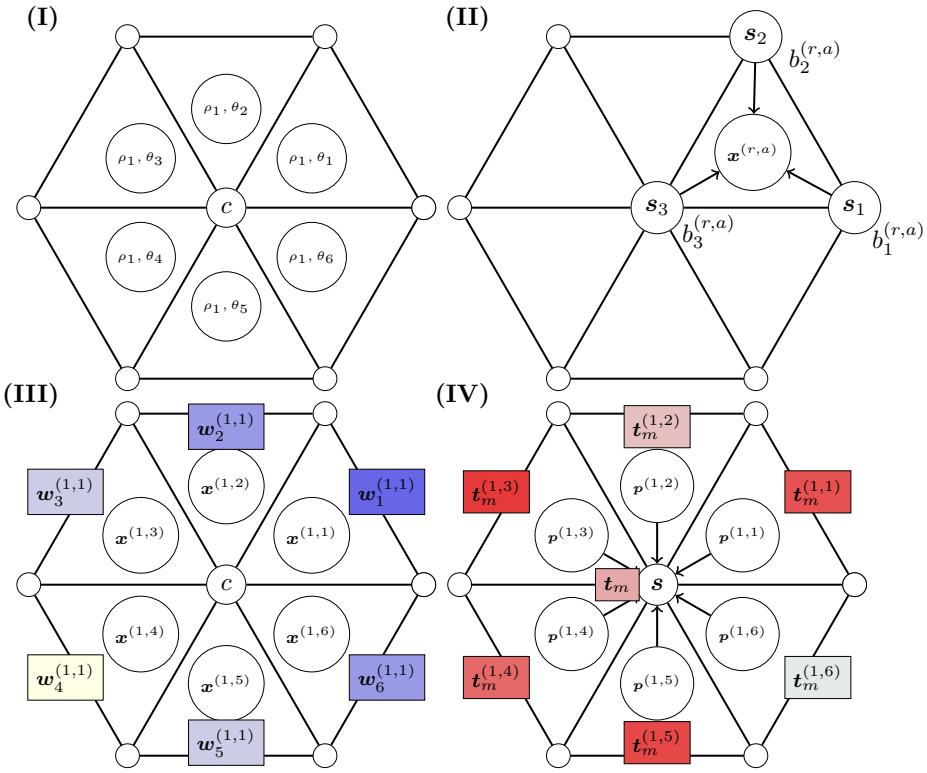


Figure 4: The discretized surface convolution visualized. **(I)** Position the template vertices $\mathbb{T}$ around the convolution center in mesh vertex c. Here, it was configured to have $N_\tau = 1$ and $N_\theta = 6$. **(II)** Interpolate surrounding mesh vertex features at the template vertices (τ_r, θ_a) by calculating $\boldsymbol{x}^{(r,a)} = \sum_i b_i^{(r,a)} \boldsymbol{s}_i$ with barycentric coordinates $b_i^{(r,a)}$ and the $\boldsymbol{s}_i$ being column vectors in $\boldsymbol{S}^{(r,a)}$. **(III)** To calculate $\boldsymbol{p}^{(r,a)}$, center the predefined weighting function $\boldsymbol{w}$ on template vertex (τ_r, θ_a) (i.e., select $\boldsymbol{w}^{(r,a)}$) and compute the weighted sum over all interpolations: $\boldsymbol{p}^{(r,a)} = \boldsymbol{X}\boldsymbol{w}^{(r,a)}$. Do so for each template vertex (τ_r, θ_a). **(IV)** Compute the inner products $\sum_i \boldsymbol{t}_{m,i}^{(r,a)} \boldsymbol{p}_i^{(r,a)}$, sum over the results at all template vertices (τ_r, θ_a), and add the self-connection $\boldsymbol{t}_m^T \boldsymbol{s}$ and bias β to calculate the m-th embedding feature $\boldsymbol{x}_m$ for the new embedding vector at the center vertex c.

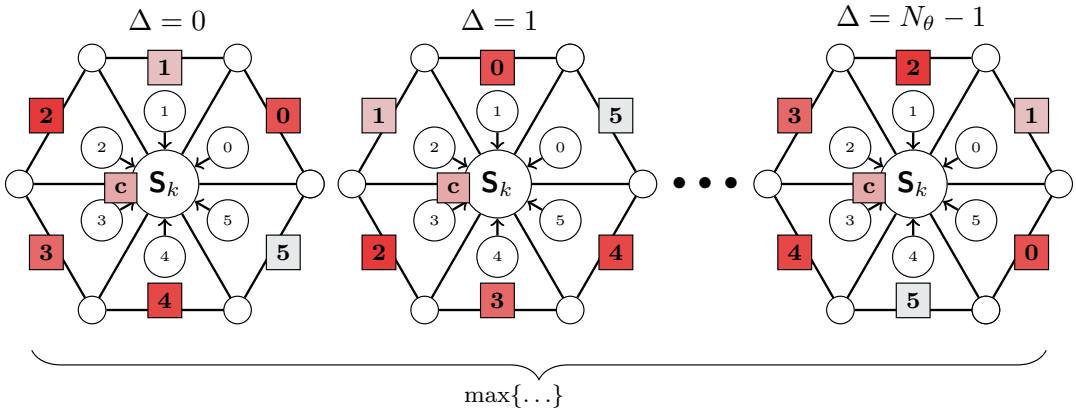


Figure 5: Angular max-pooling visualized. Incremental rotations from left to right, starting with picture **(IV)** from Figure 4 with no rotation ($\Delta = 0$) and ending in the maximal rotation ($\Delta = N_\theta - 1$). Given the convolution results for all Δ, AMP selects the maximum.

Surface convolutions using Gaussian weighting resemble *geodesic surface convolutions* (GCNNs) (Masci et al., 2015). Note that while Masci et al. (2015) did use Gaussian weighting, they determined angular weights using a point-to-set distance which requires a modification of our preprocessing stage that is not shared across all surface CNNs. Shortest angular distances can be computed during the forward pass "*on-the-fly*", making it the more suitable choice for a framework that enforces a common preprocessing pipeline across all surface CNNs.

6.3 Implementing gauge-equivariant surface convolutions

As discussed in Section 5, fundamental for computing surface convolutions is the preceding computation of local charts. However, there are multiple equally valid charts for the neighborhood around one vertex since there is no unique 0°-direction for the chart's orientation. This renders the directional localization of learned features ambiguous and represents the practical phenomenon that emerges from the theoretical *gauge-ambiguity problem*. Early work resolves this issue by choosing the 0°-direction at random, compute the convolution using multiple rotations of the template and eventually choose the maximum response over all results (Masci et al., 2015; Monti et al., 2017; Sun et al., 2020):

Definition 6.5 (Masci et al. (2015): Angular max-pooling (AMP))**.** *Angular max-pooling returns the maximum response of the convolution results over all template rotations:*

$$\boldsymbol{x}_m = \max\{\boldsymbol{x}_m^\Delta \mid \Delta \in \{0, \dots, N_\theta - 1\}\} \quad \textit{where} \quad \boldsymbol{x}_m^\Delta = \boldsymbol{t}_m^T \boldsymbol{s} + \sum_{r=1}^{N_\tau} \sum_{a=1}^{N_\theta} \boldsymbol{t}_m^{(r,a+\Delta \bmod N_\theta)^T} \boldsymbol{p}^{(r,a)} + \beta \ .$$

Figure 5 illustrates angular max-pooling. However, this approach comes with the drawback that the convolution has to be computed multiple times per layer in a forward pass. More recently, a general strategy to resolve gauge-ambiguity has been to select an arbitrary 0°-direction or gauge, respectively, and ensure that the output of the convolution rotates according to the orientation of the local gauge. This encapsulates the idea of *gauge-equivariance*, as discussed in Section 4.1.

In order to implement gauge-equivariant surface convolutions, we first need to integrate parallel transport into the signal-pullback of the non gauge-equivariant surface convolution.

Parallel transport. In practice, parallel transport on a 2D-surface in a 3D ambient space amounts to rotating a vector by an angle ψ. Existing gauge-equivariant surface convolutions typically pre-compute all required ψ. For instance, Haan et al. (2021) describe how to align local reference frames, which induces a ψ, and Wiersma et al. (2020) as well as Mitchel et al. (2021) use the Vector Heat Method (Sharp et al., 2019) to compute ψ.

Theoretically, the rotation behavior of the feature vectors (i.e., their *type*) determines the representation $\rho(\psi) \in \mathrm{GL}(\mathbb{R}^{d_{\text{out}}})$ (Cohen & Welling, 2017; Haan et al., 2021). However, in practice we *first select* the representations $\rho(\psi)$ like hyperparameters and *thereby induce* the type of the feature vectors indirectly.

In our framework, aggregation of neighboring feature vectors first occurs during feature vector interpolation at the template vertices (cf. Figure 4 **II**). Parallel transport must precede this step, as otherwise the gathered feature vectors do not share a common basis and therefore cannot be interpolated. We adopt the basis of the fiber at the convolution center as the common basis for all required, neighboring feature vectors. That is, our framework rotates neighboring feature vectors by ψ to transport them into the center fiber before interpolation:

Definition 6.6 (Signal-pullback with parallel transport)**.** *Let* $\boldsymbol{S}^{(r,a)} = [\boldsymbol{s}_1, \boldsymbol{s}_2, \boldsymbol{s}_3] \in \mathbb{R}^{d_{in} \times 3}$ *and* $\boldsymbol{b}^{(r,a)} \in \mathbb{R}^3$ *be defined as in the regular signal-pullback (cf. Definition 6.2). Furthermore, let representation* ρ *encode the type of feature vectors with* $\rho(\psi) \in \mathbb{R}^{d_{in} \times d_{in}}$ *and*

$$\widetilde{\boldsymbol{S}} = [\rho(\psi_1)\boldsymbol{s}_1,\ \rho(\psi_2)\boldsymbol{s}_2,\ \rho(\psi_3)\boldsymbol{s}_3] \ .$$

We approximate the signal-pullback with parallel transport *using the linear map:*

$$\widetilde{\boldsymbol{x}}^{(r,a)} = \widetilde{\boldsymbol{S}} \boldsymbol{b}^{(r,a)} \ .$$

Using this discretized signal-pullback with parallel transport, we can now implement gauge-equivariant surface convolutions in the same self-connection and neighborhood aggregation scheme as their non-gauge-equivariant counterparts. This way, all considered surface convolutions rely on the same discretized signal-pullback except that gauge-equivariant surface CNNs additionally implement a parallel transport.

6.4 Implementing gauge-equivariant surface convolutions

In the following, we explain the convolutions implemented by HSNs (Wiersma et al., 2020), GEM-CNNs (Haan et al., 2021) and EMANs (Basu et al., 2022) in terms of the discretized signal-pullback with preceding parallel transport. We adopt the same template-vertex-based self-connection & neighborhood aggregation scheme as in Definition 6.4, thereby enforcing an analogous discretization of gauge-equivariant architectures to their non gauge-equivariant convolutions counterparts.

In addition to differing in the discretized signal-pullback, existing surface convolutions discretizations also vary in their level of granularity: some definitions return entire embedding vectors while others return individual values in the embedding vectors. Analogous to equation 6.4, the unified notation we aim for defines the *element-wise output* of any considered gauge-equivariant surface convolution. These values can subsequently be stacked into full output embeddings.

Harmonic surface networks. Wiersma et al. (2020) extend Harmonic Networks (Worrall et al., 2017) to *Harmonic Surface Networks* (HSNs), which rely on complex-valued inner products. Originally, their surface convolution is defined over the neighborhood $\mathcal{N}_{\boldsymbol{u}}$ around vertex $\boldsymbol{u}$, where neighbors can have *arbitrary* polar coordinates. We rewrite the harmonic surface convolution into a complex inner product using matrix-vector notation. This formulation describes how the m-th complex output feature is computed in dependence of neighbors with *fixed relative locations* given by the template vertices $\mathbb{T}$:

$$\boldsymbol{x}_m^{\text{neigh}} = \sum_{r=1}^{N_\tau} \sum_{a=1}^{N_\theta} \sum_{n=1}^{d_{\text{in}}} w_{mn}^{(r)} \begin{pmatrix} \cos(\gamma\theta^{(a)} + \beta_m) & -\sin(\gamma\theta^{(a)} + \beta_m) \\ \sin(\gamma\theta^{(a)} + \beta_m) & \cos(\gamma\theta^{(a)} + \beta_m) \end{pmatrix} \begin{pmatrix} \text{Re}\left(\widetilde{\boldsymbol{x}}_n^{(r,a)}\right) \\ \text{Im}\left(\widetilde{\boldsymbol{x}}_n^{(r,a)}\right) \end{pmatrix} . \tag{16}$$

That is, HSNs learn how to rotate and adjust the norm (i.e., the modulus) of complex input values. We refer to the *trainable* rotation matrix as the *phase weight matrix* $\boldsymbol{\Phi}_{am}^{\text{neigh}}$. The network learns different phase offsets β_m per output dimension m. Details on the derivation of this equation from the original formulation for HSNs can be found in Appendix A.1.

Here, n is the index that refers to the n-th element of a d_{in}-dimensional *complex* input feature vector $\widetilde{\boldsymbol{x}}$. That is, the total feature vector stores $2d_{\text{in}}$ scalars, containing real and imaginary parts. Prior to aggregation, the input feature vector $\widetilde{\boldsymbol{x}}$ has been parallel transported into the convolution center using Definition 6.6, which we denote with the tilde. Each output feature also represents a complex number. Thus, we have an output dimensionality of $2d_{\text{out}}$.

So far, equation 16 only describes how neighbor features are aggregated. The self-connection is located in the origin $(0, 0)$ of the local chart. Therefore, we are not required to transport the feature vector and can directly work with $\boldsymbol{x}^{(0,0)}$. Additionally, since we define $\theta = 0$ for the origin, we have a simplified phase weight matrix $\boldsymbol{\Phi}_m^{\text{self}}$ for self-connections. Lastly, we train exactly one radial weight for $\tau = 0$ and therefore get the self-connection:

$$\boldsymbol{x}_m^{\text{self}} = \sum_{n=1}^{d_{\text{in}}} w_{mn} \begin{pmatrix} \cos\beta_m & -\sin\beta_m \\ \sin\beta_m & \cos\beta_m \end{pmatrix} \begin{pmatrix} \text{Re}\left(\boldsymbol{x}_n^{(0,0)}\right) \\ \text{Im}\left(\boldsymbol{x}_n^{(0,0)}\right) \end{pmatrix} . \tag{17}$$

The final m-th element of the new embedding vector associated to vertex $\boldsymbol{u}$ is given by $\boldsymbol{x}_m = \boldsymbol{x}_m^{\text{self}} + \boldsymbol{x}_m^{\text{neigh}}$.

Gauge-equivariant mesh CNNs. Haan et al. (2021) replace the isotropic kernel used in standard Graph Neural Networks (Kipf & Welling, 2017) with anisotropic, gauge-equivariant kernels, and thereby introduce *Gauge-Equivariant Mesh CNNs* (GEM-CNNs). GEM-CNNs have originally been defined on the one-hop neighborhood $\mathcal{N}_k$ around a vertex $\boldsymbol{u}$, allowing neighbors to have arbitrary polar coordinates. We substitute

$\mathcal{N}_k$ with template vertices $\mathbb{T}$, and reformulate the original version to explicitly describe how the m-th output component is computed in dependence of fixed relative locations described by $\mathbb{T}$:

$$\boldsymbol{x}_m^{\text{neigh}} = \sum_{r=1}^{N_\tau} \sum_{a=1}^{N_\theta} \sum_{n=1}^{d_{\text{in}}} \underbrace{\boldsymbol{K}_{mn}^{\text{neigh}}(\theta_a)}_{=\sum_s w_{mn}^s \boldsymbol{K}_{mn}^s(\theta_a)} \begin{pmatrix} \text{Re}\left(\widetilde{\boldsymbol{x}}_n^{(r,a)}\right) \\ \text{Im}\left(\widetilde{\boldsymbol{x}}_n^{(r,a)}\right) \end{pmatrix} . \tag{18}$$

Here, each $\boldsymbol{K}_{mn}^{\text{neigh}} \in \mathbb{R}^{2\times 2}$ describes a block in the complete weight matrix $\boldsymbol{K}^{\text{neigh}} \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$, which maps the n-th input feature of type ρ_{in} to type ρ_{out}. $\boldsymbol{K}^{\text{neigh}}$ needs to satisfy the rotation-steerable kernel constraint (cf. equation 2) to make the convolution gauge-equivariant. To this end, a block is constructed as a weighted linear combination $\boldsymbol{K}_{mn}^{\text{neigh}} = \sum_s w_{mn}^s \boldsymbol{K}_{mn}^s$ of basis kernels $\boldsymbol{K}_{mn}^s$. Thereby, the weights w_{mn}^s are learned and each $\boldsymbol{K}_{mn}^s$ is explicitly constructed to satisfy the kernel constraint, which ultimately leads to $\boldsymbol{K}^{\text{neigh}}$ satisfying it too. Achieving gauge-equivariance via this block-matrix construction is possible because the type of a complete feature vector is considered to be given by how its individual features rotate under gauge rotations. Consequently, gauge-equivariance of the complete convolution arises from gauge-equivariance of its feature-to-feature mappings. This allows to decompose the kernel constraint into constraints for individual block-matrices (Weiler & Cesa, 2019).

Recall that the type of a feature vector is described by its associated representation. Haan et al. (2021) construct representations $\rho(g) \in \text{GL}(\mathbb{R}^d)$ by building on the idea of Weiler & Cesa (2019): $\rho(g)$ can be constructed for arbitrary feature dimensions d as the direct sum of *irreducible representations* of SO(2), resulting in $\mathbb{R}^{d\times d}$ block-diagonal rotation matrices. Due to the block-diagonal structure, each feature is transformed by exactly one block in $\rho(g)$ (i.e., by one irreducible representation). This associates single features to individual rotation behavior. The rotation behavior, i.e., the type, of a complete feature vector is then described by the collective behavior of its features under gauge-rotations.

In practice, we map the n-th input feature of type ρ_{in} to its contribution to the m-th output feature of type ρ_{out} via $\boldsymbol{K}_{mn}^{\text{neigh}}$. Here, GEM-CNNs distinguish 4 map-types between rotation-invariant and rotation-equivariant features: (i) invariant-to-invariant ($\rho_0 \to \rho_0$), (ii) invariant-to-equivariant ($\rho_0 \to \rho_{\gamma'}$), (iii) equivariant-to-invariant ($\rho_\gamma \to \rho_0$), and (iv) equivariant-to-equivariant ($\rho_\gamma \to \rho_{\gamma'}$). Thereby, rotation-invariant features can be interpreted as real numbers and rotation-equivariant features as complex numbers (cf. Appendix A.2). Consequently, map type (i) is a real-valued mapping, map type (ii) lifts real numbers into the complex plane, map type (iii) projects complex numbers onto the real axis and map type (iv) implements a complex-valued function. In the course of this work, when working with GEM-CNNs or its extensions, we always assume to work with complex numbers and set the imaginary part to zero in the presence of rotation-invariant features to reduce notational clutter for case distinctions.

Since GEM-CNNs thus work with complex-valued features, similarly to HSNs, a single in- and output feature consists of two scalars, yielding $2d_{\text{in}}$-dimensional input and $2d_{\text{out}}$-dimensional output vectors.

Using a self-kernel $\boldsymbol{K}^{\text{self}}$ and the feature vector at the convolution center $\boldsymbol{x}^{(0,0)}$ (not transported) allows us to calculate the self-connection analogously:

$$\boldsymbol{x}_m^{\text{self}} = \sum_{n=1}^{d_{\text{in}}} \boldsymbol{K}_{mn}^{\text{self}} \begin{pmatrix} \text{Re}\left(\boldsymbol{x}_n^{(0,0)}\right) \\ \text{Im}\left(\boldsymbol{x}_n^{(0,0)}\right) \end{pmatrix} . \tag{19}$$

Eventually, the layer returns a feature vector with each feature defined as $\boldsymbol{x}_m = \boldsymbol{x}_m^{\text{self}} + \boldsymbol{x}_m^{\text{neigh}}$.

Equivariant mesh attention networks. Equivariant Mesh Attention Networks (EMAN) (Basu et al., 2022) integrate attention coefficients into GEM-CNNs. As an extension of GEM-CNNs, the original version of EMANs also considered solely one-hop neighborhoods. Replacing the neighborhood with $\mathbb{T}$ yields:

$$\boldsymbol{x}_m^{\text{neigh}} = \sum_{r=1}^{N_\tau} \sum_{a=1}^{N_\theta} \sum_{n=1}^{d_{\text{in}}} \alpha_{ra}^{\text{neigh}} \, \boldsymbol{K}_{mn}^{\text{neigh,value}}(\theta_a) \begin{pmatrix} \text{Re}\left(\widetilde{\boldsymbol{x}}_n^{(r,a)}\right) \\ \text{Im}\left(\widetilde{\boldsymbol{x}}_n^{(r,a)}\right) \end{pmatrix} . \tag{20}$$

The attention coefficients $\alpha_{ra}^{\text{neigh}} \in \mathbb{R}$ are calculated using common QKV-attention (Vaswani et al., 2017). Thereby, separate gauge-equivariant kernels $\boldsymbol{K}$ are used to calculate *query* ($\boldsymbol{K}^{\text{neigh,query}}$), *key* ($\boldsymbol{K}^{\text{neigh,key}}$) and *value* ($\boldsymbol{K}^{\text{neigh,value}}$) embeddings. The use of gauge-equivariant kernels to calculate attention coefficients makes attention coefficients gauge-invariant. This invariance preserves gauge-equivariance of EMANs.

In analogy to the neighbor-aggregation, we calculate the embeddings of self-connections:

$$\boldsymbol{x}_m^{\text{self}} = \sum_{n=1}^{d_{\text{in}}} \alpha^{\text{self}} \, \boldsymbol{K}_{mn}^{\text{self,value}} \begin{pmatrix} \text{Re}\left(\boldsymbol{x}_n^{(0,0)}\right) \\ \text{Im}\left(\boldsymbol{x}_n^{(0,0)}\right) \end{pmatrix} , \tag{21}$$

using the non-transported feature vector at the center vertex $\boldsymbol{x}^{(0,0)}$. Similarly to GEM-CNNs, the EMAN-layer returns a $2d_{\text{out}}$-dimensional feature vector with d_{out} complex entries $\boldsymbol{x}_m = \boldsymbol{x}_m^{\text{self}} + \boldsymbol{x}_m^{\text{neigh}}$.

6.5 Making GEM-CNNs and EMANs learn radial sensitivity and constant phase-offsets

Comparing HSNs and GEM-CNNs in our unified notation. Comparing equation 16 and equation 18 shows that HSNs learn different radial weights $\omega_{mn}^{(r)}$ per radial coordinate r while GEM-CNNs learn one set of radial weights ω_{mn}^{s} for all radial coordinates r. This difference appears due to the original implementation of GEM-CNNs, which solely considers features from one-hop neighbors, limiting the need for learning different radial weights. However, learning different weights for features positioned farther away from the convolution center is reasonable because geometric distortions of local charts tend to increase with the distance from the convolution center. Thus, features that lie farther out may require different treatments compared to those lying close.

Furthermore, the phase weight matrix of HSNs learns a constant phase offset β_m, which is not learned by GEM-CNNs. This restricts the network from learning an adjustment to the orientation of a feature, which portrays another degree of freedom that may be beneficial to a downstream task.

Proposing an adjustment to GEM-CNNs. We thus propose to modify the kernels of Haan et al. (2021) by introducing a trainable phase weight β_m to each $\widetilde{\boldsymbol{K}}_{mn}^{s}$ and substitute radial dependent weights $w_{mn}^{(s,r)}$ for the coefficients $w_{mn}^{(s)}$:

$$\boldsymbol{x}_m^{\text{neigh}} = \sum_{r=1}^{N_\tau} \sum_{a=1}^{N_\theta} \sum_{n=1}^{d_{\text{in}}} \underbrace{\widetilde{\boldsymbol{K}}_{mnr}^{\text{neigh}}(\theta_a)}_{=\sum_s w_{mn}^{(s,r)} \widetilde{\boldsymbol{K}}_{mn}^{s}(\theta_a)} \begin{pmatrix} \text{Re}\left(\widetilde{\boldsymbol{x}}_n^{(r,a)}\right) \\ \text{Im}\left(\widetilde{\boldsymbol{x}}_n^{(r,a)}\right) \end{pmatrix} . \tag{22}$$

As constant phase-weights merely represent learned orientation-offsets to features, they do not break the equivariance-property of the $\widetilde{\boldsymbol{K}}_{mn}^{s}$. We explicitly show this in Appendix A.3.

We adjust the basis-kernels for the self-connection term analogously. All adjusted basis-kernels are presented in Table 3. To distinguish from regular GEM-CNNs, we call this convolution type *GEM-CNN+*.

EMANs can be extended analogously: the kernels from Table 3 substitute their original kernels in the neighborhood aggregation and the trainable coefficients are made dependent on the radial coordinate:

$$\boldsymbol{x}_m^{\text{neigh}} = \sum_{r=1}^{N_\tau} \sum_{a=1}^{N_\theta} \sum_{n=1}^{d_{\text{in}}} \alpha_{ra}^{\text{neigh}} \, \widetilde{\boldsymbol{K}}_{mnr}^{\text{neigh,value}}(\theta_a) \begin{pmatrix} \text{Re}\left(\widetilde{\boldsymbol{x}}_n^{(r,a)}\right) \\ \text{Im}\left(\widetilde{\boldsymbol{x}}_n^{(r,a)}\right) \end{pmatrix} . \tag{23}$$

The basis-kernels for the self-connection term are adjusted analogously. We refer to EMANs using those kernels as *EMAN+*.

6.6 A summarizing surface CNN framework

Answering RQ3. In Section 3, we raised the following research question: *How can existing surface convolutions be reformulated into a unified, charting-algorithm-agnostic discretization framework? (RQ3)*

Table 3: This table presents the basis-kernels for the GEM-CNN+ (and EMAN+) convolution with integrated trainable phase weights. The upper table presents the basis-kernels for the neighborhood aggregation term. We set $\alpha_{\pm}^m = (\gamma' \pm \gamma)\theta + \beta_m$ with β_m being the output channel related *trainable* phase weight, and γ, γ' being the input- and output rotation orders of the features. The lower table presents the basis-kernels for the self-connection term. The original GEM-CNN kernels are obtained by setting $\beta_m = 0$.

neigh.	$s=1$	$s=2$	$s=3$	$s=4$
$\rho_0 \to \rho_0$	(1)	-	-	-
$\rho_\gamma \to \rho_0$	$\begin{pmatrix}\cos(\gamma\theta+\beta_m) & \sin(\gamma\theta+\beta_m)\end{pmatrix}$	$\begin{pmatrix}\sin(\gamma\theta+\beta_m) & -\cos(\gamma\theta+\beta_m)\end{pmatrix}$	-	-
$\rho_0 \to \rho_{\gamma'}$	$\begin{pmatrix}\cos(\gamma'\theta+\beta_m)\\ \sin(\gamma'\theta+\beta_m)\end{pmatrix}$	$\begin{pmatrix}\sin(\gamma'\theta+\beta_m)\\ -\cos(\gamma'\theta+\beta_m)\end{pmatrix}$	-	-
$\rho_\gamma \to \rho_{\gamma'}$	$\begin{pmatrix}\cos\alpha_-^m & -\sin\alpha_-^m\\ \sin\alpha_-^m & \cos\alpha_-^m\end{pmatrix}$	$\begin{pmatrix}\sin\alpha_-^m & \cos\alpha_-^m\\ -\cos\alpha_-^m & \sin\alpha_-^m\end{pmatrix}$	$\begin{pmatrix}\cos\alpha_+^m & \sin(\alpha_+^m)\\ \sin\alpha_+^m & -\cos\alpha_+^m\end{pmatrix}$	$\begin{pmatrix}-\sin\alpha_+^m & \cos\alpha_+^m\\ \cos\alpha_+^m & \sin\alpha_+^m\end{pmatrix}$
self				
$\rho_0 \to \rho_0$	(1)	-	-	-
$\rho_\gamma \to \rho_\gamma$	$\begin{pmatrix}\cos\beta_m & -\sin\beta_m\\ \sin\beta_m & \cos\beta_m\end{pmatrix}$	$\begin{pmatrix}\sin\beta_m & \cos\beta_m\\ -\cos\beta_m & \sin\beta_m\end{pmatrix}$	-	-
$\rho_\gamma \to \rho_{\gamma'}$	$\begin{pmatrix}0 & 0\\ 0 & 0\end{pmatrix}$	-	-	-

In Section 6, we have reformulated both non gauge-equivariant and gauge-equivariant surface convolutions in terms of a common discretization of the signal-pullback, potentially with preceding parallel transport, in a self-connection & neighborhood aggregation scheme over template vertices. In particular, we phrased each convolution such that it describes how a single output element is computed instead of the entire output embedding, enabling a direct comparison among existing surface convolutions.

Due to relying on the same signal-pullback, any surface convolution leaves only the choice of its defined trainable parameters open that constitute the template. The common signal-pullback together with a feature-wise description of the convolution thus enforces *unified notation across the discretizations of all considered surface convolutions* up to the definition of trainable weights.

We emphasize this shared computational pipeline among all discussed surface convolutions by abstracting from the particular choice of weight matrices, yielding the following *charting-method agnostic surface CNN framework*:

$$\boldsymbol{x}_m = \sum_{n=1}^{d_{\text{in}}} \boldsymbol{W}_{mn} \begin{pmatrix} a_n^{(0,0)} \\ b_n^{(0,0)} \end{pmatrix} + \sum_{r=1}^{N_\tau} \sum_{a=1}^{N_\theta} \sum_{n=1}^{d_{\text{in}}} \boldsymbol{W}_{mn}^{(r,a)} \begin{pmatrix} a_n^{(r,a)} \\ b_n^{(r,a)} \end{pmatrix} + \boldsymbol{\beta} \ , \tag{24}$$

where $\boldsymbol{W}$ and $\boldsymbol{W}^{(r,a)}$ represent the linear mappings learned by self-connections and neighborhood-aggregations, and $\boldsymbol{\beta}$ denotes a trainable bias, which is used by non-gauge-equivariant surface convolutions. The values $a_n^{(0,0)}$ and $b_n^{(0,0)}$ denote the scalar components for the n-th feature at the convolution center, which is never parallel-transported. On the other hand, $a_n^{(r,a)}$ and $b_n^{(r,a)}$ describe the n-th interpolated feature at template vertex (τ_r, θ_a) which is parallel transported in case gauge-equivariant surface convolutions are instantiated. Table 4 summarizes how to select layer attributes to build the desired convolution.

In summary, we can reformulate all discussed surface convolutions as a particular instance of a common surface CNN framework which builds on a unified discretization of the signal-pullback, emphasizing that different convolutions differ only in their choice of weight matrices and whether parallel transport is applied.

Activation functions. When using non gauge-equivariant surface convolutions, one may apply arbitrary activation functions element-wise on the resulting feature vectors. In contrast, activation functions used subsequent to gauge-equivariant surface convolutions must pay attention to the gauge-equivariance constraint (cf. Definition 4.3) when gauge-equivariance across multiple layers is desired.

Table 4: This table summarizes what weighting functions, kernels $\boldsymbol{W}_{mn}$ and $\boldsymbol{W}_{mn}^{(r,a)}$ which causes the layer to expect input features a and b. Gauge-equivariant surface convolutions always use *parallel transported* feature vectors $\widetilde{\boldsymbol{x}}^{(r,a)}$ at the template vertices and *non-transported* feature vectors $\boldsymbol{x}^{(0,0)}$ for self-connections.

Architecture	Weighting	$\boldsymbol{W}_{mn}$	$\boldsymbol{W}_{mn}^{(r,a)}$	$a_n^{(r,a)}$	$b_n^{(r,a)}$
ISC	Dirac	$\boldsymbol{T} \in \mathbb{R}^{2\times 2}$	$\boldsymbol{T}^{(r,a)} \in \mathbb{R}^{2\times 2}$	$\boldsymbol{x}_n^{(r,a)}$	$\boldsymbol{x}_{n+1}^{(r,a)}$
GCNN	Gaussian	$\boldsymbol{T} \in \mathbb{R}^{2\times 2}$	$\boldsymbol{T}^{(r,a)} \in \mathbb{R}^{2\times 2}$	$\boldsymbol{p}_n^{(r,a)}$	$\boldsymbol{p}_{n+1}^{(r,a)}$
HSN	Dirac	$w_{mn}\boldsymbol{\Phi}_m^{\text{self}}$	$w_{mn}^{(r)}\boldsymbol{\Phi}_{am}^{\text{neigh}}$	$\text{Re}\left(\widetilde{\boldsymbol{x}}_n^{(r,a)}\right)$	$\text{Im}\left(\widetilde{\boldsymbol{x}}_n^{(r,a)}\right)$
GEM-CNN	Dirac	$\sum_s w_{mn}^s \boldsymbol{K}_{mn}^{s,\text{self}}$	$\sum_s w_{mn}^s \boldsymbol{K}_{mn}^{s,\text{neigh}}(\theta_a)$	$\text{Re}\left(\widetilde{\boldsymbol{x}}_n^{(r,a)}\right)$	$\text{Im}\left(\widetilde{\boldsymbol{x}}_n^{(r,a)}\right)$
GEM-CNN+	Dirac	$\sum_s w_{mn}^s \widetilde{\boldsymbol{K}}_{mn}^{s,\text{self}}$	$\sum_s w_{mn}^{(s,r)} \widetilde{\boldsymbol{K}}_{mn}^{s,\text{neigh}}(\theta_a)$	$\text{Re}\left(\widetilde{\boldsymbol{x}}_n^{(r,a)}\right)$	$\text{Im}\left(\widetilde{\boldsymbol{x}}_n^{(r,a)}\right)$
EMAN	Dirac	$\alpha^{\text{self}} \sum_s w_{mn}^s \boldsymbol{K}_{mn}^{s,\text{self}}$	$\alpha_{ra}^{\text{neigh}} \sum_s w_{mn}^s \boldsymbol{K}_{mn}^{s,\text{neigh}}(\theta_a)$	$\text{Re}\left(\widetilde{\boldsymbol{x}}_n^{(r,a)}\right)$	$\text{Im}\left(\widetilde{\boldsymbol{x}}_n^{(r,a)}\right)$
EMAN+	Dirac	$\alpha^{\text{self}} \sum_s w_{mn}^s \widetilde{\boldsymbol{K}}_{mn}^{s,\text{self}}$	$\alpha_{ra}^{\text{neigh}} \sum_s w_{mn}^{(s,r)} \widetilde{\boldsymbol{K}}_{mn}^{s,\text{neigh}}(\theta_a)$	$\text{Re}\left(\widetilde{\boldsymbol{x}}_n^{(r,a)}\right)$	$\text{Im}\left(\widetilde{\boldsymbol{x}}_n^{(r,a)}\right)$

In this work, we consider β-ReLU activation applied on the magnitude of each complex-valued feature (Weiler et al., 2018):

$$\boldsymbol{\phi}(\boldsymbol{x}_m) = \text{ReLU}\left(||\boldsymbol{x}_m||_2 - \beta\right) \frac{\boldsymbol{x}_m}{||\boldsymbol{x}_m||_2}\ , \tag{25}$$

where β is a learnable parameter subtracted from the feature magnitude. This activation only alters the norm of a feature while it preserves its current orientation and thus maintains gauge-equivariance.

Programming. We have implemented the parametric framework (cf. equation 24) and instantiated the discussed surface convolutions using the framework in *our library*[1]. Moreover, the library provides all necessary utilities for studying surface convolutions such as charting algorithms, algorithms to efficiently compute barycentric coordinates, magnitude activation functions and local shape descriptors that can be used as initial input features for the vertices.

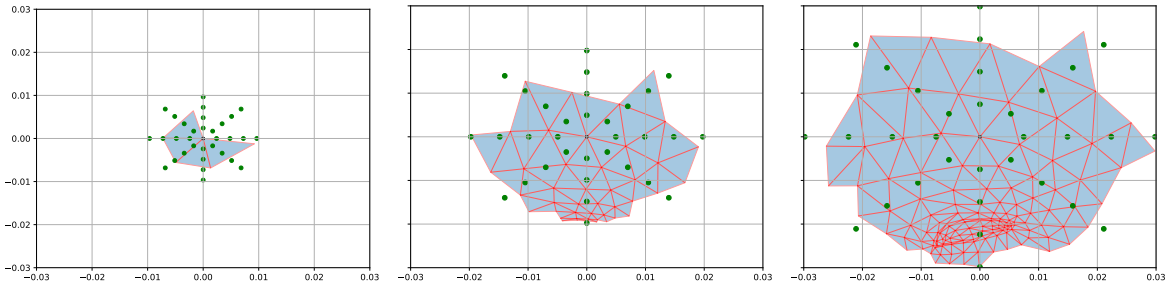
7 Ablation Study

In Section 3, we raised two research questions regarding whether surface CNN performance depends on the used charting algorithm (RQ1) and how the chart radius affects the performance of surface CNNs (RQ2). In this section, we systematically conduct experiments to answer these research questions using the surface CNN framework from equation 24. For this, we consider two problem settings: predicting point-correspondences between FAUST meshes (Bogo et al., 2014) and shape classification on ModelNet10 (Wu et al., 2015).

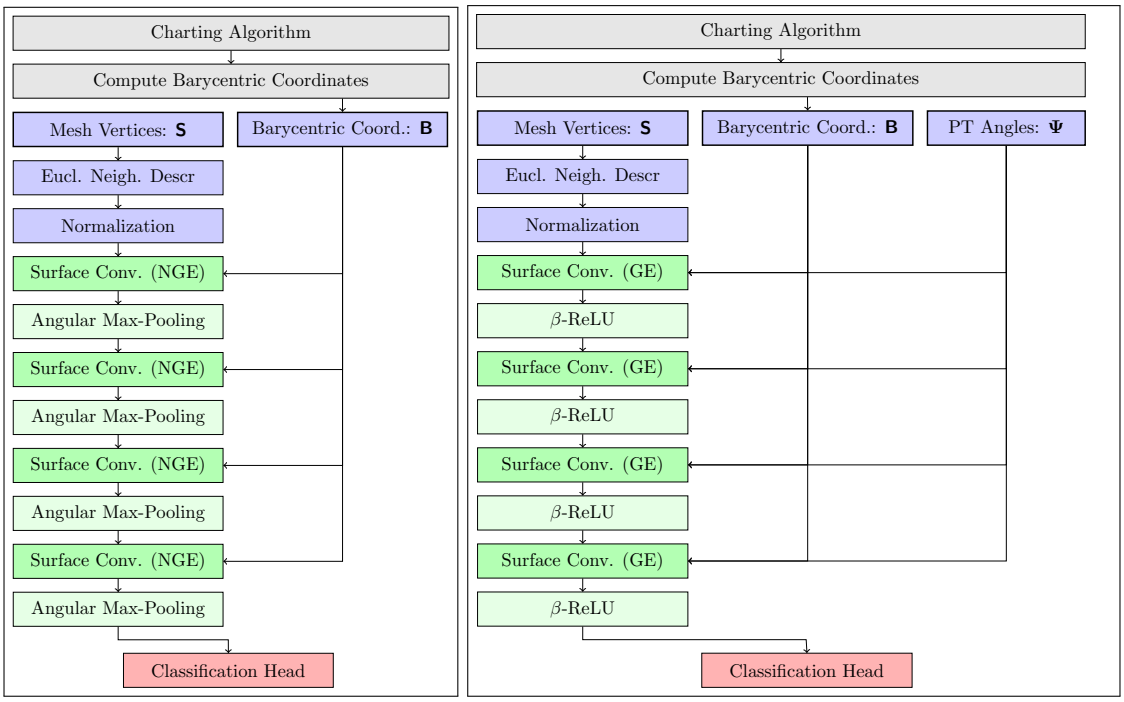
Chart and template radii. The chart radius determines the size of the neighborhood in which a trainable template is placed, and thus also the amount of locally available information to the convolution. To study the effect of varying the chart radius on surface CNNs, we compute multiple sets of charts. Each set is computed using one charting algorithm and one maximum allowed (geodesic) chart radius c_R. While the majority of charts in a set have a radius close to c_R, individual chart radii may be smaller depending on the local surface geometry.

For each set, we define one template radius τ_R (cf. Definition 6.1) as the median of all given chart radii contained in the considered set. Considering that the majority of chart radii in a set are approximately c_R

[1]The code is attached as supplementary material. The library will be made public on GitHub upon paper acceptance.



(a) An illustration of charts with increasing maximum allowed chart radii and a discretized template (green dots, $N_\tau = 4$ and $N_\theta = 8$) whose radius is set to the median geodesic extension of all local charts in the FAUST dataset (Bogo et al., 2014). Barycentric coordinates are computed for each green dot and stored in a barycentric coordinates tensor $\mathbf{B}$ which is passed as additional input to the network architecture. Template vertices outside any triangle are zeroed.



(b) The surface CNN architecture *blueprints* for all experiments in our ablation study. The "Surface Conv."-blocks are substituted with the studied surface convolution operations from Table 4. The "Classification Head" changes depending on the task. Furthermore, depending on whether we use non gauge-equivariant or gauge-equivariant surface convolutions, we either instantiate the left or right blueprint. **[Left]** The non gauge-equivariant architecture (NGE). Each convolution uses point-wise ReLU-activation and is followed by angular max-pooling (AMP). The architecture requires 3D-coordinates of the mesh vertices and barycentric coordinates as input. **[Right]** The gauge-equivariant architecture (GE). Instead of point-wise ReLU activation and AMP, each convolution is followed by a β-ReLU activation. This architecture expects the parallel transport (PT) angles $\mathbf{\Psi}$ as additional input.

Figure 6: An illustration of the preprocessing and surface CNN architecture blueprints for our ablation study.

we have $\tau_R \approx c_R$. Since τ_R represents a single quantity for a set of varying chart radii, we thus report test statistics in dependence of used template radii which, because of $\tau_R \approx c_R$, allows us to draw conclusions regarding the impact of chart radii on surface CNNs (RQ2). The template resolution is set to $N_\tau = 4$ and $N_\theta = 8$ in any of our experiments.

In practice, we begin our preprocessing by normalizing all triangle meshes to have a geodesic diameter of 1, which allows us to express the chart radius as a percentage of the geodesic diameter. Then, let $C \in \{\text{DGPC}, \text{FMM}, \text{HDM}, \text{TP}\}$ be a charting algorithm that takes a mesh M, a vertex in the mesh $v \in M$ and a maximum chart radius c_R as input, returning a chart $C(M, \boldsymbol{v}, c_R)$ centered in v as output. We compute the following sets of charts:

$$A_{c_R} = \{C(M, \boldsymbol{v}, c_R) \,|\, \boldsymbol{v} \in M, M \in \mathcal{D}\} \text{ for } c_R \in \{0.01, 0.02, ..., T_R\} \ , \tag{26}$$

where $\mathcal{D}$ is either the FAUST or the ModelNet10 dataset. T_R is the upper bound for chart radii. We define the template radius dependent on one set of charts A_{c_R}:

$$\tau_R := \text{median}\big(\{\text{radius}\,(C(M, \boldsymbol{v}, c_R)) \,|\, C(M, \boldsymbol{v}, c_R) \in A_{c_R}\}\big) \ . \tag{27}$$

Exact numbers are presented in the respective sub-section of each benchmark. Figure 6 (a) illustrates local charts and templates exemplarily for three increasingly large maximum chart radii c_R.

Network architecture. In order to quantify the behavior of surface CNNs, we are going to train a total of 420 different surface CNN configurations over the two datasets. We deliberately do not include intermediate batch-normalizations, skip-connections or other architectural design choices that may have an effect on the training results since we want to study the relative performance of surface convolutions and minimize unrelated secondary effects.

For both datasets, we use the same surface CNN architecture blueprints depicted in Figure 6 (b). The blueprints are instantiated once with each convolutions listed in Table 4. Network architectures using ISC, GCNN or HSN convolutions use $32 \to 64 \to 96 \to 64$ as intermediate output dimensions. GEM-CNNs, EMANs and their +-extensions always consider the input features to be gauge-transformation invariant features, i.e., of type ρ_0, and use the following intermediate output feature types for the FAUST experiments:

$$\underbrace{8(\rho_1 \oplus \rho_2)}_{\boldsymbol{x} \in \mathbb{R}^{32}} \to \underbrace{16(\rho_1 \oplus \rho_2)}_{\boldsymbol{x} \in \mathbb{R}^{64}} \to \underbrace{24(\rho_1 \oplus \rho_2)}_{\boldsymbol{x} \in \mathbb{R}^{96}} \to \underbrace{16(\rho_1 \oplus \rho_2)}_{\boldsymbol{x} \in \mathbb{R}^{64}} \ , \tag{28}$$

and the following intermediate output features for the ModelNet10 experiments:

$$8(\rho_1 \oplus \rho_2) \to 16(\rho_1 \oplus \rho_2) \to 24(\rho_1 \oplus \rho_2) \to \underbrace{64\,\rho_0}_{\boldsymbol{x} \in \mathbb{R}^{64}} \ . \tag{29}$$

The only difference in feature types between the dataset-related architectures occurs in the last layer: in difference to vertex-level predictions in the FAUST experiments using a linear layer, the architectures used in the ModelNet10 experiments use a permutation invariant DeepSet classification head (Zaheer et al., 2017) to classify entire shapes from local feature vectors. This layer sums over the feature vectors and subsequently classifies the sum using a linear layer.

Since local feature vectors are represented with respect to their own local gauge, we would need to parallel transport all features into a common basis before summation. This is not required for gauge-invariant features of type ρ_0. Furthermore, when testing different feature types, we found that ρ_γ with too large rotation orders ($\gamma \geq 3$) caused the training to become unstable. Hence, we kept γ small by selecting $\gamma \in \{0, 1, 2\}$.

Lastly, while the original input to the surface CNNs are merely the 3D coordinates of the mesh vertices, we compute Euclidean neighborhood feature vectors for the surface convolutions at the beginning of each forward pass and normalize them in each feature dimension. To this end, we use z-score normalization and determine the mean as well as the standard deviation for each dimension in the preprocessing stage. For the construction of the feature vectors we consider the k nearest neighbors of a mesh vertex $\boldsymbol{x}$, calculate their relative positions $\boldsymbol{x}_k - \boldsymbol{x}$ and concatenate them to one feature vector. For all experiments, we set $k = 32$, yielding 96-dimensional input vectors.

7.1 FAUST

Dataset. In our first experiment, we test surface CNNs on the FAUST dataset Bogo et al. (2014). This dataset contains triangle meshes of ten human subjects in ten different poses, yielding 100 ordered triangle meshes in total. Each mesh consists of 6890 vertices. We divide the dataset into a training, validation, and test split. The training split contains the first 70 meshes, the validation split contains the subsequent 10 meshes and the test split contains the last 20 meshes.

Learning problem. Given a reference shape $\mathcal{R}$ and a query shape $\mathcal{Q}$, for every vertex $\boldsymbol{x}_l \in \mathcal{Q}$ the surface CNN has to predict index k_l of the corresponding vertex $\boldsymbol{x}_{k_l} \in \mathcal{R}$. We select the first mesh of the dataset to be $\mathcal{R}$ throughout all experiments. Given that $\mathcal{R}$ has 6890 vertices, we train a classifier $\phi : \mathcal{D} \times \mathcal{I} \to \mathbb{R}^{6890}$, whereby $\mathcal{D}$ is the set of all FAUST shapes and $\mathcal{I}$ an index set referring to one out of 6890 vertices. Note that the same index k_l may be predicted multiple times. While ϕ estimates vertex-wise predictions, it requires the whole shape as input to aggregate neighborhood information for the prediction. In practice, we use a fully connected layer with linear activation as the classification head.

Preprocessing. FAUST-meshes are *registered*. That is, originally any vertex $\boldsymbol{x}_k$ of any query mesh $\mathcal{Q}$ corresponds to vertex $\boldsymbol{x}_k$ in any reference shape $\mathcal{R}$. While in this benchmark predictions are made on a vertex-level, we are still concerned that this fixed vertex ordering introduces advantageous learning effects that cannot be assumed to exist in the general case. Hence, we randomly permute the vertex ordering of the query meshes $\mathcal{Q}$ once during preprocessing. We compute local charts for five different max-radii c_R in between of 1% and 5% of the geodesic diameter, define the template vertices accordingly (cf. equation 27) and determine the barycentric coordinates for the defined template vertices as described in Section 6.1. Lastly, we compute angles for parallel transports using the vector heat method (Sharp et al., 2019).

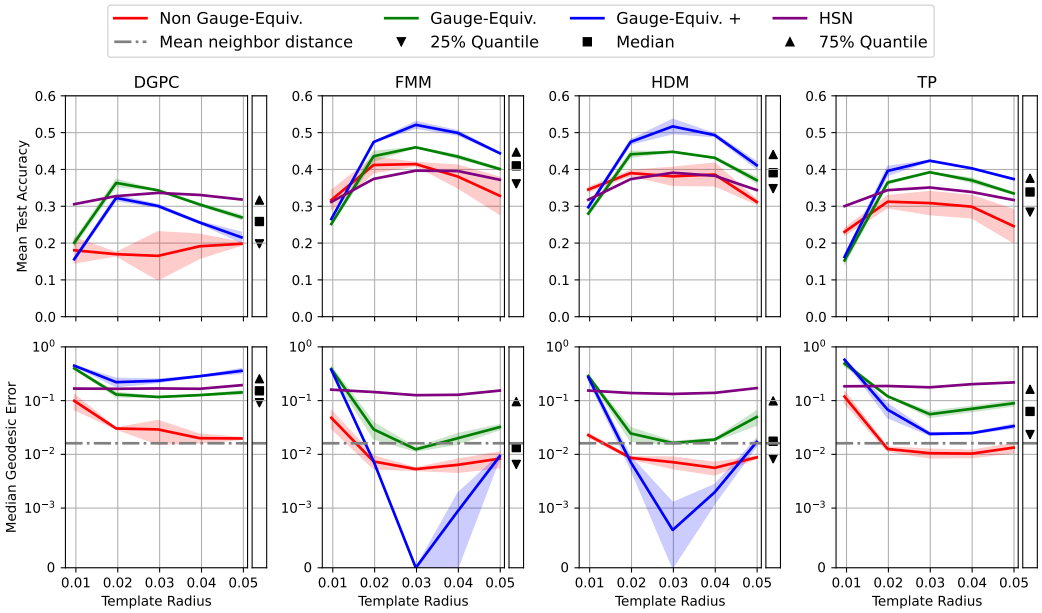
Loss and optimizer configuration. We minimize the cross-entropy loss using the Adam optimizer (Kingma & Ba, 2015) and apply an exponential decay rate onto the learning rate. The learning rate and decay rate are found at the hand of the validation set via Bayesian optimization and knowledge from experience (cf. Appendix, Table 12, left). They vary among the used network architectures but are equal across different charting configurations. The amount of trainable weights for each network architecture and inference times are reported in Table 5. In total, we train 140 different surface CNNs in this FAUST benchmark.

Observations. Figure 7 (a) shows accuracies and median geodesic errors for all point-correspondence predictions over template radii computed for the test split. While accuracy measures only the fraction of exactly correct predicted correspondences, geodesic error quantifies how far predicted correspondences are from their ground truth. We have marked the mean neighbor distance for vertices in the reference shape (i.e., mean edge-length) in the geodesic error plots as a reference point for interpreting how close the predictions lie to the actual ground truth correspondence.

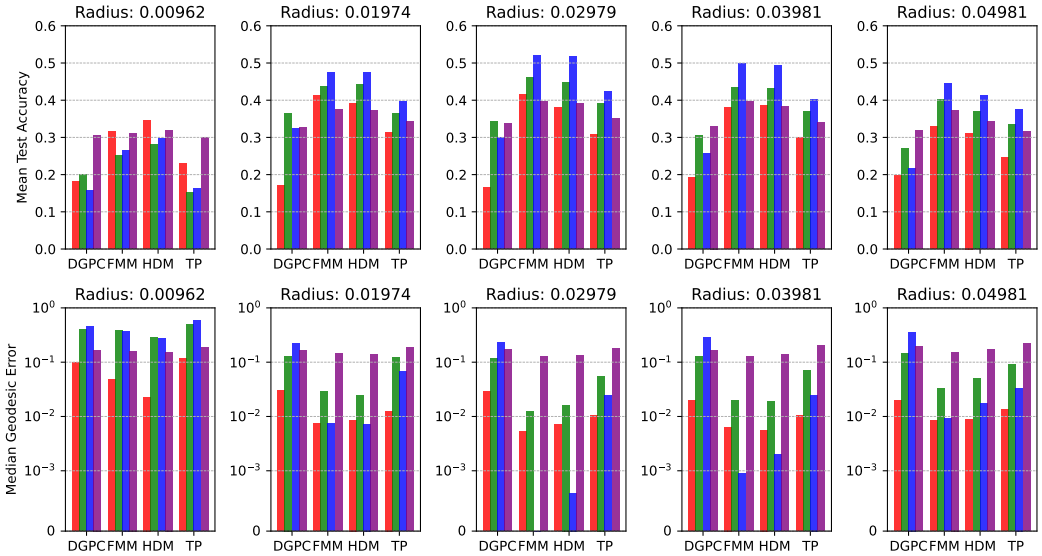
Considering that the template radius serves as a proxy for chart radii, we can observe that the majority of surface CNNs exhibit "*arc-performance*" over chart radii in their accuracy scores. Arc-performance is characterized by worse performance at small or large chart radii and good performance when using mid-range radii. The geodesic error plots qualitatively illustrate reciprocal arcs compared to the test accuracy

Table 5: The amount of weights and inference times for the prediction of 6890 point-correspondences (i.e., one prediction for each vertex of one FAUST shape) using an NVIDIA A40 graphics card. The classification heads share the same amount of weights across architecture types. Therefore, we once report the total number of trainable parameters and once the number of trainable parameters without the classification head. Bold numbers indicate either the least amount of weights or the fastest inference time.

	ISC	GCNN	HSN	GEM-CNN	GEM-CNN+	EMAN	EMAN+
# weights	1 022 763	1 022 763	**470 059**	478 891	540 459	581 291	642 859
# weights (without classification head)	574 720	574 720	**22 016**	30 848	92 416	133 248	194 816
inference time	**0.09s**	**0.09s**	0.12s	0.16s	0.20s	0.16s	0.21s



(a) Test performances over template radii grouped by charting algorithms.



(b) Test performances for each charting algorithm grouped by template radii.

Figure 7: Mean scores for test accuracies and median geodesic errors in the *FAUST* point-correspondence benchmark over network architecture *classes* using $N_\tau = 4$ and $N_\theta = 8$. The **non gauge-equivariant** scores describe ISC and GCNN networks. The **gauge-equivariant** scores describe GEM-CNNs and EMANs. The **gauge-equivariant+** scores describe GEM-CNNs+ and EMANs+. The **purple** scores describe harmonic surface networks. The median performance, the lower 25% and upper 75% quantiles over all architecture types and template radii per metric and charting-method is illustrated in the inset scatter plots.

Table 6: Median and quantile statistics for the FAUST benchmark. IQR refers to the inter quantile range in between the 25% and 75% quantiles. Bold values indicate the best scores in their respective statistic. Moreover, ↑ indicates that larger values are better and ↓ shows that lower values are better.

metric	statistic	DGPC	FMM	HDM	TP
test accuracy	25% quantile ↑	0.19794	**0.36087**	0.34789	0.28405
	median ↑	0.25858	**0.40976**	0.39107	0.33881
	75% quantile ↑	0.31682	**0.44724**	0.44059	0.37619
	IQR ↓	0.11889	**0.08638**	0.09270	0.09214
geodesic error	25% quantile ↓	0.09147	**0.00649**	0.00828	0.02336
	median ↓	0.15220	**0.01337**	0.01757	0.06296
	75% quantile ↓	0.25566	**0.09578**	0.09891	0.16279
	IQR ↓	0.16419	**0.08929**	0.09064	0.13943

arcs with the exception of those from HSNs. That is, the majority of surface CNNs achieve low median geodesic errors in their point-correspondence predictions when accuracies are high and vice versa. The median geodesic errors of HSNs, on the other hand, appear to be mostly invariant to varying chart radii.

To quantify the effect of charting algorithms on the performance of surface CNNs, we have added an inset axes to the right of each plot in Figure 7 (a). These show the 25% and 75% quantiles together with the median over all measured test accuracies and median geodesic errors. Their exact values are additionally presented in Table 6. Comparing the median values across charting algorithms yields a ranking of charting algorithms for training surface CNNs on the FAUST benchmark. The best surface CNNs have been trained when using FMM. HDM causes slightly worse performing models, followed by surface CNNs trained on tangent projections. The least well performing models used the DGPC algorithm. Comparing the inter quantile ranges (IQR) between the 25% and 75% quantiles shows varying robustness levels of surface CNNs in dependence of the chosen charting algorithm. The models trained on FMM charts not only achieved the best performances, but also the most robust. Surface CNNs trained on tangent projections and HDM charts cause slightly larger IQR and therefore more varying results. The DGPC algorithm causes the largest IQR and therefore the least robust training results.

ANOVA hypothesis tests. So far, we have analyzed the experiment results qualitatively at the hand of Figure 7 (a). Now we want to investigate whether changing the underlying charting algorithm (RQ1) or varying the chart radius (RQ2) has a statistical significant impact on the downstream performance of surface CNNs. To this end, we conduct two sets of ANOVA hypothesis tests: the first set conducts 10 hypothesis tests, two for each fixed template radius (considering both test metrics), using charting algorithms as treatments. The second set conducts 8 hypothesis tests, each of which fixes the charting algorithm and uses template radii as treatments. Each test corresponds to a table that crosses surface CNNs and preprocessing configurations (cf. Appendix, Tables 13 and 14). Each of these tables is visualized in one sub-plot of Figure 7 (a) or (b). The treatments are shown along the x-axis. Each row within a table corresponds to the performance over the treatments for one particular surface CNN.

Table 7: The resulting p-values for the ANOVA hypothesis tests for the groups depicted in Figure 7.

template radius	p-value (acc.)	p-value (geodesic err.)
0.00962	< 0.01	> 0.1
0.01974	< 0.01	< 0.05
0.02979	< 0.01	< 0.01
0.03981	< 0.01	< 0.05
0.04981	< 0.01	< 0.05

charting method	p-value (acc.)	p-value (geodesic err.)
DGPC	> 0.1	< 0.1
FMM	< 0.01	< 0.01
HDM	< 0.01	< 0.01
TP	< 0.01	< 0.01

The first set of hypothesis tests targets our first research question: Are the downstream performances of surface CNNs affected by the charting algorithm? For each template radius, we treat each surface CNN as an individual subject. The treatments are the 4 charting algorithms DGPC, FMM, HDM and TP. The response variable represents either the achieved accuracy or the geodesic error for a given charting algorithm (cf. Figure 7, b). Our null hypothesis claims that the charting algorithm does not affect the response variable (i.e., test metric) for a fixed template radius. However, as reported by Table 7 (left), the p-values for the test accuracy tests were consistently < 0.01, making the null hypothesis highly unlikely. For geodesic errors we have p-values which are at least < 0.05 for all cases except the smallest template radius. Hence, the null hypothesis becomes unlikely with respect to geodesic errors once large enough template radii are used. Since we measured better performances when using larger radii, this would therefore be the prominent case in practice. We therefore reject the null-hypothesis in most cases and conclude that the selection of the charting algorithm has a statistically significant effect on the performance of surface CNNs in the FAUST benchmark.

The second set of hypothesis tests investigates our second research question: How does the chart radius affect the performance of surface CNNs? We have described arc-performance over chart radii before. The hypothesis test now serve to determine whether these changes over chart radii are statistically significant. For each charting algorithm, we treat each surface CNN as an individual subject (Figure 7, a). Each subject is evaluated under 5 treatments, i.e., the different template radii that serve as a proxy for a set of chart radii. As before, the response variables are the test accuracy or the geodesic error for a given template radius. In this test, the null hypothesis claims that the template radius does not affect the response variable for a fixed charting algorithm. However, besides the p-values for the hypothesis test regarding the DGPC algorithm, all p-values for both response variables are smaller than 0.01 (cf. Table 7, right), enough to reject the null hypothesis. The p-values for the DGPC algorithm indicate that the template radius has a statistically relevant impact on the geodesic error, whereas the classification accuracy is not significantly affected. Nevertheless, the majority of hypothesis tests indicate that the template radius and thus the chart radius does affect surface CNNs. This represents empirical evidence for the chart radius being a critical parameter for surface CNNs in the FAUST benchmark.

7.2 ModelNet10

Dataset. ModelNet10 (MN10) (Wu et al., 2015) contains about 5 000 triangle meshes across 10 classes (bathtub, bed, chair, desk, dresser, monitor, night stand, sofa, table and toilet). After preprocessing, the meshes contain between 500 to 6 000 vertices. MN10 defines a training and a test split. While we maintain the predefined test split, we use 90% of the predefined training split for training and the residual 10% as a validation split to find hyperparameters.

Learning problem. Given a query shape $\mathcal{Q}$, the surface CNN ϕ has to predict $\mathcal{Q}$'s class y. That is, we train a classifier $\phi : \mathcal{D} \to \mathcal{C}$, where $\mathcal{D}$ is the set of all MN10 shapes and $\mathcal{C}$ the set of all classes. In contrast to the FAUST benchmark, the surface CNN needs to aggregate all vertex embeddings before it can make a prediction for the shape. This mechanism is implemented in the form of a permutation-invariant DeepSet layer (Zaheer et al., 2017) which predicts the class y from the sum over all vertex embeddings:

$$\hat{y} = \phi\left(\sum_i \psi(f_i)\right) . \tag{30}$$

Prior to the sum, a *gauge-equivariant* surface CNN is required to either (a) parallel transport all final vertex embeddings $\psi(f_i)$ into a common gauge or (b) predict gauge-invariant embeddings (type ρ_0) in its last layer. In this work, we decided to use the latter approach (cf. equation 29).

Preprocessing. We use the algorithm of Huang et al. (2018) to generate manifold meshes, i.e. watertight meshes with edges being associated to exactly two triangles, from the original MN10 shapes. Each mesh is then normalized to have a geodesic diameter of 1. We define 10 different max radii c_R for the local charts varying from 1% to 10% with respect to the geodesic diameter. The computation of local charts is followed

Table 8: The amount of weights and inference times for predicting the class of one ModelNet10 shape with 6042 vertices using an NVIDIA A40 graphics card. The classification heads share the same amount of weights across architecture types. Therefore, we once report the total number of trainable parameters and once the number of trainable parameters without the classification head. Bold numbers indicate either the least amount of weights or the fastest inference time.

	ISC	GCNN	HSN	GEM-CNN	GEM-CNN+	EMAN	EMAN+
# weights	575 370	575 370	**22 666**	40 746	120 778	173 866	253 898
# weights (without classification head)	574 720	574 720	**22 016**	40 096	120 128	173 216	253 248
inference time	**0.08s**	**0.08s**	0.13s	0.17s	0.18s	0.17s	0.19s

by the definition of template vertices, the computation of barycentric coordinates as described in Section 6.1, and the computation of angles for parallel transports using the vector heat method (Sharp et al., 2019).

Loss and optimizer configuration. For training, we minimize cross-entropy using the Adam optimizer (Kingma & Ba, 2015) with an exponential decay applied to the learning rate. Thereby, learning rate and decay rate have been found using Bayesian optimization and knowledge from experience. Similarly to the FAUST ablations, the hyperparameters are equal for different charting configurations but differ across network architectures. They are reported in Table 12 (right). The number of weights and inference times per network architecture for meshes with 6 042 vertices are reported in Table 8. Overall, our MN10 benchmark encompasses the training of 280 different surface CNNs.

Observations. Regarding the effect of template radii on the performance of surface CNNs, Figure 8 (a) shows a trend of improved performance for increasing template radii. The most substantial improvements appear at smaller template radii. Particularly gauge-equivariant networks and their +-extensions appear to be sensitive to small template radii. In contrast to the FAUST benchmark, we cannot observe performance degradation at large template radii and thus have no arc-performance behavior for any architecture class in the MN10 benchmark.

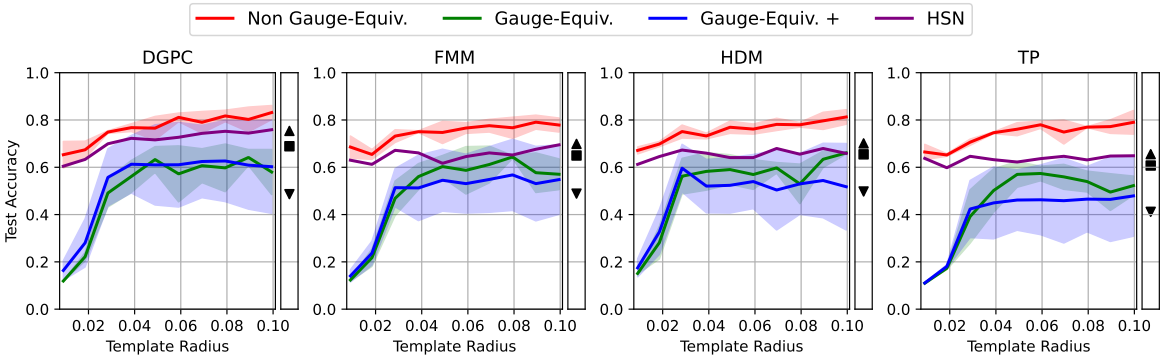
Similar to the FAUST benchmark, we can additionally quantify differences among the charting algorithms by ranking them according to the observed median and IQR values for the test-statistics. Here, the DGPC algorithm yields the best training results, followed by HDM. FMM produces slightly worse results. Tangent plane projections cause the lowest median test accuracy. Using IQR as a measure for stability, we see that HDM creates the most stable results, followed by FMM, then TP and lastly DGPC.

ANOVA hypothesis tests. Analogously to our FAUST benchmark, we conduct ANOVA hypothesis tests. The first set of tests addresses the effects of varying charting algorithms for fixed template radii (RQ1). The second set of tests examines the effects of varying template radii for fixed charting algorithms (RQ2). The associated ANOVA tables are presented in the Appendix (cf. Tables 15 and 16). Each table corresponds to a sub-plot in Figure 8 (a) or (b).

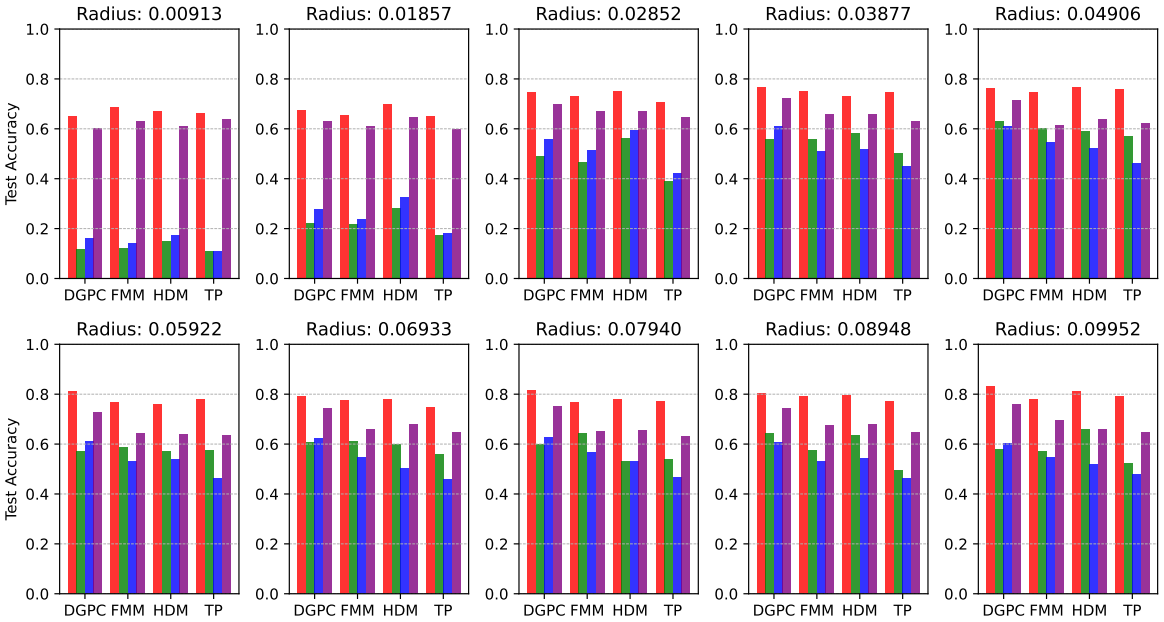
According to the p-values for the set of hypothesis tests that address RQ1, which are shown per template radius in Table 10 (left), we cannot observe statistically significant effects of the charting algorithms unlike

Table 9: Median and quantile statistics for the ModelNet10 benchmark. IQR refers to the inter quantile range in between the 25% and 75% quantile. Bold values indicate the best scores in their respective statistic. Furthermore, ↑ indicates that larger values are better and ↓ shows that lower values are better.

metric	statistic	DGPC	FMM	HDM	TP
test accuracy	25% quantile ↑	0.48623	0.48844	**0.49752**	0.41162
	median ↑	**0.68888**	0.64978	0.65474	0.60738
	75% quantile ↑	**0.75385**	0.69906	0.70127	0.65556
	IQR ↓	0.26762	0.21063	**0.20374**	0.24394



(a) Test accuracies over template radii grouped by charting algorithms.



(b) Test accuracies for each charting algorithm grouped by template radii.

Figure 8: Mean scores for test accuracies and median geodesic errors in the *ModelNet10* shape classification benchmark over network architecture *classes* using $N_\tau = 4$ and $N_\theta = 8$. The classes are similar to the classes in Figure 7.

Table 10: The resulting p-values for the ANOVA hypothesis tests for the groups depicted in Figure 8.

template radius	p-value (acc.)
0.009127	> 0.1
0.018571	> 0.1
0.028523	> 0.1
0.038772	> 0.1
0.049063	> 0.1
0.059215	> 0.1
0.069327	> 0.1
0.079404	> 0.1
0.089476	> 0.1
0.099523	> 0.1

charting algorithm	p-value (acc.)
DGPC	< 0.01
FMM	< 0.05
HDM	< 0.05
TP	< 0.1

in the FAUST benchmark. Regarding RQ2, the p-values reported for fixed charting algorithms and varying chart radii (cf. Table 10, right) are below a significance threshold for all charting algorithms. This highlights a statistically significant impact of the template radius, and thus, by construction, the chart radius, on test accuracies of surface CNNs.

7.3 Discussion

We start this section by discussing our first research question in light of the ablation results: *Are the downstream performances of surface CNNs affected by their chosen charting algorithm? (RQ1)*

Discussion RQ1. FMM, HDM and DGPC search for distance functions that satisfy the Eikonal equation. However, the algorithms vary in their way of approximating local distances (cf. Table 2). Moreover, tangent plane projections can only be considered an approximation to local surface charting by construction. These differences can cause varying geometric distortions depending on the chosen charting algorithm. In particular, errors in geodesic or angular coordinates cause the signal-pullback to extract noisy features. These noisy features are then used by the convolution, suggesting that different charting algorithms lead to varying downstream performances.

Our observations in both dataset benchmarks allow to rank charting algorithms based on their induced downstream performance and robustness of surface CNNs. This suggests that the choice of a charting algorithm matters for surface CNNs. However, we could verify *statistically significant differences* between charting algorithms only in the FAUST benchmark and not in the MN10 benchmark. We hypothesize that the magnitude of the downstream effects caused by charting algorithms *depends on the locality of the learning problem.*

In particular, global shape predictions, as required by the MN10 benchmark, do not directly rely on the embedding of a single vertex, but rather on the embeddings estimated for an entire subset of vertices that are particularly meaningful for the assignment of the ground truth class to the query shape. The importance of individual embeddings is therefore questionable in a global prediction task. Furthermore, from a technical perspective, the effects of local geometric distortions that propagate into local embeddings can be concealed by the aggregation over all embeddings (cf. equation 30), reducing their observable effect in the final shape prediction. Hence, we suspect that the downstream effects of local geometric distortions are less visible in global learning tasks compared to local learning tasks.

In summary, our FAUST benchmark provides empirical evidence that the *charting algorithm has a statistically significant effect on the downstream performance of surface CNNs.* However, we suspect that the magnitude of the downstream effect of the charting algorithm depends on the locality of the learning problem, as shown by the MN10 results.

In the next part of our discussion, we examine our ablation results w.r.t. our second research question: *How does the chart radius affect the performance of surface CNNs? (RQ2)*

Discussion RQ2. In contrast to Euclidean CNNs, surface CNNs do not infer the neighborhood size from the template size. The neighborhoods for surface convolutions are constructed using a charting algorithm that requires the chart's extent to be known beforehand as a hyperparameter. While too small neighborhoods might not carry enough information to solve the task, too large neighborhoods are more likely to be geometrically distorted or even completely invalid in case the injectivity radius is exceeded. This suggests the existence of a "sweet spot" which balances information contained in the neighborhoods and geometric stability. Most likely, this sweet spot is a value that is close to the true injectivity radius of the majority of surface points.

In our ablations, we define the template radius as the median of the set containing all chart radii associated with a given maximum radius and one charting algorithm (cf. equation 27). Thus, varying the template radius corresponds to varying the radii of the underlying charts, enabling us to use the template radius as a proxy for studying the effect of chart radii on surface CNN performance.

Our FAUST benchmark shows that the performance of surface CNNs over template radii in local prediction tasks is characterized by *arcs*: low performance for small and large radii and peak performance for mid-range radii. The most prominent example is the top-performing gauge-equivariant+ model that uses FMM as charting algorithm and uses about 3% of the geodesic diameter as the template radius, which represents the median of all investigated template radii.

Our global prediction task on the MN10 benchmark shows template radius dependent performance, too. Small template radii perform worse compared to larger template radii. However, instead of a performance drop, we only observe saturating performances for larger template radii.

Across both benchmark datasets, most hypothesis tests suggest a statistically significant impact of the template radius on the test metrics. Since the template radius serves as a proxy for chart radii, we conclude that the *chart radius is a critical parameter for surface CNNs*. Therefore, we argue that future experiments involving surface CNNs should treat the chart radius as a tunable hyperparameter, since inadequate chart radii can cause surface CNNs to substantially underperform. Our ablations suggest that good performing chart radii lie in between 1% and 5% of the geodesic diameter, which can be used as a reasonable starting point for future architectural design choices.

8 Limitations and Future Work

In this work, we analyzed downstream effects caused by varying charting algorithms and chart radii in a controlled discretization setting. This analysis is subject to limitations, which gives rise to interesting future work.

Considering the effects of charting algorithms, we focus on the most commonly used charting algorithms in the literature on surface convolutions. Other promising charting algorithms, which were not considered in the course of this work, such as Verninas et al. (2026), exist, but yet have to be adopted by surface convolutions. Our charting-method-agnostic surface CNN framework can be a possible starting point for further investigations in this direction.

Regarding the effects of chart radii, we have systematically studied changes in surface CNN performances for a window of pre-selected maximum chart radii. Although empirically returning reasonable results, we cannot claim that the selected chart radii are theoretically valid. This would require knowledge about all local injectivity for all considered shapes, which we do not have. Estimating injectivity radii for a set of arbitrary shapes would alleviate tedious hyperparameter searches, but exceeds the scope of this work.

Lastly, the receptive field of a CNN grows with an increasing number of subsequent convolutions. In our ablation, we used varying template radii in dependence of underlying chart extents which also adjust the receptive field, but only a fixed number of convolutions. We did not study the effects of changing the number of subsequent convolutions for fixed template radii, leaving another possible ablation dimension for future work.

9 Conclusion

To organize the diverse literature on isometry-invariant, spatial surface convolutions, we started this work by providing a survey that structures existing work by how they approach the gauge-ambiguity problem and which charting algorithms they use. Our survey shows that comparisons among surface CNNs jointly change the convolution and the charting algorithm, therefore obscuring the source of performance differences. Additionally, the effect of varying chart radii on the performance of surface CNNs has received little attention in existing literature.

To study the effects of changing charting algorithms and varying chart radii in controlled discretization settings, we reformulated existing surface convolutions into a unified discretization based on which we developed a charting-method agnostic surface CNN framework with an accompanying Python library. In addition to providing practical means for creating controlled discretization settings, we further showcased the advantages of the unified notation by deriving a new, well-performing surface convolution from the comparison of HSNs and GEM-CNNs. Using this framework, we conducted ablations whose results highlight that an appropriate chart radius selection is essential for training well performing surface CNNs. Specifically, we have observed arc performance over chart radii in our FAUST benchmark, indicating the existence of a "sweet spot" for chart radii, most likely a value close to the injectivity radius of the majority surface points. Moreover, our ablations show that the charting algorithm can have a statistically significant effect in local surface prediction tasks.

We conclude that future comparisons of surface CNNs should be conducted in charting-method-controlled environments, e.g., by using the same charting algorithm for all surface CNNs, and chart radii should be treated as critical and tunable hyperparameters when designing surface CNNs.

References

Sourya Basu, Jose Gallego-Posada, Francesco Viganò, James Rowbottom, and Taco Cohen. Equivariant mesh attention networks. *Transactions on Machine Learning Research*, 2022. URL https://openreview.net/forum?id=3IqqJh2Ycy.

Jens Behley, Martin Garbade, Andres Milioto, Jan Quenzel, Sven Behnke, Cyrill Stachniss, and Jurgen Gall. Semantickitti: A dataset for semantic scene understanding of lidar sequences. In *ICCV*, 2019. URL https://openaccess.thecvf.com/content_ICCV_2019/papers/Behley_SemanticKITTI_A_Dataset_for_Semantic_Scene_Understanding_of_LiDAR_Sequences_ICCV_2019_paper.pdf.

Federica Bogo, Javier Romero, Matthew Loper, and Michael J. Black. Faust: Dataset and evaluation for 3d mesh registration. In *CVPR*, 2014. doi: 10.1109/CVPR.2014.491.

Boris Bonev, Thorsten Kurth, Christian Hundt, Jaideep Pathak, Maximilian Baust, Karthik Kashinath, and Anima Anandkumar. Spherical fourier neural operators: Learning stable dynamics on the sphere. In *ICML*, 2023. doi: 10.5555/3618408.3618525.

Boris Bonev, Thorsten Kurth, Ankur Mahesh, Mauro Bisson, Jean Kossaifi, Karthik Kashinath, Anima Anandkumar, William D. Collins, Michael S. Pritchard, and Alexander Keller. Fourcastnet 3: A geometric approach to probabilistic machine-learning weather forecasting at scale, 2025. URL https://arxiv.org/abs/2507.12144.

Davide Boscaini, Jonathan Masci, Emanuele Rodoià, and Michael Bronstein. Learning shape correspondence with anisotropic convolutional neural networks. In *NeurIPS*, 2016. URL https://proceedings.neurips.cc/paper_files/paper/2016/file/228499b55310264a8ea0e27b6e7c6ab6-Paper.pdf.

Giorgos Bouritsas, Sergiy Bokhnyak, Stylianos Ploumpis, Michael Bronstein, and Stefanos Zafeiriou. Neural 3d morphable models: Spiral convolutional networks for 3d shape representation learning and generation. In *ICCV*, 2019. URL https://openaccess.thecvf.com/content_ICCV_2019/papers/Bouritsas_Neural_3D_Morphable_Models_Spiral_Convolutional_Networks_for_3D_Shape_ICCV_2019_paper.pdf.

Glen E. Bredon. *Topology and Geometry*. Graduate Texts in Mathematics. Springer, 1993. ISBN 978-1-4419-3103-0 978-1-4757-6848-0. doi: 10.1007/978-1-4757-6848-0.

Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Velickovic. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *CoRR*, 2021. URL https://arxiv.org/abs/2104.13478.

Taco Cohen, Maurice Weiler, Berkay Kicanaoglu, and Max Welling. Gauge equivariant convolutional networks and the icosahedral cnn. In *ICML*, 2019. URL https://proceedings.mlr.press/v97/cohen19d.html.

Taco S. Cohen and Max Welling. Steerable CNNs. In *ICLR*, 2017. URL https://openreview.net/forum?id=rJQKYt5ll.

Taco S. Cohen, Mario Geiger, Jonas Köhler, and Max Welling. Spherical CNNs. In *ICLR*, 2018. URL https://openreview.net/forum?id=Hkbd5xZRb.

Mauro Comi, Yijiong Lin, Alex Church, Alessio Tonioni, Laurence Aitchison, and Nathan F. Lepora. Touchsdf: A deepsdf approach for 3d shape reconstruction using vision-based tactile sensing. *IEEE Robotics and Automation Letters*, 2024. doi: 10.1109/LRA.2024.3396054.

Keenan Crane, Clarisse Weischedel, and Max Wardetzky. The heat method for distance computation. *Commun. ACM*, 2017. doi: 10.1145/3131280.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *NeurIPS*, 2016. URL https://proceedings.neurips.cc/paper_files/paper/2016/file/04df4d434d481c5bb723be1b6df1ee65-Paper.pdf.

E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1959. doi: 10.1007/BF01386390.

Christer Ericson. *Real-Time Collision Detection*. CRC Press, Inc., USA, 2004. ISBN 1558607323. doi: 10.5555/1121584.

Carlos Esteves, Jean-Jacques Slotine, and Ameesh Makadia. Scaling spherical CNNs. In *ICML*, 2023. URL https://proceedings.mlr.press/v202/esteves23a.html.

Shunwang Gong, Lei Chen, Michael Bronstein, and Stefanos Zafeiriou. Spiralnet++: A fast and highly efficient mesh convolution operator. In *ICCV workshops*, 2019. doi: 10.1109/ICCVW.2019.00509.

Pim De Haan, Maurice Weiler, Taco Cohen, and Max Welling. Gauge equivariant mesh cnns: Anisotropic convolutions on geometric graphs. In *ICLR*, 2021. URL https://openreview.net/forum?id=Jnspzp-oIZE.

Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. Meshcnn: a network with an edge. *ACM Trans. Graph.*, 2019. doi: 10.1145/3306346.3322959.

Shi-Min Hu, Zheng-Ning Liu, Meng-Hao Guo, Jun-Xiong Cai, Jiahui Huang, Tai-Jiang Mu, and Ralph R. Martin. Subdivision-based mesh convolution networks. *ACM Trans. Graph.*, 2022. doi: 10.1145/3506694.

Jingwei Huang, Hao Su, and Leonidas Guibas. Robust watertight manifold surface generation method for shapenet models. *arXiv preprint arXiv:1802.01698*, 2018. URL http://arxiv.org/abs/1802.01698.

R. Kimmel and J. A. Sethian. Computing geodesic paths on manifolds. *PNAS*, 1998. doi: 10.1073/pnas.95.15.8431.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. URL http://arxiv.org/abs/1412.6980.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017. URL https://openreview.net/forum?id=SJU4ayYgl.

Iasonas Kokkinos, Michael M Bronstein, Roee Litman, and Alex M Bronstein. Intrinsic shape context descriptors for deformable shapes. In *CVPR*, 2012. doi: 10.1109/CVPR.2012.6247671.

Ilya Kostrikov, Zhongshi Jiang, Daniele Panozzo, Denis Zorin, and Joan Bruna. Surface networks. In *CVPR*, 2018. doi: 10.1109/CVPR.2018.00269.

Remi Lam, Alvaro Sanchez-Gonzalez, Matthew Willson, Peter Wirnsberger, Meire Fortunato, Ferran Alet, Suman Ravuri, Timo Ewalds, Zach Eaton-Rosen, Weihua Hu, et al. Learning skillful medium-range global weather forecasting. *Science*, 2023. doi: 10.1126/science.adi2336.

Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. Pointcnn: Convolution on x-transformed points. In *NeurIPS*, 2018. URL `https://proceedings.neurips.cc/paper_files/paper/2018/file/f5f8590cd58a54e94377e6ae2eded4d9-Paper.pdf`.

I. Ludwig, D. Tyson, and M. Campen. Halfedgecnn for native and flexible deep learning on triangle meshes. *Computer Graphics Forum*, 2023. doi: 10.1111/cgf.14898.

Jonathan Masci, Davide Boscaini, Michael Bronstein, and Pierre Vandergheynst. Geodesic convolutional neural networks on riemannian manifolds. In *ICCV workshops*, 2015. doi: 10.1109/ICCVW.2015.112.

Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *IROS*, 2015. doi: 10.1109/IROS.2015.7353481.

Eivind Lyche Melvær and Martin Reimers. Geodesic polar coordinates on polygonal meshes. In *Computer Graphics Forum*, 2012. doi: 10.1111/j.1467-8659.2012.03187.x.

Thomas W. Mitchel, Vladimir G. Kim, and Michael Kazhdan. Field convolutions for surface cnns. In *ICCV*, 2021. doi: 10.1109/ICCV48922.2021.00985.

Thomas W Mitchel, Noam Aigerman, Vladimir G Kim, and Michael Kazhdan. Möbius convolutions for spherical cnns. In *ACM SIGGRAPH*, 2022. doi: 10.1145/3528233.3530724.

Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodolà, Jan Svoboda, and Michael M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. doi: 10.1109/CVPR.2017.576.

Stanley Osher and James A Sethian. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. *JCP*, 1988. doi: 10.1016/0021-9991(88)90002-2.

Jung Yeon Park, Lawson L.S. Wong, and Robin Walters. Modeling dynamics over meshes with gauge equivariant nonlinear message passing. In *NeurIPS*, 2023. doi: 10.5555/3666122.3666794.

Adrien Poulenard and Maks Ovsjanikov. Multi-directional geodesic neural networks via equivariant convolution. *ACM Trans. Graph.*, 2018. doi: 10.1145/3272127.3275102.

Charles R Qi, Hao Su, Matthias Nießner, Angela Dai, Mengyuan Yan, and Leonidas J Guibas. Volumetric and multi-view cnns for object classification on 3d data. In *CVPR*, 2016. doi: 10.1109/CVPR.2016.609.

Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *CVPR*, 2017a. doi: 10.1109/CVPR.2017.16.

Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: deep hierarchical feature learning on point sets in a metric space. In *NeurIPS*, 2017b. doi: 10.5555/3295222.3295263.

Samuele Salti, Federico Tombari, and Luigi Di Stefano. Shot: Unique signatures of histograms for surface and texture description. *Computer Vision and Image Understanding*, 2014. doi: 10.1016/j.cviu.2014.04.011.

Stefan C. Schonsheck, Bin Dong, and Rongjie Lai. Parallel transport convolution: A new tool for convolutional neural networks on manifolds. *CoRR*, 2018. URL `http://arxiv.org/abs/1805.07857`.

Jean-Pierre Serre. *Linear representations of finite groups*, volume 42 of *Graduate Texts in Mathematics*. Springer, 1977. ISBN 978-1-4684-9458-7. doi: 10.1007/978-1-4684-9458-7.

J A Sethian. A fast marching level set method for monotonically advancing fronts. *PNAS*, 1996. doi: 10.1073/pnas.93.4.1591.

Nicholas Sharp, Yousuf Soliman, and Keenan Crane. The vector heat method. *ACM Trans. Graph.*, 2019. doi: 10.1145/3243651.

Julian Suk, Pim de Haan, Phillip Lippe, Christoph Brune, and Jelmer M. Wolterink. Mesh neural networks for se(3)-equivariant hemodynamics estimation on the artery wall. *Computers in Biology and Medicine*, 2024. doi: 10.1016/j.compbiomed.2024.108328.

Zhiyu Sun, Ethan Rooke, Jerome Charton, Yusen He, Jia Lu, and Stephen Baek. Zernet: Convolutional neural networks on arbitrary surfaces via zernike local tangent space estimation. *Computer Graphics Forum*, 2020. doi: 10.1111/cgf.14012.

Freyr Sverrisson, Jean Feydy, Bruno E. Correia, and Michael M. Bronstein. Fast end-to-end learning on protein surfaces. In *CVPR*, 2021. doi: 10.1109/CVPR46437.2021.01502.

Maxim Tatarchenko, Jaesik Park, Vladlen Koltun, and Qian-Yi Zhou. Tangent convolutions for dense prediction in 3d. In *CVPR*, 2018. doi: 10.1109/CVPR.2018.00409.

Sathamangalam R Srinivasa Varadhan. On the behavior of the fundamental solution of the heat equation with variable coefficients. *Communications on Pure and Applied Mathematics*, 1967.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Hippolyte Verninas, Caner Korkmaz, Stefanos Zafeiriou, Tolga Birdal, and Simone Foti. Parallelised differentiable straightest geodesics for 3d meshes. In *CVPR*, 2026. URL https://openaccess.thecvf.com/content/CVPR2026/papers/Verninas_Parallelised_Differentiable_Straightest_Geodesics_for_3D_Meshes_CVPR_2026_paper.pdf.

Hui Wang, Yu Guo, and Zhengyou Wang. Face-based cnn on triangular mesh with arbitrary connectivity. *Electronics*, 2022. doi: 10.3390/electronics11152466.

Maurice Weiler and Gabriele Cesa. General e(2)-equivariant steerable cnns. In *NeurIPS*, 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/45d6637b718d0f24a237069fe41b0db4-Paper.pdf.

Maurice Weiler, Mario Geiger, Max Welling, Wouter Boomsma, and Taco S Cohen. 3d steerable cnns: Learning rotationally equivariant features in volumetric data. In *NeurIPS*, 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/488e4104520c6aab692863cc1dba45af-Paper.pdf.

Ruben Wiersma, Elmar Eisemann, and Klaus Hildebrandt. Cnns on surfaces using rotation-equivariant features. *ACM Trans. Graph.*, 2020. doi: 10.1145/3386569.3392437.

Daniel E Worrall, Stephan J Garbin, Daniyar Turmukhambetov, and Gabriel J Brostow. Harmonic networks: Deep translation and rotation equivariance. In *CVPR*, 2017. doi: 10.1109/CVPR.2017.758.

Wenxuan Wu, Zhongang Qi, and Li Fuxin. Pointconv: Deep convolutional networks on 3d point clouds. In *CVPR*, 2019. doi: 10.1109/CVPR.2019.00985.

Zhirong Wu, Shuran Song, Aditya Khosla, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets for 2.5d object recognition and next-best-view prediction. In *CVPR*, 2015. URL https://openaccess.thecvf.com/content_cvpr_2015/papers/Wu_3D_ShapeNets_A_2015_CVPR_paper.pdf.

Taher Yacoub, Camille Depenveiller, Atsushi Tatsuma, Tin Barisin, Eugen Rusakov, Udo Göbel, Yuxu Peng, Shiqiang Deng, Yuki Kagaya, Joon Hong Park, et al. Shrec 2025: Protein surface shape retrieval including electrostatic potential. *Computers & Graphics*, 2025. doi: 10.1016/j.cag.2025.104394.

Li Yi, Hao Su, Xingwen Guo, and Leonidas J Guibas. Syncspeccnn: Synchronized spectral cnn for 3d shape segmentation. In *CVPR*, 2017. doi: 10.1109/CVPR.2017.697.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander Smola. Deep sets. In *NeurIPS*, 2017. URL `https://proceedings.neurips.cc/paper_files/paper/2017/file/f22e4747da1aa27e363d86d40ff442fe-Paper.pdf`.

Yuanzhuo Zhu, Xianjun Li, Chen Niu, Fan Wang, and Jianhua Ma. Geometric deep learning with adaptive full-band spatial diffusion for accurate, efficient, and robust cortical parcellation. *Medical Image Analysis*, 2025. doi: 10.1016/j.media.2025.103492.

A Appendix

A.1 Rewriting harmonic surface networks

Wiersma et al. (2020) extend Harmonic Networks (Worrall et al., 2017) to *Harmonic Surface Networks* (HSNs). Their original formulation of the surface convolution is defined over the neighborhood $\mathcal{N}_k$ of vertex k:

$$\boldsymbol{x}^{l+1}_{k,\Gamma+\gamma} = \sum_{j\in\mathcal{N}_k} \omega_j \boldsymbol{R}(r_{kj})\, e^{i(\gamma\theta_{kj}+\beta)}\, P_{j\to k}\big(\boldsymbol{x}^l_{j,\Gamma}\big) \tag{31}$$

Thereby, ω_j represents pre-computed integration weights, $\boldsymbol{R}(r_{kj})$ describes learnable radial weights, $e^{i(\gamma\theta_{kj}+\beta)}$ the phase with predefined rotation order $\gamma \in \mathbb{N}$ and learnable phase-offset β, θ_{kj} describes the angular polar coordinate of the input and $P_{j\to k}(\boldsymbol{x}^l_{j,\Gamma})$ shows the parallel transport of the input features $\boldsymbol{x}^l_{j,\Gamma}$ with rotation order Γ to the convolution center in mesh vertex k.

In contrast to our implementation, the original formulation of HSNs allowed neighbors to have *arbitrary* polar coordinates. To make this work, Wiersma et al. (2020) defined the radial parameters $\boldsymbol{R}$ as interpolated learnable weight matrices

$$\boldsymbol{R}(r_{kj}) = \sum_{q}^{Q} \mu_q(r_{kj})\, \boldsymbol{W}_q \ , \tag{32}$$

where each $\mu_q(r) \in \mathbb{R}$ represents an interpolation coefficient in dependence of the neighbors radial coordinate r. Prior to training, these are precomputed and stored. Lastly, $\boldsymbol{W}_q \in \mathbb{R}^{d_{\text{out}}\times n_{\text{in}}}$ represents a trainable weight matrix.

In addition to radial weights, HSNs learn phase weights β. The parameter γ denotes a predefined rotation order and Γ is the rotation order of the input features. The value θ_{kj} represents the angular coordinate of the input feature. The term ω_j denotes precomputed integration weights

$$\omega_j = \frac{1}{3}\sum_{ab} a_{jab} \ , \tag{33}$$

where a_{jab} represent triangle areas adjacent to vertex j. Wiersma et al. (2020) also precompute and store the integration weights.

The vector $\boldsymbol{x}^l_{j,\Gamma} \in \mathbb{C}^n$ denotes complex features returned by layer l of neighbor j following rotation order Γ, and

$$P_{j\to k}\big(\boldsymbol{x}^l_{j,\Gamma}\big) = \begin{pmatrix} a^{(1)}_{kj}\, e^{i\Gamma\Phi^{(1)}_{kj}} \\ \vdots \\ a^{(d_{\text{in}})}_{kj}\, e^{i\Gamma\Phi^{(d_{\text{in}})}_{kj}} \end{pmatrix} \ , \tag{34}$$

represents the parallel transport of the complex feature vector $\boldsymbol{x}^l_{j,\Gamma}$ from neighbor j to convolution center k.

For notational brevity, we leave away the sub- and superscripts for rotation orders and layers. This allows us to write:

$$\boldsymbol{x}_k = \sum_{j \in \mathcal{N}_k} \omega_j \boldsymbol{R}(r_{kj})\, e^{i(\gamma\theta_{kj}+\beta)}\, P_{j\to k}\big(\boldsymbol{x}_j\big)\ . \tag{35}$$

Our framework describes neighborhoods $\mathcal{N}_k$ with the set of template vertices $\mathbb{T}$. We thus replace neighbor index j with tuples of radial- and angular indices (r, a):

$$\boldsymbol{x}_k = \sum_{r=1}^{N_\tau}\sum_{a=1}^{N_\theta} \omega_{ra} \boldsymbol{R}(r_{kr})\, e^{i(m\theta_{ka}+\beta)}\, P_{(r,a)\to k}\big(\boldsymbol{x}_{(r,a)}\big)\ . \tag{36}$$

Using the template vertices instead of neighbors with arbitrary polar coordinates removes the need to interpolate radial weights since neighbors now always have known, fixed radial distances. That is, we only need to learn one weight matrix $\boldsymbol{R}_r \in \mathbb{R}^{d_{\text{out}}\times d_{\text{in}}}$ usable in any vertex k, thereby reducing computation complexity and the amount of required weights. Analogously, the relative angular coordinates of neighbors remain the same for any center vertex k, removing the dependence of θ_{ka} on k:

$$\boldsymbol{x}_k = \sum_{r=1}^{N_\tau}\sum_{a=1}^{N_\theta} \omega_{ra} \boldsymbol{R}_r\, e^{i(\gamma\theta_a+\beta)}\ +_{(r,a)\to k}\big(\boldsymbol{x}_{(r,a)}\big) \tag{37}$$

$$= \sum_{r=1}^{N_\tau}\sum_{a=1}^{N_\theta} \omega_{ra}\, \boldsymbol{R}_r\, e^{i(\gamma\theta_a+\beta)} \begin{pmatrix} a^{(1)}_{kra}\, e^{i\Gamma\Phi^{(1)}_{kra}} \\ \vdots \\ a^{(d_{\text{in}})}_{kra}\, e^{i\Gamma\Phi^{(d_{\text{in}})}_{kra}} \end{pmatrix} \tag{38}$$

$$= \sum_{r=1}^{N_\tau}\sum_{a=1}^{N_\theta} \omega_{ra} \underbrace{\begin{pmatrix} w^{(r)}_{11} & \dots & w^{(r)}_{1d} \\ \vdots & \ddots & \vdots \\ w^{(r)}_{d_{\text{out}}1} & \dots & w^{(r)}_{d_{\text{out}}d_{\text{in}}} \end{pmatrix}}_{=\boldsymbol{R}_r} \begin{pmatrix} e^{i(m\theta_a+\beta)}\, a^{(1)}_{kra}\, e^{i\Gamma\Phi^{(1)}_{kra}} \\ \vdots \\ e^{i(m\theta_a+\beta)}\, a^{(d_{\text{in}})}_{kra}\, e^{i\Gamma\Phi^{(d_{\text{in}})}_{kra}} \end{pmatrix}\ . \tag{39}$$

For the implementation of the harmonic surface convolution we are particularly interested in how we compute the m-th entry of the resulting embedding $\boldsymbol{x}_k$:

$$\boldsymbol{x}_{km} = \sum_{r=1}^{N_\tau}\sum_{a=1}^{N_\theta}\sum_{n=1}^{d_{\text{in}}} \omega_{ra}\, w^{(r)}_{mn}\, e^{i(\gamma\theta_a+\beta)}\, a^{(n)}_{kra}\, e^{i\Gamma\Phi^{(n)}_{kra}} \tag{40}$$

$$= \sum_{r=1}^{N_\tau}\sum_{a=1}^{N_\theta} \omega_{ra} \sum_{n=1}^{d_{\text{in}}} w^{(r)}_{mn} e^{i(\gamma\theta_a+\beta)}\, a^{(n)}_{kra}\, e^{i\Gamma\Phi^{(n)}_{kra}}\ . \tag{41}$$

We set the integration weights $\omega_{ra} = 1$ to alleviate from preprocessing complexity and to improve comparability to other surface convolution types that did not use integration weights. A detailed study of the impact of ω on the training performance of surface convolutions is left for future work.

The term $w^{(r)}_{mn}\, e^{i(\gamma\theta_a+\beta)} \cdot a^{(n)}_{kra}\, e^{i\Gamma\Phi^{(n)}_{kra}}$ represents multiplication between two complex numbers. We can rewrite complex multiplication in matrix-vector notation:

$$\boldsymbol{x}_{km} = \sum_{r=1}^{N_\tau}\sum_{a=1}^{N_\theta}\sum_{n=1}^{d_{\text{in}}} w^{(r)}_{mn} \begin{pmatrix} \cos(\gamma\theta_a+\beta) & -\sin(\gamma\theta_a+\beta) \\ \sin(\gamma\theta_a+\beta) & \cos(\gamma\theta_a+\beta) \end{pmatrix} \begin{pmatrix} a^{(n)}_{kra}\cos(\Gamma\,\Phi^{(n)}_{kra}) \\ a^{(n)}_{kra}\sin(\Gamma\,\Phi^{(n)}_{kra}) \end{pmatrix} \tag{42}$$

$$= \sum_{r=1}^{N_\tau}\sum_{a=1}^{N_\theta}\sum_{n=1}^{d_{\text{in}}} w^{(r)}_{mn} \begin{pmatrix} \cos(\gamma\theta_a+\beta) & -\sin(\gamma\theta_a+\beta) \\ \sin(\gamma\theta_a+\beta) & \cos(\gamma\theta_a+\beta) \end{pmatrix} \begin{pmatrix} \text{Re}\left(\widetilde{x}^{(k,r,a)}_n\right) \\ \text{Im}\left(\widetilde{x}^{(k,r,a)}_n\right) \end{pmatrix}\ . \tag{43}$$

Eventually, we can learn different β per output dimension q while maintaining gauge-equivariance for output dimension q:

$$\boldsymbol{x}_{km} = \sum_{r=1}^{N_\tau} \sum_{a=1}^{N_\theta} \sum_{n=1}^{d_{\text{in}}} w_{mn}^{(r)} \begin{pmatrix} \cos(\gamma\theta_a + \beta_q) & -\sin(\gamma\theta_a + \beta_q) \\ \sin(\gamma\theta_a + \beta_q) & \cos(\gamma\theta_a + \beta_q) \end{pmatrix} \begin{pmatrix} \text{Re}\left(\widetilde{x}_n^{(k,r,a)}\right) \\ \text{Im}\left(\widetilde{x}_n^{(k,r,a)}\right) \end{pmatrix} . \tag{44}$$

The context typically clarifies the fact that we are interested in the m-th embedding feature for the feature vector associated to vertex k. Hence, we drop the index k from the equation for notational brevity:

$$\boldsymbol{x}_{m} = \sum_{r=1}^{N_\tau} \sum_{a=1}^{N_\theta} \sum_{n=1}^{d_{\text{in}}} w_{mn}^{(r)} \begin{pmatrix} \cos(\gamma\theta_a + \beta_q) & -\sin(\gamma\theta_a + \beta_q) \\ \sin(\gamma\theta_a + \beta_q) & \cos(\gamma\theta_a + \beta_q) \end{pmatrix} \begin{pmatrix} \text{Re}\left(\widetilde{x}_n^{(r,a)}\right) \\ \text{Im}\left(\widetilde{x}_n^{(r,a)}\right) \end{pmatrix} . \tag{45}$$

This equation represents the harmonic surface convolution shown in equation 16.

A.2 GEM-CNN basis kernels implement complex valued functions

In Section 6.3, we represent individual input features for GEM-CNNs as complex values. Core to this notation idea is the following observation: because ρ_γ is constructed from irreducible representations (irreps) of SO(2), those individual features that transform according to these irreps can be considered two-dimensional quantities which we identify with complex values. The only exception are rotation-invariant features (type ρ_0), whose corresponding irrep is the real-valued identity function. However, we can "lift" this identity and its associated one-dimensional, real-valued feature into two dimensions by appending zeros (cf. Table 11), allowing us to treat all input features as two-dimensional quantities in our notation.

The basis kernels of GEM-CNNs used in equation 18 define functions between feature types. To clarify how complex-valued features are used, we derive equivalent function definitions for each basis kernel of a map type, whose domains and codomains characterize how complex numbers are processed in terms of lifts into the complex domain, projections onto the real axis or transformations within the complex domain. To this end, we identify real and complex values with the following vectors:

$$\underbrace{r\cos\alpha}_{\in\mathbb{R}} \mathrel{\widehat{=}} r \begin{pmatrix} \cos\alpha \\ 0 \end{pmatrix}, \ \underbrace{r\left(\cos\alpha + i\sin\alpha\right)}_{\in\mathbb{C}} \mathrel{\widehat{=}} r \begin{pmatrix} \cos\alpha \\ \sin\alpha \end{pmatrix} , \tag{46}$$

and assume $r = 1$ to reduce notational clutter.

Map type $\boldsymbol{\rho_0 \to \rho_0}$. This map type considers one basis kernel, expecting a rotation invariant feature of type ρ_0 as input and returning the same feature as output:

$$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \cos\alpha \\ 0 \end{pmatrix} = \begin{pmatrix} \cos\alpha \\ 0 \end{pmatrix} \tag{47}$$

This basis kernel thus implements a function $f : \mathbb{R} \to \mathbb{R}$ with $r\cos\alpha \mapsto r\cos\alpha$, corresponding to the identity function for rotation-invariant, real-valued features.

Map type $\boldsymbol{\rho_\gamma \to \rho_0}$. This map type considers two basis kernels, expects rotation-equivariant features of type ρ_γ as input, and returns rotation-invariant features:

$$\begin{pmatrix} \cos(\gamma\theta) & \sin(\gamma\theta) \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \cos\alpha \\ \sin\alpha \end{pmatrix} = \begin{pmatrix} \cos(\gamma\theta)\,\cos\alpha + \sin(\gamma\theta)\,\sin\alpha \\ 0 \end{pmatrix} = \begin{pmatrix} \cos(\gamma\theta - \alpha) \\ 0 \end{pmatrix} \tag{48}$$

$$\begin{pmatrix} \sin(\gamma\theta) & -\cos(\gamma\theta) \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \cos\alpha \\ \sin\alpha \end{pmatrix} = \begin{pmatrix} \sin(\gamma\theta)\,\cos\alpha - \cos(\gamma\theta)\,\sin\alpha \\ 0 \end{pmatrix} = \begin{pmatrix} \sin(\gamma\theta - \alpha) \\ 0 \end{pmatrix} \tag{49}$$

Table 11: Zero padded basis kernels. Like in Haan et al. (2021), $c_\pm = \cos((\gamma' \pm \gamma)\theta)$ and $s_\pm = \sin((\gamma' \pm \gamma)\theta)$.

neigh.	original (Haan et al., 2021)				zero-padded	
$\rho_0 \to \rho_0$	(1)	-	-	-	$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$	-
$\rho_\gamma \to \rho_0$	$\begin{pmatrix} \cos\gamma\theta & \sin\gamma\theta \end{pmatrix}$	$\begin{pmatrix} \sin\gamma\theta & -\cos\gamma\theta \end{pmatrix}$	-	-	$\begin{pmatrix} \cos\gamma\theta & \sin\gamma\theta \\ 0 & 0 \end{pmatrix}$	$\begin{pmatrix} \sin\gamma\theta & -\cos\gamma\theta \\ 0 & 0 \end{pmatrix}$
$\rho_0 \to \rho_{\gamma'}$	$\begin{pmatrix} \cos\gamma'\theta \\ \sin\gamma'\theta \end{pmatrix}$	$\begin{pmatrix} \sin\gamma'\theta \\ -\cos\gamma'\theta \end{pmatrix}$	-	-	$\begin{pmatrix} \cos\gamma'\theta & 0 \\ \sin\gamma'\theta & 0 \end{pmatrix}$	$\begin{pmatrix} \sin\gamma'\theta & 0 \\ -\cos\gamma'\theta & 0 \end{pmatrix}$
$\rho_\gamma \to \rho_{\gamma'}$	$\begin{pmatrix} c_- & -s_- \\ s_- & c_- \end{pmatrix}$	$\begin{pmatrix} s_- & c_- \\ -c_- & s_- \end{pmatrix}$	$\begin{pmatrix} c_+ & s_+ \\ s_+ & -c_+ \end{pmatrix}$	$\begin{pmatrix} -s_+ & c_+ \\ c_+ & s_+ \end{pmatrix}$	equal to original	
self						
$\rho_0 \to \rho_0$	(1)	-	-	-	$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$	-
$\rho_\gamma \to \rho_\gamma$	$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$	-	-	equal to original	
$\rho_\gamma \to \rho_{\gamma'}$	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$	-	-	-	equal to original	

The basis kernels thus implement projections $f_1, f_2 : \mathbb{C} \to \mathbb{R}$ onto the real axis:

$$\begin{pmatrix} \cos(\gamma\theta) & \sin(\gamma\theta) \\ 0 & 0 \end{pmatrix} \text{ corresponds to: } r\,(\cos\alpha + i\sin\alpha) \mapsto r\cos(\gamma\theta - \alpha) \tag{50}$$

$$\begin{pmatrix} \sin(\gamma\theta) & -\cos(\gamma\theta) \\ 0 & 0 \end{pmatrix} \text{ corresponds to: } r\,(\cos\alpha + i\sin\alpha) \mapsto r\sin(\gamma\theta - \alpha) \tag{51}$$

Map type $\boldsymbol{\rho_0 \to \rho_{\gamma'}}$. This map type also considers two basis kernels, but expects rotation-invariant features as input and returns rotation-equivariant features:

$$\begin{pmatrix} \cos(\gamma'\theta) & 0 \\ \sin(\gamma'\theta) & 0 \end{pmatrix} \begin{pmatrix} \cos\alpha \\ 0 \end{pmatrix} = \cos\alpha \begin{pmatrix} \cos(\gamma'\theta) \\ \sin(\gamma'\theta) \end{pmatrix} \tag{52}$$

$$\begin{pmatrix} \sin(\gamma'\theta) & 0 \\ -\cos(\gamma'\theta) & 0 \end{pmatrix} \begin{pmatrix} \cos\alpha \\ 0 \end{pmatrix} = \cos\alpha \begin{pmatrix} \sin(\gamma'\theta) \\ -\cos(\gamma'\theta) \end{pmatrix} \tag{53}$$

The basis kernels thus lift real values into the complex plane via functions $f_3, f_4 : \mathbb{R} \to \mathbb{C}$:

$$\begin{pmatrix} \cos(\gamma'\theta) & 0 \\ \sin(\gamma'\theta) & 0 \end{pmatrix} \text{ corresponds to: } r\,\cos\alpha \mapsto r\,\cos\alpha\, e^{i\gamma'\theta} \tag{54}$$

$$\begin{pmatrix} \sin(\gamma'\theta) & 0 \\ -\cos(\gamma'\theta) & 0 \end{pmatrix} \text{ corresponds to: } r\,\cos\alpha \mapsto -i\,r\,\cos\alpha\, e^{i\gamma'\theta} \tag{55}$$

where we use Euler's formula for $\cos(\gamma'\theta) + i\sin(\gamma'\theta) = e^{i\gamma'\theta}$.

Map type $\boldsymbol{\rho_\gamma \to \rho_{\gamma'}}$. This map type considers four different basis-kernels. To highlight their differences, we factorize each basis-kernel into two matrices:

$$\begin{pmatrix} \cos((\gamma'-\gamma)\theta+\beta) & -\sin((\gamma'-\gamma)\theta+\beta) \\ \sin((\gamma'-\gamma)\theta+\beta) & \cos((\gamma'-\gamma)\theta+\beta) \end{pmatrix} = \begin{pmatrix} \cos((\gamma'-\gamma)\theta+\beta) & -\sin((\gamma'-\gamma)\theta+\beta) \\ \sin((\gamma'-\gamma)\theta+\beta) & \cos((\gamma'-\gamma)\theta+\beta) \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$\begin{pmatrix} \sin((\gamma'-\gamma)\theta+\beta) & \cos((\gamma'-\gamma)\theta+\beta) \\ -\cos((\gamma'-\gamma)\theta+\beta) & \sin((\gamma'-\gamma)\theta+\beta) \end{pmatrix} = \begin{pmatrix} \cos((\gamma'-\gamma)\theta+\beta) & -\sin((\gamma'-\gamma)\theta+\beta) \\ \sin((\gamma'-\gamma)\theta+\beta) & \cos((\gamma'-\gamma)\theta+\beta) \end{pmatrix} \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$$

$$\begin{pmatrix} \cos((\gamma'+\gamma)\theta+\beta) & \sin((\gamma'+\gamma)\theta+\beta) \\ \sin((\gamma'+\gamma)\theta+\beta) & -\cos((\gamma'+\gamma)\theta+\beta) \end{pmatrix} = \begin{pmatrix} \cos((\gamma'+\gamma)\theta+\beta) & -\sin((\gamma'+\gamma)\theta+\beta) \\ \sin((\gamma'+\gamma)\theta+\beta) & \cos((\gamma'+\gamma)\theta+\beta) \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

$$\begin{pmatrix} -\sin((\gamma'+\gamma)\theta+\beta) & \cos((\gamma'+\gamma)\theta+\beta) \\ \cos((\gamma'+\gamma)\theta+\beta) & \sin((\gamma'+\gamma)\theta+\beta) \end{pmatrix} = \begin{pmatrix} \cos((\gamma'+\gamma)\theta+\beta) & -\sin((\gamma'+\gamma)\theta+\beta) \\ \sin((\gamma'+\gamma)\theta+\beta) & \cos((\gamma'+\gamma)\theta+\beta) \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

We can see that except the first kernel, which only rotates the input feature, any other kernel transforms the input before rotation. The transformation becomes clear once we multiply a complex number in vector notation with the right matrices from the matrix factorizations:

$$\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \begin{pmatrix} \cos(x) \\ \sin(x) \end{pmatrix} = \begin{pmatrix} \sin(x) \\ -\cos(x) \end{pmatrix} \mathrel{\widehat{=}} \sin(x) - i\cos(x) = -i(\cos(x) + i\sin(x)) = -ie^{ix} \text{ (multiply } -i)$$

$$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \begin{pmatrix} \cos(x) \\ \sin(x) \end{pmatrix} = \begin{pmatrix} \cos(x) \\ -\sin(x) \end{pmatrix} \mathrel{\widehat{=}} \cos(x) - i\sin(x) = e^{-ix} \text{ (conjugate input)}$$

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \cos(x) \\ \sin(x) \end{pmatrix} = \begin{pmatrix} \sin(x) \\ \cos(x) \end{pmatrix} \mathrel{\widehat{=}} \sin(x) + i\cos(x) = i(\cos(x) - i\sin(x)) = ie^{-ix} \text{ (conjugate \& multiply } i)$$

Rotation matrices are equivalent to the function that multiplies a complex value onto a given complex value:

$$\begin{pmatrix} \cos(x) & -\sin(x) \\ \sin(x) & \cos(x) \end{pmatrix} \text{ corresponds to: } z \mapsto e^{ix} z \ .$$

Taking everything together, the basis kernels of this map type give rise to the following complex-valued functions for $\mathbb{C} \to \mathbb{C}$:

$$\begin{pmatrix} \cos((\gamma'-\gamma)\theta+\beta) & -\sin((\gamma'-\gamma)\theta+\beta) \\ \sin((\gamma'-\gamma)\theta+\beta) & \cos((\gamma'-\gamma)\theta+\beta) \end{pmatrix} \text{ corresponds to: } z \mapsto e^{i((\gamma'-\gamma)\theta+\beta)}\, z \tag{56}$$

$$\begin{pmatrix} \sin((\gamma'-\gamma)\theta+\beta) & \cos((\gamma'-\gamma)\theta+\beta) \\ -\cos((\gamma'-\gamma)\theta+\beta) & \sin((\gamma'-\gamma)\theta+\beta) \end{pmatrix} \text{ corresponds to: } z \mapsto -ie^{i((\gamma'-\gamma)\theta+\beta)}\, z \tag{57}$$

$$\begin{pmatrix} \cos((\gamma'+\gamma)\theta+\beta) & \sin((\gamma'+\gamma)\theta+\beta) \\ \sin((\gamma'+\gamma)\theta+\beta) & -\cos((\gamma'+\gamma)\theta+\beta) \end{pmatrix} \text{ corresponds to: } z \mapsto e^{i((\gamma'+\gamma)\theta+\beta)}\, \overline{z} \tag{58}$$

$$\begin{pmatrix} -\sin((\gamma'+\gamma)\theta+\beta) & \cos((\gamma'+\gamma)\theta+\beta) \\ \cos((\gamma'+\gamma)\theta+\beta) & \sin((\gamma'+\gamma)\theta+\beta) \end{pmatrix} \text{ corresponds to: } z \mapsto ie^{i((\gamma'+\gamma)\theta+\beta)}\, \overline{z} \tag{59}$$

A.3 Learning a constant phase shift does not break gauge-equivariance

In this section, we show that extending the basis kernels of GEM-CNNs (Haan et al., 2021) by learnable phase offsets does not break their gauge-equivariance property. To this end, it suffices to show that the linear combination of basis kernels satisfies the rotation-steerable kernel constraint (Weiler et al., 2018):

$$t(g_{\boldsymbol{u}}^{-1}\, \boldsymbol{v}) = \rho_{\text{out}}(g_{\boldsymbol{u}}^{-1})\ t(\boldsymbol{v})\ \rho_{\text{in}}(g_{\boldsymbol{u}}) \ , \tag{60}$$

where ρ_{in} and ρ_{out} are the representations of the input and output features, given by (Haan et al., 2021):

$$\rho_0(g) = 1,\ \rho_\gamma(g) = \begin{pmatrix} \cos(\gamma g) & -\sin(\gamma g) \\ \sin(\gamma g) & \cos(\gamma g) \end{pmatrix} \ . \tag{61}$$

GEM-CNNs employ four map types: $\rho_0 \to \rho_0$, $\rho_\gamma \to \rho_0$, $\rho_0 \to \rho_{\gamma'}$ and $\rho_\gamma \to \rho_{\gamma'}$, each of which is associated to a set of basis kernels which are linearly combined during the convolution. A linear combination of basis kernels decomposes the constraint of the result onto its addends:

$$\begin{aligned} t(g_{\boldsymbol{u}}^{-1}\,\boldsymbol{v}) &= \rho_{\text{out}}(g_{\boldsymbol{u}}^{-1})\;t(\boldsymbol{v})\;\rho_{\text{in}}(g_{\boldsymbol{u}}) \\ &= \rho_{\text{out}}(g_{\boldsymbol{u}}^{-1})\;\left(\sum_s t^s(\boldsymbol{v})\right)\;\rho_{\text{in}}(g_{\boldsymbol{u}}) \\ &= \sum_s \left(\rho_{\text{out}}(g_{\boldsymbol{u}}^{-1})\;t^s(\boldsymbol{v})\;\rho_{\text{in}}(g_{\boldsymbol{u}})\right) = \sum_s t^s(g_{\boldsymbol{u}}^{-1}\boldsymbol{v})\;. \end{aligned} \tag{62}$$

Differently phrased, if each basis kernel satisfies the kernel constraint, then their linear combination does too. Thus, we only need to show that each basis kernel satisfies the kernel constraint separately in each map type to prove that the equivariance constraint is preserved.

In our context we have $g = \Delta$ and $\boldsymbol{v} = \theta$, where β is a learnable constant phase offset and θ the angular coordinate of a template vertex. Consequently, the kernel constraint becomes:

$$t(g_{\boldsymbol{u}}^{-1}\,\boldsymbol{v}) = \rho_{\text{out}}(g_{\boldsymbol{u}}^{-1})\;t(\boldsymbol{v})\;\rho_{\text{in}}(g_{\boldsymbol{u}}) \Leftrightarrow t(\theta - \Delta) = \rho_{\text{out}}(-\Delta)\;t(\theta)\;\rho_{\text{in}}(\Delta)\;.$$

Proof. **Map types $\boldsymbol{\rho_0 \to \rho_0}$ and $\boldsymbol{\rho_\gamma \to \rho_0}$.** Both map types share the fact that their output features are of type $\rho_{\text{out}}(-\Delta) = 1$. That is, we need to show that the kernel factorizes as $t(\theta - \Delta) = t(\theta)\rho_{\text{in}}(\Delta)$. However, map type $\rho_0 \to \rho_0$ additionally constrains the input to be of type ρ_0 and only considers $t(\theta) = 1$. Consequently, the kernel constraint simplifies to $t(\theta - \Delta) = t(\theta)\rho_{\text{in}}(\Delta) = 1$ which is true. On the other hand, for the first basis kernel of map type $\rho_\gamma \to \rho_0$ we have:

$$\begin{aligned} t(\theta - \Delta) &= \big(\cos(\gamma(\theta - \Delta) + \beta) \quad \sin(\gamma(\theta - \Delta) + \beta)\big) \\ &= \big(\cos(\gamma\theta - \gamma\Delta + \beta) \quad \sin(\gamma\theta - \gamma\Delta + \beta)\big) \\ &= \big(\cos(\gamma\theta + \beta)\cos(\gamma\Delta) - \sin(\gamma\theta + \beta)\sin(\gamma\Delta) \quad \sin(\gamma\theta + \beta)\cos(\gamma\Delta) - \cos(\gamma\theta + \beta)\sin(\gamma\Delta)\big) \\ &= \big(\cos(\gamma\theta + \beta) \quad \sin(\gamma\theta + \beta)\big)\begin{pmatrix}\cos(\gamma\Delta) & -\sin(\gamma\Delta) \\ \sin(\gamma\Delta) & \cos(\gamma\Delta)\end{pmatrix} \\ &= t(\theta)\,\rho_{\text{in}}(\Delta)\;. \end{aligned}$$

For the second basis-kernel of map type $\rho_\gamma \to \rho_0$ we have:

$$\begin{aligned} t(\theta - \Delta) &= \big(\sin(\gamma(\theta - \Delta) + \beta) \quad -\cos(\gamma(\theta - \Delta) + \beta)\big) \\ &= \big(\sin(\gamma\theta - \gamma\Delta + \beta) \quad -\cos(\gamma\theta - \gamma\Delta + \beta)\big) \\ &= \big(\sin(\gamma\theta + \beta)\cos(\gamma\Delta) - \cos(\gamma\theta + \beta)\sin(\gamma\Delta) \quad -\cos(\gamma\theta + \beta)\cos(\gamma\Delta) - \sin(\gamma\theta + \beta)\sin(\gamma\Delta)\big) \\ &= \big(\sin(\gamma\theta + \beta) \quad -\cos(\gamma\theta + \beta)\big)\begin{pmatrix}\cos(\gamma\Delta) & -\sin(\gamma\Delta) \\ \sin(\gamma\Delta) & \cos(\gamma\Delta)\end{pmatrix} \\ &= t(\theta)\,\rho_{\text{in}}(\Delta)\;. \end{aligned}$$

Overall, both basis-kernels for map type $\rho_\gamma \to \rho_0$ satisfy the kernel constraint $t(\theta - \Delta) = t(\theta)\rho_{\text{in}}(\Delta)$ and therefore ensure gauge-equivariance.

Map type $\boldsymbol{\rho_0 \to \rho_{\gamma'}}$. This map type expects input features of type ρ_0 and returns output features of type $\rho_{\gamma'}$. Thus, the basis kernels for this map type need to satisfy $t(\theta - \Delta) = \rho_{\text{out}}(-\Delta)t(\theta)$. For the first

basis-kernel of map type $\rho_0 \to \rho_{\gamma'}$ we get:

$$\begin{aligned}
t(\theta - \Delta) &= \begin{pmatrix} \cos(\gamma'(\theta - \Delta) + \beta) \\ \sin(\gamma'(\theta - \Delta) + \beta) \end{pmatrix} = \begin{pmatrix} \cos(\gamma'\theta - \gamma'\Delta + \beta) \\ \sin(\gamma'\theta - \gamma'\Delta + \beta) \end{pmatrix} \\
&= \begin{pmatrix} \cos(\gamma'\theta + \beta)\cos(\gamma'\Delta) - \sin(\gamma'\theta + \beta)\sin(\gamma'\Delta) \\ \sin(\gamma'\theta + \beta)\cos(\gamma'\Delta) - \cos(\gamma'\theta + \beta)\sin(\gamma'\Delta) \end{pmatrix} \\
&= \begin{pmatrix} \cos(\gamma'\Delta) & -\sin(\gamma'\Delta) \\ \sin(\gamma'\Delta) & \cos(\gamma'\Delta) \end{pmatrix}^T \begin{pmatrix} \cos(\gamma'\theta + \beta) \\ \sin(\gamma'\theta + \beta) \end{pmatrix} \\
&= \rho_{\text{out}}(-\Delta) t(\theta) \ .
\end{aligned}$$

For the second basis-kernel of map type $\rho_0 \to \rho_{\gamma'}$ we get:

$$\begin{aligned}
t(\theta - \Delta) &= \begin{pmatrix} \sin(\gamma'(\theta - \Delta) + \beta) \\ -\cos(\gamma'(\theta - \Delta) + \beta) \end{pmatrix} = \begin{pmatrix} \sin(\gamma'\theta - \gamma'\Delta + \beta) \\ -\cos(\gamma'\theta - \gamma'\Delta + \beta) \end{pmatrix} \\
&= \begin{pmatrix} \sin(\gamma'\theta + \beta)\cos(\gamma'\Delta) - \cos(\gamma'\theta + \beta)\sin(\gamma'\Delta) \\ -\cos(\gamma'\theta + \beta)\cos(\gamma'\Delta) - \sin(\gamma'\theta + \beta)\sin(\gamma'\Delta) \end{pmatrix} \\
&= \begin{pmatrix} \cos(\gamma'\Delta) & -\sin(\gamma'\Delta) \\ \sin(\gamma'\Delta) & \cos(\gamma'\Delta) \end{pmatrix}^T \begin{pmatrix} \sin(\gamma'\theta + \beta) \\ -\cos(\gamma'\theta + \beta) \end{pmatrix} \\
&= \rho_{\text{out}}(-\Delta) t(\theta) \ .
\end{aligned}$$

Hence, both basis-kernels satisfy the kernel constraint $t(\theta - \Delta) = \rho_{\text{out}}(-\Delta)t(\theta)$.

Map type $\boldsymbol{\rho_\gamma \to \rho_{\gamma'}}$. The map type $\rho_\gamma \to \rho_{\gamma'}$ expects input features of type ρ_γ and returns output features of type $\rho_{\gamma'}$. The basis-kernels for this map type hence need to satisfy $t(\theta - \Delta) = \rho_{\text{out}}(-\Delta)t(\theta)\rho_{\text{in}}(\Delta)$. In this case, we use the complex notation from Appendix A.2.

Basis-kernels that use $e^{i(\gamma'+\gamma)\theta}$ always conjugate the input. We denote this in the surface convolution as:

$$(s * t)(\boldsymbol{u}; \omega_{\boldsymbol{u}}) = \int_{B_R(\mathbf{0})} t(\boldsymbol{v}) \ \overline{\rho(g_{\boldsymbol{q}_v \to \boldsymbol{u}}) \ [s \circ \exp_{\boldsymbol{u}} \circ \omega_{\boldsymbol{u}}](\boldsymbol{v})} \ d\boldsymbol{v}.$$

Furthermore, considering the surface convolution under gauge-transformation shows:

$$\begin{aligned}
(s * t)(\boldsymbol{u}; g) &= \int_{B_R(\mathbf{0})} t(g_{\boldsymbol{u}}^{-1}\boldsymbol{v}) \ \overline{\rho_{\text{in}}(g_{\boldsymbol{u}}^{-1})\rho_{\text{in}}(g_{\boldsymbol{q}_v \to \boldsymbol{u}}) \ [s \circ \exp_{\boldsymbol{u}} \circ \omega_{\boldsymbol{u}}](\boldsymbol{v})} \ d\boldsymbol{v} \\
&= \int_{B_R(\mathbf{0})} \rho_{\text{out}}(g_{\boldsymbol{u}}^{-1}\boldsymbol{v}) \ t(\boldsymbol{v}) \ \overline{\rho_{\text{in}}(g_{\boldsymbol{u}}) \ \rho_{\text{in}}(g_{\boldsymbol{u}}^{-1}) \ \rho_{\text{in}}(g_{\boldsymbol{q}_v \to \boldsymbol{u}}) \ [s \circ \exp_{\boldsymbol{u}} \circ \omega_{\boldsymbol{u}}](\boldsymbol{v})} \ d\boldsymbol{v} \ .
\end{aligned}$$

Hence, kernels that use $e^{i((\gamma'+\gamma)\theta+\beta)}$ have to satisfy $t(\theta - \Delta) = \rho_{\text{out}}(-\Delta)\, t(\theta)\, \overline{\rho_{\text{in}}(\Delta)}$. For the first basis-kernel we have:

$$\begin{aligned}
t(\theta - \Delta) &= e^{i((\gamma'-\gamma)(\theta-\Delta)+\beta)} = e^{i\left((\gamma'-\gamma)\theta - (\gamma'-\gamma)\Delta + \beta\right)} = e^{i\left((\gamma'-\gamma)\theta + \gamma\Delta - \gamma'\Delta + \beta\right)} \\
&= e^{-i\gamma'\Delta}\, e^{i((\gamma'-\gamma)\theta+\beta)}\, e^{i\gamma\Delta} = \rho_{\text{out}}(-\Delta)\, t(\theta)\, \rho_{\text{in}}(\Delta) \ .
\end{aligned}$$

For the second basis-kernel we have:

$$\begin{aligned}
t(\theta - \Delta) &= -ie^{i((\gamma'-\gamma)(\theta-\Delta)+\beta)} = -i\left(e^{-i\gamma'\Delta}\, e^{i((\gamma'-\gamma)\theta+\beta)}\, e^{i\gamma\Delta}\right) = e^{-i\gamma'\Delta}\, \left(-ie^{i((\gamma'-\gamma)\theta+\beta)}\right)\, e^{i\gamma\Delta} \\
&= \rho_{\text{out}}(-\Delta)\, t(\theta)\, \rho_{\text{in}}(\Delta) \ .
\end{aligned}$$

For the third basis-kernel we have:

$$\begin{aligned}
t(\theta - \Delta) &= e^{i((\gamma'+\gamma)(\theta-\Delta)+\beta)} = e^{i((\gamma'+\gamma)\theta - (\gamma'+\gamma)\Delta + \beta)} = e^{i((\gamma'+\gamma)\theta - \gamma'\Delta - \gamma\Delta + \beta)} \\
&= e^{-i\gamma'\Delta} e^{i((\gamma'+\gamma)\theta+\beta)} e^{-i\gamma\Delta} = \rho_{\text{out}}(-\Delta)\, t(\theta)\, \overline{\rho_{\text{in}}(\Delta)} \ .
\end{aligned}$$

For the fourth basis-kernel we have:

$$\begin{aligned} t(\theta - \Delta) &= i\, e^{i((\gamma'+\gamma)(\theta-\Delta)+\beta)} = i\left(e^{-i\gamma'\Delta} e^{i((\gamma'+\gamma)\theta+\beta)} e^{-i\gamma\Delta}\right) \\ &= e^{-i\gamma'\Delta}\left(i\, e^{i((\gamma'+\gamma)\theta+\beta)}\right) e^{-i\gamma\Delta} = \rho_{\text{out}}(-\Delta)\, t(\theta)\, \overline{\rho_{\text{in}}(\Delta)}\ . \end{aligned}$$

Therefore, all basis-kernels with phase offset satisfy their according kernel-constraint. Following equation 62, the constant phase offset thus does not break gauge-equivariance. Regardless of the map type, all basis-kernels for the self-connection term are independent of θ (cf. Table 3) and therefore invariant against changes $\theta - \Delta$. Hence, there is nothing left to show.

□

A.4 Used training hyperparameters in the FAUST and MN10 benchmark

Table 12: [**Left**] The used hyperparameters in our FAUST experiments. [**Right**] The used hyperparameters in our ModelNet10 experiments.

architecture	learning rate	learning rate decay
ISC	0.001607898453845114	0.9864880235560984
GCNN	0.002046618521178776	0.9311311646714975
HSN	0.022097267612157905	0.8260252932597774
GEM-CNN	0.0018836448731641313	0.9256143992721044
GEM-CNN+	0.0009850466443080594	0.7501912420925704
EMAN	0.015038740598499179	0.8783889303633541
EMAN+	0.010328768164729468	0.8215298221367153

architecture	learning rate	learning rate decay
ISC	0.0002410105106189093	0.8034224441564631
GCNN	0.0006115439743163413	0.9880825277196467
HSN	0.010805416052491235	0.6396591826543045
GEM-CNN	0.0009852045395011522	0.999999
GEM-CNN+	0.0009121999513112784	0.6192298704092247
EMAN	0.005571247671593134	0.6317157699404963
EMAN+	0.006066563780736304	0.6118024021331133

A.5 FAUST benchmark: ANOVA tables

Table 13: The tables used to conduct ANOVA hypothesis tests on observed test-metrics (Table 7, left). [**Left**] P-values for accuracies. [**Right**] P-values for geodesic-errors.

Charting algorithm		DGPC	FMM	HDM	TP	p-value
ISC		0.214964	0.343505	0.351705	0.221132	
GCNN		0.145929	0.290254	0.339064	0.239666	
HSN		0.305689	0.311705	0.317772	0.300319	
GEM-CNN	$c_R = 0.01$	0.208723	0.253338	0.284086	0.162816	0.000087
GEM-CNN+		0.160356	0.264158	0.293418	0.158810	
EMAN		0.192896	0.250290	0.276335	0.142663	
EMAN+		0.152126	0.266313	0.301488	0.166023	
ISC		0.174289	0.433106	0.397417	0.328977	
GCNN		0.165370	0.391226	0.382801	0.295885	
HSN		0.327627	0.374238	0.373599	0.343737	
GEM-CNN	$c_R = 0.02$	0.353832	0.449165	0.449594	0.360806	0.000149
GEM-CNN+		0.315276	0.474057	0.468512	0.383716	
EMAN		0.373077	0.422801	0.432692	0.367954	
EMAN+		0.330152	0.474906	0.480798	0.408062	
ISC		0.230776	0.409761	0.405936	0.340058	
GCNN		0.099804	0.419151	0.356814	0.277293	
HSN		0.336328	0.396691	0.391074	0.350958	
GEM-CNN	$c_R = 0.03$	0.343353	0.460029	0.448004	0.392525	0.000126
GEM-CNN+		0.296923	0.512068	0.497453	0.422496	
EMAN		0.343396	0.460189	0.447895	0.392293	
EMAN+		0.304702	0.530276	0.536343	0.424898	
ISC		0.224078	0.411560	0.416988	0.329477	
GCNN		0.158607	0.349724	0.355363	0.268367	
HSN		0.330718	0.395653	0.382772	0.338810	
GEM-CNN	$c_R = 0.04$	0.301647	0.439528	0.429833	0.364543	0.000007
GEM-CNN+		0.258578	0.505131	0.488962	0.402525	
EMAN		0.307750	0.429927	0.433287	0.376597	
EMAN+		0.253048	0.493208	0.497482	0.403585	
ISC		0.201923	0.378766	0.318179	0.290813	
GCNN		0.194303	0.278070	0.305581	0.201219	
HSN		0.318360	0.372010	0.344071	0.316967	
GEM-CNN	$c_R = 0.05$	0.263999	0.398549	0.378244	0.332837	0.000104
GEM-CNN+		0.201567	0.442627	0.420929	0.372068	
EMAN		0.276023	0.403621	0.363302	0.336836	
EMAN+		0.229507	0.445319	0.401974	0.375784	

Charting algorithm		DGPC	FMM	HDM	TP	p-value
ISC		0.068228	0.030480	0.020919	0.148622	
GCNN		0.130104	0.065305	0.024641	0.089148	
HSN		0.167584	0.160045	0.153959	0.184847	
GEM-CNN	$c_R = 0.01$	0.390258	0.339245	0.284664	0.434796	0.296582
GEM-CNN+		0.441575	0.372734	0.302522	0.595879	
EMAN		0.406575	0.434131	0.290759	0.543465	
EMAN+		0.452490	0.379512	0.243496	0.557461	
ISC		0.031049	0.005378	0.008183	0.011433	
GCNN		0.029545	0.009350	0.009014	0.013743	
HSN		0.166018	0.144566	0.138549	0.187097	
GEM-CNN	$c_R = 0.02$	0.142992	0.019390	0.017566	0.120338	0.017364
GEM-CNN+		0.259977	0.007490	0.009108	0.087797	
EMAN		0.117511	0.039059	0.031486	0.121024	
EMAN+		0.181257	0.007340	0.005389	0.048093	
ISC		0.015411	0.005605	0.005432	0.008571	
GCNN		0.042819	0.005026	0.008889	0.012446	
HSN		0.168459	0.126260	0.133177	0.176957	
GEM-CNN	$c_R = 0.03$	0.114716	0.011438	0.015145	0.048718	0.009301
GEM-CNN+		0.251346	0.000000	0.001266	0.025531	
EMAN		0.118870	0.013367	0.017408	0.062960	
EMAN+		0.217056	0.000000	0.000000	0.022581	
ISC		0.016235	0.004630	0.004132	0.008708	
GCNN		0.023708	0.008140	0.007097	0.011943	
HSN		0.166004	0.128428	0.139585	0.202250	
GEM-CNN	$c_R = 0.04$	0.124319	0.015546	0.017799	0.062328	0.015118
GEM-CNN+		0.279305	0.000000	0.002741	0.024139	
EMAN		0.128974	0.024410	0.019893	0.078317	
EMAN+		0.290512	0.001900	0.001213	0.025459	
ISC		0.018960	0.005647	0.008369	0.010663	
GCNN		0.020548	0.010999	0.009208	0.016013	
HSN		0.193798	0.153771	0.171131	0.218261	
GEM-CNN	$c_R = 0.05$	0.132226	0.028793	0.034786	0.079397	0.026512
GEM-CNN+		0.394546	0.009931	0.014403	0.036444	
EMAN		0.152202	0.035660	0.064651	0.099171	
EMAN+		0.322086	0.008624	0.019737	0.030680	

Table 14: The tables used to conduct ANOVA hypothesis tests on observed test-accuracies (Table 7, right). [**Left**] P-values for accuracies. [**Right**] P-values for geodesic-errors.

Chart max. radius c_R		0.01	0.02	0.03	0.04	0.05	p-value
ISC		0.214964	0.174289	0.230776	0.224078	0.201923	
GCNN		0.145929	0.165370	0.099804	0.158607	0.194303	
HSN		0.305689	0.327627	0.336328	0.330718	0.318360	
GEM-CNN	DGPC	0.208723	0.353832	0.343353	0.301647	0.263999	0.116119
GEM-CNN+		0.160356	0.315276	0.296923	0.258578	0.201567	
EMAN		0.192896	0.373077	0.343396	0.307750	0.276023	
EMAN+		0.152126	0.330152	0.304702	0.253048	0.229507	
ISC		0.343505	0.433106	0.409761	0.411560	0.378766	
GCNN		0.290254	0.391226	0.419151	0.349724	0.278070	
HSN		0.311705	0.374238	0.396691	0.395653	0.372010	
GEM-CNN	FMM	0.253338	0.449165	0.460029	0.439528	0.398549	0.000001
GEM-CNN+		0.264158	0.474057	0.512068	0.505131	0.442627	
EMAN		0.250290	0.422801	0.460189	0.429927	0.403621	
EMAN+		0.266313	0.474906	0.530276	0.493208	0.445319	
ISC		0.351705	0.397417	0.405936	0.416988	0.318179	
GCNN		0.339064	0.382801	0.356814	0.355363	0.305581	
HSN		0.317772	0.373599	0.391074	0.382772	0.344071	
GEM-CNN	HDM	0.284086	0.449594	0.448004	0.429833	0.378244	0.000025
GEM-CNN+		0.293418	0.468512	0.497453	0.488962	0.420929	
EMAN		0.276335	0.432692	0.447895	0.433287	0.363302	
EMAN+		0.301488	0.480798	0.536343	0.497482	0.401974	
ISC		0.221132	0.328977	0.340058	0.329477	0.290813	
GCNN		0.239666	0.295885	0.277293	0.268367	0.201219	
HSN		0.300319	0.343737	0.350958	0.338810	0.316967	
GEM-CNN	TP	0.162816	0.360806	0.392525	0.364543	0.332837	0.000003
GEM-CNN+		0.158810	0.383716	0.422496	0.402525	0.372068	
EMAN		0.142663	0.367954	0.392293	0.376597	0.336836	
EMAN+		0.166023	0.408062	0.424898	0.403585	0.375784	

Chart max. radius c_R		0.01	0.02	0.03	0.04	0.05	p-value
ISC		0.068228	0.031049	0.015411	0.016235	0.018960	
GCNN		0.130104	0.029545	0.042819	0.023708	0.020548	
GCNN		0.167584	0.166018	0.168459	0.166004	0.193798	
GEM-CNN	DGPC	0.390258	0.142992	0.114716	0.124319	0.132226	0.092728
GEM-CNN+		0.441575	0.259977	0.251346	0.279305	0.394546	
EMAN		0.406575	0.117511	0.118870	0.128974	0.152202	
EMAN+		0.452490	0.181257	0.217056	0.290512	0.322086	
ISC		0.030480	0.005378	0.005605	0.004630	0.005647	
GCNN		0.065305	0.009350	0.005026	0.008140	0.010999	
GCNN		0.160045	0.144566	0.126260	0.128428	0.153771	
GEM-CNN	FMM	0.339245	0.019390	0.011438	0.015546	0.028793	0.000040
GEM-CNN+		0.372734	0.007490	0.000000	0.000000	0.009931	
EMAN		0.434131	0.039059	0.013367	0.024410	0.035660	
EMAN+		0.379512	0.007340	0.000000	0.001900	0.008624	
ISC		0.020919	0.008183	0.005432	0.004132	0.008369	
GCNN		0.024641	0.009014	0.008889	0.007097	0.009208	
GCNN		0.153959	0.138549	0.133177	0.139585	0.171131	
GEM-CNN	HDM	0.284664	0.017566	0.015145	0.017799	0.034786	0.000571
GEM-CNN+		0.302522	0.009108	0.001266	0.002741	0.014403	
EMAN		0.290759	0.031486	0.017408	0.019893	0.064651	
EMAN+		0.243496	0.005389	0.000000	0.001213	0.019737	
ISC		0.148622	0.011433	0.008571	0.008708	0.010663	
GCNN		0.089148	0.013743	0.012446	0.011943	0.016013	
GCNN		0.184847	0.187097	0.176957	0.202250	0.218261	
GEM-CNN	TP	0.434796	0.120338	0.048718	0.062328	0.079397	0.000036
GEM-CNN+		0.595879	0.087797	0.025531	0.024139	0.036444	
EMAN		0.543465	0.121024	0.062960	0.078317	0.099171	
EMAN+		0.557461	0.048093	0.022581	0.025459	0.030680	

A.6 ModelNet10 benchmark: ANOVA tables

Table 15: The tables used to conduct ANOVA hypothesis tests on observed test-accuracies (Table 10, right).

Chart max. radius c_R		0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09	0.10	p-value
ISC		0.596916	0.637665	0.754405	0.784141	0.814978	0.829295	0.835903	0.841410	0.854626	0.861233	
GCNN		0.709251	0.710352	0.743392	0.751101	0.715859	0.792952	0.744493	0.792952	0.750000	0.801762	
HSN		0.604626	0.633260	0.699339	0.722467	0.715859	0.726872	0.743392	0.752203	0.744493	0.758811	
GEM-CNN	DGPC	0.118943	0.240088	0.550661	0.628855	0.687225	0.690529	0.678414	0.698238	0.675110	0.675110	0.007737
GEM-CNN+		0.204846	0.382159	0.678414	0.734581	0.779736	0.789648	0.777533	0.798458	0.795154	0.799559	
EMAN		0.118943	0.203744	0.431718	0.493392	0.578194	0.453745	0.535242	0.496696	0.607930	0.484582	
EMAN+		0.122247	0.178414	0.436123	0.491189	0.440529	0.431718	0.471366	0.454846	0.422908	0.405286	
ISC		0.638767	0.633260	0.758811	0.754405	0.794053	0.788546	0.802863	0.811674	0.820485	0.808370	
GCNN		0.732379	0.675110	0.704846	0.746696	0.700441	0.743392	0.748899	0.722467	0.761013	0.748899	
HSN		0.629956	0.612335	0.671806	0.660793	0.616740	0.645374	0.660793	0.651982	0.676211	0.694934	
GEM-CNN	FMM	0.118943	0.254405	0.541850	0.615639	0.649780	0.687225	0.689427	0.649780	0.660793	0.635463	0.032900
GEM-CNN+		0.166300	0.287445	0.594714	0.650881	0.676211	0.655286	0.687225	0.718062	0.688326	0.694934	
EMAN		0.127753	0.180617	0.395374	0.505507	0.556167	0.486784	0.534141	0.637665	0.493392	0.505507	
EMAN+		0.114537	0.186123	0.432819	0.374449	0.414097	0.406388	0.409692	0.417401	0.373348	0.400881	
ISC		0.656388	0.709251	0.778634	0.744493	0.795154	0.783040	0.807269	0.789648	0.832599	0.843612	
GCNN		0.686123	0.687225	0.723568	0.720264	0.743392	0.740088	0.755507	0.769824	0.759912	0.781938	
HSN		0.612335	0.645374	0.672907	0.659692	0.640969	0.640969	0.679515	0.655286	0.679515	0.658590	
GEM-CNN	HDM	0.153084	0.350220	0.637665	0.661894	0.653084	0.632159	0.620044	0.613436	0.668502	0.665198	0.026721
GEM-CNN+		0.212555	0.421806	0.697137	0.633260	0.640969	0.654185	0.672907	0.658590	0.701542	0.700441	
EMAN		0.148678	0.212555	0.486784	0.503304	0.527533	0.506608	0.574890	0.446035	0.599119	0.656388	
EMAN+		0.136564	0.231278	0.495595	0.407489	0.406388	0.425110	0.334802	0.400881	0.386564	0.334802	
ISC		0.632159	0.646476	0.692731	0.742291	0.788546	0.795154	0.799559	0.767621	0.800661	0.840308	
GCNN		0.697137	0.657489	0.719163	0.750000	0.733480	0.764317	0.698238	0.772026	0.743392	0.740088	
HSN		0.637665	0.598018	0.646476	0.632159	0.622247	0.636564	0.646476	0.631057	0.647577	0.648678	
GEM-CNN	TP	0.109031	0.178414	0.504405	0.593612	0.624449	0.604626	0.612335	0.583700	0.572687	0.562775	0.099190
GEM-CNN+		0.111233	0.188326	0.544053	0.601322	0.589207	0.610132	0.637665	0.603524	0.642070	0.649780	
EMAN		0.110132	0.167401	0.278634	0.409692	0.515419	0.542952	0.506608	0.495595	0.417401	0.482379	
EMAN+		0.110132	0.172907	0.302863	0.297357	0.333700	0.314978	0.279736	0.327092	0.286344	0.308370	

Table 16: The first 5 tables used to conduct ANOVA hypothesis tests on observed test-accuracies (Table 10, left).

Charting algorithm		DGPC	FMM	HDM	TP	p-value
ISC		0.596916	0.638767	0.656388	0.632159	
GCNN		0.709251	0.732379	0.686123	0.697137	
HSN		0.604626	0.629956	0.612335	0.637665	
GEM-CNN	$c_R = 0.01$	0.118943	0.118943	0.153084	0.109031	0.997933
GEM-CNN+		0.204846	0.166300	0.212555	0.111233	
EMAN		0.118943	0.127753	0.148678	0.110132	
EMAN+		0.122247	0.114537	0.136564	0.110132	
ISC		0.637665	0.633260	0.709251	0.646476	
GCNN		0.710352	0.675110	0.687225	0.657489	
HSN		0.633260	0.612335	0.645374	0.598018	
GEM-CNN	$c_R = 0.02$	0.240088	0.254405	0.350220	0.178414	0.892969
GEM-CNN+		0.382159	0.287445	0.421806	0.188326	
EMAN		0.203744	0.180617	0.212555	0.167401	
EMAN+		0.178414	0.186123	0.231278	0.172907	
ISC		0.754405	0.758811	0.778634	0.692731	
GCNN		0.743392	0.704846	0.723568	0.719163	
HSN		0.699339	0.671806	0.672907	0.646476	
GEM-CNN	$c_R = 0.03$	0.550661	0.541850	0.637665	0.504405	0.498758
GEM-CNN+		0.678414	0.594714	0.697137	0.544053	
EMAN		0.431718	0.395374	0.486784	0.278634	
EMAN+		0.436123	0.432819	0.495595	0.302863	
ISC		0.784141	0.754405	0.744493	0.742291	
GCNN		0.751101	0.746696	0.720264	0.750000	
HSN		0.722467	0.660793	0.659692	0.632159	
GEM-CNN	$c_R = 0.04$	0.628855	0.615639	0.661894	0.593612	0.739200
GEM-CNN+		0.734581	0.650881	0.633260	0.601322	
EMAN		0.493392	0.505507	0.503304	0.409692	
EMAN+		0.491189	0.374449	0.407489	0.297357	
ISC		0.814978	0.794053	0.795154	0.788546	
GCNN		0.715859	0.700441	0.743392	0.733480	
HSN		0.715859	0.616740	0.640969	0.622247	
GEM-CNN	$c_R = 0.05$	0.687225	0.649780	0.653084	0.624449	0.763457
GEM-CNN+		0.779736	0.676211	0.640969	0.589207	
EMAN		0.578194	0.556167	0.527533	0.515419	
EMAN+		0.440529	0.414097	0.406388	0.333700	

Charting algorithm		DGPC	FMM	HDM	TP	p-value
ISC		0.829295	0.788546	0.783040	0.795154	
GCNN		0.792952	0.743392	0.740088	0.764317	
HSN		0.726872	0.645374	0.640969	0.636564	
GEM-CNN	$c_R = 0.06$	0.690529	0.687225	0.632159	0.604626	0.867605
GEM-CNN+		0.789648	0.655286	0.654185	0.610132	
EMAN		0.453745	0.486784	0.506608	0.542952	
EMAN+		0.431718	0.406388	0.425110	0.314978	
ISC		0.835903	0.802863	0.807269	0.799559	
GCNN		0.744493	0.748899	0.755507	0.698238	
HSN		0.743392	0.660793	0.679515	0.646476	
GEM-CNN	$c_R = 0.07$	0.678414	0.689427	0.620044	0.612335	0.745866
GEM-CNN+		0.777533	0.687225	0.672907	0.637665	
EMAN		0.535242	0.534141	0.574890	0.506608	
EMAN+		0.471366	0.409692	0.334802	0.279736	
ISC		0.841410	0.811674	0.789648	0.767621	
GCNN		0.792952	0.722467	0.769824	0.772026	
HSN		0.752203	0.651982	0.655286	0.631057	
GEM-CNN	$c_R = 0.08$	0.698238	0.649780	0.613436	0.583700	0.640547
GEM-CNN+		0.798458	0.718062	0.658590	0.603524	
EMAN		0.496696	0.637665	0.446035	0.495595	
EMAN+		0.454846	0.417401	0.400881	0.327092	
ISC		0.854626	0.820485	0.832599	0.800661	
GCNN		0.750000	0.761013	0.759912	0.743392	
HSN		0.744493	0.676211	0.679515	0.647577	
GEM-CNN	$c_R = 0.09$	0.675110	0.660793	0.668502	0.572687	0.640513
GEM-CNN+		0.795154	0.688326	0.701542	0.642070	
EMAN		0.607930	0.493392	0.599119	0.417401	
EMAN+		0.422908	0.373348	0.386564	0.286344	
ISC		0.861233	0.808370	0.843612	0.840308	
GCNN		0.801762	0.748899	0.781938	0.740088	
HSN		0.758811	0.694934	0.658590	0.648678	
GEM-CNN	$c_R = 0.1$	0.675110	0.635463	0.665198	0.562775	0.825467
GEM-CNN+		0.799559	0.694934	0.700441	0.649780	
EMAN		0.484582	0.505507	0.656388	0.482379	
EMAN+		0.405286	0.400881	0.334802	0.308370	