LEARNING GEOMETRIC OPERATORS ON MESHES

Yu Wang
MIT, USA
wangyu9@mit.edu

Vladimir Kim
Adobe Research, USA
vokim@adobe.com

Michael M. Bronstein
Imperial College London, UK
USI, Switzerland
Intel, Israel
michael.bronstein@gmail.com

Justin Solomon
MIT, USA
jsolomon@mit.edu

ABSTRACT

Applications in computer vision, simulation, and computer-aided design motivate interest in deep learning methods that operate directly on triangle meshes. As an analogy to convolutional filters on images, these methods successively apply local geometric operators like the Laplacian and the Dirac operator to per-element features; this pipeline can be interpreted as applying a filter to features written in the operators' spectral bases. Such a technique is analogous to using convolutional filters with fixed kernels on images, which severely limits representation capacity. As a flexible alternative, we propose learning geometric operators from data in a task-specific fashion. Inspired by numerical algorithms operating on irregular domains, our framework learns from meshes using a parametric learnable family of linear operators, generalizing previous architectures.

1 INTRODUCTION

While deep learning has long been applied to grid-structured domains, there has been increasing interest in methods that operate on triangle meshes, a natural representation in geometric deep learning (Bronstein et al., 2017). It may appear that graph-based deep learning methodologies apply directly to meshes, but key added structure distinguishes meshes from general graphs. In particular, graph-based learning is designed to capture combinatorial structures like node degrees; combinatorial information, however, is irrelevant to mesh geometry and is likely to differ among multiple representations of the same shape. Indeed, in some sense the goal of geometric learning is to extract information that is *invariant* to irrelevant combinatorial structure, fundamentally distinguishing problems in this domain.

Early attempts in graphics and shape analysis use the Laplacian of the vertex adjacency graph of the triangle mesh. These methods, however, were quickly replaced due to dependency and sensitivity to sampling. As an alternative, more recent methods use discretizations of geometric operators like the surface (cotangent) Laplacian, which converges to the manifold Laplace–Beltrami operator under refinement; this property alleviates some sensitivity to meshing. A recent example, Surface Networks (Kostrikov et al., 2018), follows this pattern and uses the Dirac and Laplacian operators to diffuse and propagate information to neighboring mesh vertices in a *local geometry-aware fashion*. Backed by theoretical understanding and physical interpretation, these operators have been proven powerful for a variety of tasks. These methods, however, implicitly use *fixed kernels*, which severely limits their representation capacity. The choice and/or design of appropriate operators is carried out manually, requiring domain knowledge and empirical testing.

In this paper, we pursue an alternative approach: We learn the appropriate operator for a specific task from a parametric family, generalizing existing methods. In particular, we propose a framework capable of learning a family of discretized operators via the finite element methods (FEM), including the Laplacian and Dirac operators.

A challenge when learning geometric operators is to parameterize their entries consistently, ensuring domain independence and discretization invariance. We design a parametric family of discretized operators encouraging these principles. With our novel "operator layer" as the basic building block, we propose an end-to-end trainable deep learning architecture combining operators in a fashion that reflects information flows in numerical geometry processing algorithms. Layers of our network resemble steps of iterative methods used e.g. to compute eigenvalues or solve linear systems. This design—combined with our generalization of operators previously used to design successful hand-crated and learned features—benefits from and improves upon the success of algorithms in both learning and classical geometry processing.

Related Work. A large number of methods for geometric deep learning have been proposed in recent years, including: *Volumetric grid*-based methods (Wu et al., 2015; Qi et al., 2016b); *Multiple-view image*-based methods (Su et al., 2015; Huang et al., 2018); *Spectral* domain methods (Bruna et al., 2014; Defferrard et al., 2016; Yi et al., 2016); *Chart*-based spatial methods: Geodesic CNN (GCNN) (Masci et al., 2015), Anisotropic CNN (ACNN) (Boscaini et al., 2015), and Tangent Convolutions (TC) (Tatarchenko et al., 2018); *Global parameterization*-based methods (Sinha et al., 2016) (Maron et al., 2017); *Point cloud*-based methods: PointNet (Qi et al., 2016a), point convolution (Atzmon et al., 2018), continuous convolution (Wang et al., 2018a), dynamic graph layers (Wang et al., 2018c); *Graph*-based methods: (Monti et al., 2016; Qi et al., 2017; Verma et al., 2018; Litany et al., 2018). There are also very recent work for learning on triangle meshes (Kostrikov et al., 2018; Ranjan et al., 2018; Tan et al., 2018; Gao et al., 2018; Hanocka et al., 2018).

2 Learnable Operators and Learning Architecture

Desiderata. Many transformations can be applied to meshes without affecting their interpretation. Geometric learning algorithms must be *invariant* to these changes, motivating a list of considerations for design of a mesh-based learning pipeline:

1. *Permutation invariance:* Since there is no canonical ordering of mesh vertices, the learning procedure should not depend on the permutation of vertices and faces.
2. *Representation invariance:* The procedure should produce similar results when vertices are shifted along the surface or when the mesh topology is perturbed.
3. *Transformation invariance:* It may be necessary to ignore certain classes of geometric transformations. In image analysis, common invariances are to in-plane translation or (less frequently) rotation. For shapes, invariance to rigid motion or isometry is often desirable; the latter is commonly enforced for articulated shapes.
4. *Domain independence:* The algorithm should generalize across different shape categories rather than remaining specific to deformations of a single base object.

These requirements are not unfamiliar in geometry processing. Laplacian shape analysis is such a solution enjoys all the desiderata, having been successfully applied to various tasks including shape descriptor, segmentation, and correspondence (Reuter et al., 2009; 2006; Kokkinos et al., 2012; Bronstein et al., 2011; Ovsjanikov et al., 2012) and integrated into learning pipelines (Litman & Bronstein, 2014; Litany et al., 2017). Recently (Cosmo et al., 2018) study the inverse problem of recovering surface from eigenvalues. A popular way to address these desiderata in geometry processing is to construct eigenvalue problems for carefully-designed operators whose spectra capture geometric information.

We consider the following two common settings in geometric learning:

- *Global tasks* learn a function $\mathbf{f}(\cdot)$ taking a mesh $\mathcal{T}$ to a latent embedding in $\mathbb{R}^k$.
- *Vertex-wise tasks* learn a function $\mathbf{F}(\cdot)$ from a mesh $\mathcal{T}$ to k-dimensional per-vertex features.

The following proposition indicates that operator-based methods for these learning tasks can only depend on spectral structure:

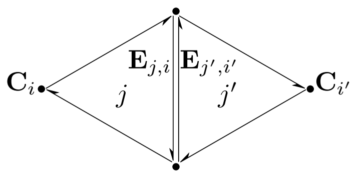
Proposition 1 *Under appropriate assumptions on the invariant structure of* $\mathbf{f}$ *and* $\mathbf{F}$*, there is some geometric operator* $\mathbf{A}$ *discretized on the mesh* $\mathcal{T}$ *with the eigendecomposition* $\mathbf{A} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^\top$*, such that* $\mathbf{f}(\mathcal{T})$ *only depends on* $\mathbf{\Lambda}$*, and* $\mathbf{F}(\mathcal{T})$ *only depends on* $\mathbf{\Lambda}$ *and* $\mathbf{Q}$.

Proposition 1 suggests that learning architectures on meshes should learn functions of operator spectra; detailed reasoning can be found in the supplementary document. Spectra of individual fixed operators, however, have limitations. For example, the intrinsic Laplacian is insensitive to bending without stretching. It has been shown that the Dirac spectrum is sensitive to high-frequency detail, suitable for tasks like texture detection but not higher-level tasks like classification (Wang et al., 2018b). Fixed operators also imply irreversible assumptions about the data: The Laplacian is intrinsic, and the Dirac is rigid-motion invariant. Ideally, the level of invariance to assorted transformations should be learned from data. For these reasons, we propose to learn the appropriate operators given a dataset and learning task.

2.1 OPERATOR PARAMETERIZATION

Convolutional neural networks (LeCun et al., 1989) are built on (linear) convolutional operators. In theory, each operator can be obtained by creating a #pixels $\times$ #pixels sparse matrix and filling in entries from the convolution kernel. Analogously we parameterize linear operators on meshes as #face$\times$#vertex sparse matrices, whose entries are *domain-dependent*. Extra complexity comes from the fact that we have to handle irregular domains; local geometry is used to "realize" the operator kernel, resulting in domain-dependent matrix entries.

As with graph neural networks (Scarselli et al., 2009) and their adaptations to meshes, in each layer of our architecture we allow each vertex to communicate to its immediate neighbors on the mesh. Hence, we consider operators whose discrete form has sparsity determined by the mesh.



Parameterization of a single operator. We begin by describing the form of our parameterized operator. Each layer of our network involves a set of c operators $\mathbf{H}^1, \ldots, \mathbf{H}^c \in \mathbb{R}^{f\times n}$, where c is the number of channels, and n, f are numbers of vertices and faces, respectively. Each $\mathbf{H}^k \in \mathbb{R}^{f\times n}$ is a *rectangular* matrix taking one value per vertex to one value per face. We will use $\mathbf{H}^k_{j,i}$ to refer to the element at position (j, i) of matrix $\mathbf{H}^k$, and we will use $\mathbf{H}_{j,i}$ (no superscript) to refer to the vector in $\mathbb{R}^c$ of values $\mathbf{H}^k_{j,i}$ over all k. We will use i for vertices and j for triangles. Figure 2.1 illustrates how we gather features from the mesh to construct each operator $\mathbf{H}^k$. We prescribe $\mathbf{H}^k_{j,i} = 0$ any time triangle j is *not* adjacent to vertex i; that is, $\mathbf{H}^k$ has a sparsity pattern determined by the mesh topology. Features for constructing $\mathbf{H}^k$ are gathered from two sources:

- *Directed edge* features, which are associated to an orientation of each edge. As shown in Figure 2.1, there are two directed edges per edge on the mesh, and three directed edges per triangle ordered by (counterclockwise) orientation. We store these features in a tensor $\mathbf{E} \in \mathbb{R}^{f\times 3\times d}$, where d is the edge feature dimension.
- *Vertex features*, which are associated to individual vertices, stored in a matrix $\mathbf{C} \in \mathbb{R}^{n\times d'}$, where d' denotes the vertex feature dimension.

We will describe how these features are obtained from the input mesh in §2.2. Let $\mathbf{C}_i \in \mathbb{R}^{d'}$ denote the vertex feature of vertex i (i-th row of $\mathbf{C}$), and take j to be the index of a triangle adjacent to vertex i. Define $\mathbf{E}_{j,i} \in \mathbb{R}^d$ to be the directed edge feature associated to the directed edge of triangle j opposite from vertex i, as labelled in the figure above. Also note the $E_{j,i}$ and $E_{j',i'}$ as in the figure do not necessarily have the same value. Then, the corresponding element of $\mathbf{H}$ is computed as:

$$\mathbf{H}_{j,i} = \mathrm{MLP}_{\boldsymbol{\theta}}(\mathbf{C}_i, \mathbf{E}_{j,i}) \in \mathbb{R}^c, \tag{1}$$

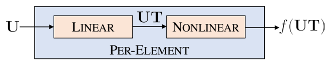
where $\mathrm{MLP}_{\boldsymbol{\theta}}$ denotes a multi-layer perceptron with parameters $\boldsymbol{\theta}$ to be learned. The same $\boldsymbol{\theta}$ is applied to all directed edges across all meshes. Note $\mathbf{H}_{j,i}$ can depend not only on features at the opposite vertex i, but also on features at the other vertices of triangle j; these features can be stored in $\mathbf{E}_{j,i}$. Indeed, it is possible to argue that *any* operator with our prescribed sparsity pattern and input features must take this form. In particular, in the supplementary document, we show that usual operators like the Dirac operator, the gradient operator, the Laplacian, and—importantly—a general linear operator discretized using first-order finite elements can be written in this form.

2.2 LEARNING ARCHITECTURE

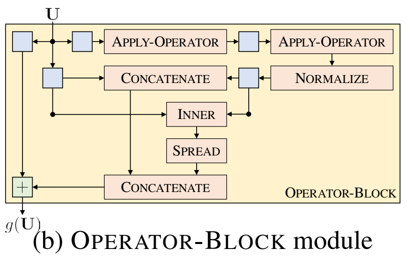
Our next step is to combine the parameterized operators above into a learning architecture. The design of our architecture is inspired by numerical algorithms for geometry processing; our method mimics information flows in these techniques.

Building Block Operations and Architecture We introduce a few very simple building blocks common in numerical solvers, which we will combine to formulate our learning architecture:

- INNER: $\mathbf{C}_1 \in \mathbb{R}^{n\times c}, \mathbf{C}_2 \in \mathbb{R}^{n\times c} \mapsto \operatorname{diag}(\mathbf{C}_1^\top \mathbf{C}_2) \in \mathbb{R}^c$
- SPREAD: $\lambda \in \mathbb{R} \mapsto \lambda \mathbf{1}^{n\times 1}$
- NORMALIZE: $\mathbf{C} \in \mathbb{R}^{n\times c} \mapsto \mathbf{S} \in \mathbb{R}^{n\times c}$, where $\mathbf{S}_{:,i} \leftarrow \mathbf{C}_{:,i}/\|\mathbf{C}_{:,i}\|_\mathbf{M}$ and $||\mathbf{C}||^2_\mathbf{M} := \operatorname{trace}(\mathbf{C}^\top \mathbf{M}\mathbf{C})$, where $\mathbf{M}$ is the mass matrix computed on the mesh
- LINEAR: $\mathbf{C} \in \mathbb{R}^{n\times c}, \mathbf{T} \in \mathbb{R}^{c\times t} \mapsto \mathbf{CT} \in \mathbb{R}^{n\times t}$
- NONLINEAR: Applies batch normalization and ReLU per vertex to $\mathbf{C} \in \mathbb{R}^{n\times c}$; in Figure 1(a), this nonlinearity is denoted using the function f
- PER-ELEMENT: Combined linear and nonlinear per-element operations; see Figure 1(a)
- CONCATENATE: Concatenates columns of two matrices
- APPLY-OPERATOR: Applies an instance of the operator $\mathbf{H}$ as obtained using Eq. 1 in §2.1 as $\mathbf{C} \in \mathbb{R}^{n\times c} \mapsto \mathbf{HC} \in \mathbb{R}^{f\times c}$, by multiplying the operator matrix $\mathbf{H}$ by the input signal matrix $\mathbf{C}$; when $\mathbf{H}^i \in \mathbb{R}^{f\times n}, i = 1, ..., k$ and $k > 1$, i.e., there are multiple channels, we apply each channel $\mathbf{H}^i$ independently and stack the results $\{\mathbf{H}^i\mathbf{C}\}_{i=1}^k$ into a matrix $\mathbb{R}^{f\times ck}$. If the input signal $\mathbf{C}$ has f rather than n rows, the transpose of $\mathbf{H}^i$ is applied. The output of this matrix-vector multiply is then scaled by triangle areas or inverse per-vertex areas if the operator is transposed.



(a) PER-ELEMENT module



(b) OPERATOR-BLOCK module

Figure 1: Modules used to construct our deep network architecture. Blue boxes in the OPERATOR-BLOCK module (b) refer to instances of the PER-ELEMENT module (a). Our full OPERATOR-BLOCK module takes as input a multi-channel signal per vertex and outputs another such signal.

A complete layer of our network, titled OPERATOR-BLOCK, is shown in Figure 1. The structure of this layer is similar to the one proposed for Surface Networks (Kostrikov et al., 2018, Figure 2). It is also straightforward to show that a single iteration of conjugate gradients or eigenvalue power iteration can be implemented using the steps in OPERATOR-BLOCK, allowing for the possibility that our network learns steps of these well-known numerical algorithms. Then our method are capable to replicate and generalize the very successful spectral shape and image analysis methods, which extract principal components and information insensitive to noises, graph combinatorics, and remeshing. Our overall learning architecture simply applies the OPERATOR-BLOCK multiple times; and for global tasks, our architecture additionally performs max pooling over vertices and transforms the result using a fully-connected layer.

Mesh features for operator parameterization. $\mathrm{MLP}_{\boldsymbol{\theta}}$ in (1) is a multi-layer perceptron with 3 hidden layers, each with width of 32 or 48; this amounts to applying four PER-ELEMENT modules from Figure 1(a) to each of the directed edge features. In (1), the linear transformation weights and bias in the perceptron are reused three times, once for each directed edge in a given triangle.

3 RESULTS

Method	#Parameters	Accuracy (%)
Ours	30k + 40k	99.0
Dirac	30k + 0	97.8
Dirac (large)	90k + 0	98.3

Table 1: MNIST classification accuracy (testing) for each method. #Parameters counts those outside and inside operators separately and does not include those used in the fully-connected layer.

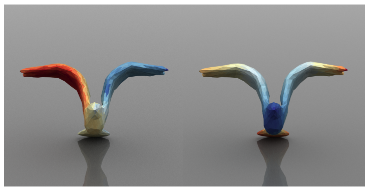
Mesh MNIST classification. We convert the 28×28 images from the MNIST handwritten digit dataset to meshes, so that each pixel becomes a vertex in the mesh, and the additional z coordinate is proportional to the pixel intensity. We build a simple architecture: The input per-vertex signal is the z coordinate; we apply three OPERATOR-BLOCK modules, each outputting a 48-dimensional per-vertex feature. Then, we apply a fully-connected layer, which flattens the $28 \times 28 \times 48$ features and successively reduces its dimension following $(28 \times 28 \times 48) \to 512 \to 256 \to 128 \to 10$. We test the resulting architecture for both our method and the Dirac Surface Networks. Note if we replace $\mathrm{MLP}_{\boldsymbol{\theta}}$ with a fixed formula, our architecture becomes a Dirac network. The table above summarizes the results, in which the parameters outside and inside operators are counted separately. Since our method uses more parameters than the Dirac networks due to its learnable operators, for the Dirac networks, we additionally test a larger Dirac network, by increasing the output dimensions of the OPERATOR-BLOCK to 96. We see that our method, with a moderate number of parameters, outperforms Dirac networks in both setups.

Method	Setup	Accuracy (%)
PointNet++	Raw Feature	60.18
Ours	Raw Feature	78.75
PointNet++	Global Feature	96.06
Ours	Global Feature	97.90

Nonrigid shape classification. We apply our method to the SHREC15 nonrigid shape classification task. We downsample the original meshes to 2000 faces. We compare our method with PointNet++ in setups similar to theirs, using an architecture which applies eight OPERATOR-BLOCK modules, each outputting a 32-dimensional per-vertex feature. PointNet++ takes in raw features (positions) as the input signal; accordingly, we set up our method to use raw mesh features for operator learning and random noise as the input signal. Another setting of PointNet++ uses auxiliary features including (PCA-reduced) multi-scale curvature, the heat kernel signature, and the wave kernel signature as input; accordingly we set up our method with heat kernel signatures as input and again use extrinsic raw features (positions, edge vectors, lengths and angles) to learn operators. We see that in both cases, our method outperforms PointNet++.

Method	#Parameters	Reconstruction Error
Laplacian	1019k	17.34
Dirac	1019k	16.84
Ours	1089k	12.98

Animation prediction. We compare our method with Surface Networks on the task of temporal prediction of nonlinear dynamics. We test our operator within their proposed architecture, with the same learning setup detailed in their §6.2. The table above shows that with the few parameters introduced by our learnable operator, we are able to outperform Surface Networks by a large margin.



Visualization of learned operator. While previous geometric deep learning algorithms rely on prescribed operators, our method learns *new* operators motivated by a task or dataset. Although the space of all possible operators is high-dimensional, we can use indirect means to visualize their action. Here we visualize operators learned in our animation prediction experiment. On the left we show a dominant eigenfunction of the operators. Since these eigenfunctions are difficult to distinguish visually from Laplacian eigenfunctions, on the right we use the framework of shape differences (Rustamov et al., 2013) to highlight regions where our operators are *different* from the cotangent Laplacian operator, which is probably responsible for the improvement. Interpretability of the learned geometric operators is an interesting next step for future work.

4 CONCLUSION

Learning from meshes is fundamentally different than learning from images. When processing mesh data, we must cope with irregular structures and the fact that changes as innocuous as rotation affect the coordinates of every vertex. Rather than shoehorning meshes into existing networks designed for images or graphs, our work is a step toward geometric deep learning built end-to-end for shapes. Drawing inspiration from geometry processing workflows, our procedure remains sensible from the perspective of classical and data-oriented mesh processing, with the flexibility of a parametric model. Our method essentially implements a generic linear operator assembler and learns from a family of sensible operators. In the future, we would like to solidify the connection of our framework to the finite element method (FEM) and experiment with function bases whose support reaches over a larger region on the mesh, leading to architectures that are dramatically different from GNNs and less sensitive to mesh discretization. We are also interested in drawing connections to discrete exterior calculus (DEC) (Hirani, 2003).

Acknowledgements. We thank Ersin Yumer for discussions during the design stage of this work. J. Solomon acknowledges the support of Army Research Office grant W911NF-12-R-0011, of NSF grant IIS-1838071, from an Amazon Research Award, from the MITIBM Watson AI Laboratory, from the Toyota-CSAIL Joint Research Center, and from the Skoltech-MIT Next Generation Program.

REFERENCES

Matan Atzmon, Haggai Maron, and Yaron Lipman. Point convolutional neural networks by extension operators. *arXiv preprint arXiv:1803.10091*, 2018.

Davide Boscaini, Davide Eynard, Drosos Kourounis, and Michael M Bronstein. Shape-from-operator: Recovering shapes from intrinsic operators. In *Computer Graphics Forum*, volume 34, pp. 265–274, 2015.

Susanne Brenner and Ridgway Scott. *The mathematical theory of finite element methods*, volume 15. Springer Science & Business Media, 2007.

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

Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann Lecun. Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations (ICLR2014), CBLS, April 2014*, 2014.

Venkat Chandrasekaran, Pablo A Parrilo, and Alan S Willsky. Convex graph invariants. *SIAM Review*, 54(3):513–541, 2012.

Albert Chern, Felix Knöppel, Ulrich Pinkall, and Peter Schröder. Shape from metric. *ACM Transactions on Graphics (TOG)*, 37(4):63, 2018.

Etienne Corman, Justin Solomon, Mirela Ben-Chen, Leonidas Guibas, and Maks Ovsjanikov. Functional characterization of intrinsic and extrinsic geometry. *ACM Transactions on Graphics (TOG)*, 36(2):14, 2017.

Luca Cosmo, Mikhail Panine, Arianna Rampini, Maks Ovsjanikov, Michael M Bronstein, and Emanuele Rodolà. Isospectralization, or how to hear shape, style, and correspondence. *arXiv:1811.11465*, 2018.

Keenan Crane, Ulrich Pinkall, and Peter Schröder. Spin transformations of discrete surfaces. In *ACM Transactions on Graphics (TOG)*, volume 30, pp. 104. ACM, 2011.

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

Lin Gao, Jie Yang, Yi-Ling Qiao, Yu-Kun Lai, Paul L Rosin, Weiwei Xu, and Shihong Xia. Automatic unpaired shape deformation transfer. In *SIGGRAPH Asia 2018 Technical Papers*, pp. 237. ACM, 2018.

Carolyn Gordon, David L Webb, and Scott Wolpert. One cannot hear the shape of a drum. *Bulletin of the American Mathematical Society*, 27(1):134–138, 1992.

Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. Meshcnn: A network with an edge. *arXiv preprint arXiv:1809.05910*, 2018.

Anil Nirmal Hirani. *Discrete exterior calculus*. PhD thesis, California Institute of Technology, 2003.

Haibin Huang, E Kalegorakis, Siddhartha Chaudhuri, Duygu Ceylan, V Kim, and E Yumer. Learning local shape descriptors with viewbased convolutional neural networks. *ACM Transactions on Graphics*, 2, 2018.

Mark Kac. Can one hear the shape of a drum? *American Mathematical Monthly*, 73(4):1–23, 1966.

Iasonas Kokkinos, Michael M Bronstein, Roee Litman, and Alex M Bronstein. Intrinsic shape context descriptors for deformable shapes. In *Proc. CVPR*, pp. 159–166, 2012.

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

Yann LeCun et al. Generalization and network design strategies. *Connectionism in perspective*, pp. 143–155, 1989.

Ron Levie, Federico Monti, Xavier Bresson, and Michael M Bronstein. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. *arXiv preprint arXiv:1705.07664*, 2017.

Michael Levitin, Leonid Parnovski, Iosif Polterovich, and David A Sher. Sloshing, Steklov and corners i: Asymptotics of sloshing eigenvalues. *arXiv:1709.01891*, 2017.

Or Litany, Tal Remez, Emanuele Rodolà, Alexander M Bronstein, and Michael M Bronstein. Deep functional maps: Structured prediction for dense shape correspondence. In *ICCV*, pp. 5660–5668, 2017.

Or Litany, Alex Bronstein, Michael Bronstein, and Ameesh Makadia. Deformable shape completion with graph convolutional autoencoders. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1886–1895, 2018.

Roee Litman and Alexander M Bronstein. Learning spectral descriptors for deformable shape correspondence. *IEEE transactions on pattern analysis and machine intelligence*, 36(1):171–180, 2014.

Hsueh-Ti Derek Liu, Alec Jacobson, and Keenan Crane. A Dirac operator for extrinsic shape analysis. In *Computer Graphics Forum*, volume 36, pp. 139–149. Wiley Online Library, 2017.

H. Maron, M. Galun, N. Aigerman, M. Trope, N. dym, , E. Yumer, V. Kim, and Y. Lipman. Convolutional neural networks on surfaces via seamless toric covers. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2017)*, 2017.

Jonathan Masci, Davide Boscaini, Michael Bronstein, and Pierre Vandergheynst. Geodesic convolutional neural networks on riemannian manifolds. In *Proceedings of the IEEE international conference on computer vision workshops*, pp. 37–45, 2015.

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. *arXiv preprint arXiv:1611.08402*, 2016.

Maks Ovsjanikov, Mirela Ben-Chen, Justin Solomon, Adrian Butscher, and Leonidas Guibas. Functional maps: a flexible representation of maps between shapes. *Transactions on Graphics*, 31(4): 30, 2012.

Ulrich Pinkall and Konrad Polthier. Computing discrete minimal surfaces and their conjugates. *Experimental Mathematics*, 2(1):15–36, 1993.

Adrien Poulenard and Maks Ovsjanikov. Multi-directional geodesic neural networks via equivariant convolution. *Transactions on Graphics*, 37(6), 2018.

Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *arXiv preprint arXiv:1612.00593*, 2016a.

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 *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5648–5656, 2016b.

Charles R Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *arXiv preprint arXiv:1706.02413*, 2017.

Anurag Ranjan, Timo Bolkart, Soubhik Sanyal, and Michael J Black. Generating 3d faces using convolutional mesh autoencoders. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 704–720, 2018.

Steven C Rennich, Darko Stosic, and Timothy A Davis. Accelerating sparse Cholesky factorization on GPUs. In *Proceedings of the 4th Workshop on Irregular Applications: Architectures and Algorithms*, pp. 9–16. IEEE Press, 2014.

Martin Reuter, Franz-Erich Wolter, and Niklas Peinecke. Laplace–Beltrami spectra as 'shape-DNA' of surfaces and solids. *Computer-Aided Design*, 38(4):342–366, 2006.

Martin Reuter, Silvia Biasotti, Daniela Giorgi, Giuseppe Patanè, and Michela Spagnuolo. Discrete Laplace–Beltrami operators for shape analysis and segmentation. *Computers & Graphics*, 33(3): 381–390, 2009.

Raif M Rustamov, Maks Ovsjanikov, Omri Azencot, Mirela Ben-Chen, Frédéric Chazal, and Leonidas Guibas. Map-based exploration of intrinsic shape differences and variability. *Transactions on Graphics*, 32(4):72, 2013.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

Ayan Sinha, Jing Bai, and Karthik Ramani. Deep learning 3d shape surfaces using geometry images. In *European Conference on Computer Vision*, pp. 223–240. Springer, 2016.

Hao Su, Charles R Qi, Yangyan Li, and Leonidas J Guibas. Render for cnn: Viewpoint estimation in images using cnns trained with rendered 3d model views. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2686–2694, 2015.

Qingyang Tan, Lin Gao, Yu-Kun Lai, and Shihong Xia. Variational autoencoders for deforming 3d mesh models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5841–5850, 2018.

Maxim Tatarchenko, Jaesik Park, Vladlen Koltun, and Qian-Yi Zhou. Tangent convolutions for dense prediction in 3d. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3887–3896, 2018.

Federico Tombari, Samuele Salti, and Luigi Di Stefano. Unique signatures of histograms for local surface description. In *Proc. ECCV*, pp. 356–369, 2010.

Amir Vaxman, Marcel Campen, Olga Diamanti, Daniele Panozzo, David Bommes, Klaus Hildebrandt, and Mirela Ben-Chen. Directional field synthesis, design, and processing. In *Computer Graphics Forum*, volume 35, pp. 545–572. Wiley Online Library, 2016.

Nitika Verma, Edmond Boyer, and Jakob Verbeek. Feastnet: Feature-steered graph convolutions for 3d shape analysis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2598–2606, 2018.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, pp. 2692–2700, 2015.

Shenlong Wang, Simon Suo, Wei-Chiu Ma, Andrei Pokrovsky, and Raquel Urtasun. Deep parametric continuous convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2589–2597, 2018a.

Yu Wang, Mirela Ben-Chen, Iosif Polterovich, and Justin Solomon. Steklov spectral geometry for extrinsic shape analysis. *ACM Transactions on Graphics (TOG)*, 38(1):7, 2018b.

Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *arXiv preprint arXiv:1801.07829*, 2018c.

Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1912–1920, 2015.

Zi Ye, Olga Diamanti, Chengcheng Tang, Leonidas Guibas, and Tim Hoffmann. A unified discrete framework for intrinsic and extrinsic Dirac operators for geometry processing. In *Computer Graphics Forum*, volume 37, pp. 93–106. Wiley Online Library, 2018.

Li Yi, Hao Su, Xingwen Guo, and Leonidas Guibas. Syncspeccnn: Synchronized spectral cnn for 3d shape segmentation. *arXiv preprint arXiv:1612.00606*, 2016.

Wei Zeng, Ren Guo, Feng Luo, and Xianfeng Gu. Discrete heat kernel determines discrete riemannian metric. *Graphical Models*, 74(4):121–129, 2012.

SUPPLEMENTARY MATERIAL

A EXTENDED DISCUSSION OF RELATED WORK

The success of deep learning for image analysis and vision has inspired interest in developing analogous methods for geometric data. Unlike images, however, a key consideration in geometry involves the choice and manipulation of representations. The most straightforward extension of image-based methods uses a volumetric grid (Wu et al., 2015; Qi et al., 2016b) or using multiple-view images (Su et al., 2015; Huang et al., 2018), benefiting from classical grid-based deep learning. These approaches, however, have many drawbacks, including high complexity and a lack of invariance to transformations like deformation or rotation.

A new class of *geometric deep learning* algorithms directly operates on 3D representations like meshes and point clouds (Bronstein et al., 2017); this field is part of a broader program to generalize deep networks to graph-structured data. In this paper, we are primarily interested in geometric deep learning on meshes; we summarize related methods below.

Spatial approach. Spatial domain methods are *chart*-based, applying filters after parameterizing to a local coordinate system. Examples include Geodesic CNN (GCNN) (Masci et al., 2015), Anisotropic CNN (ACNN) (Boscaini et al., 2015), MoNet (Monti et al., 2016), and Tangent Convolutions (TC) (Tatarchenko et al., 2018). A central issue with these methods is that there exists no canonical parametrization or coordinate system for most surfaces, making it hard to resolve angular ambiguities. ACNN and TC align planar filters to principal curvature directions, which may not be continuous and are sensitive to noise. To resolve this issue, GCNN uses angular pooling all polar directions equally; this restricts the possible filters, in analogy to rotationally-symmetric kernels on images. This difficulty has recently been addressed with the equivariant convolution construction (Poulenard & Ovsjanikov, 2018).

Global parameterization-based methods map 3D surfaces to domains with shift-invariant structure such as the plane (Sinha et al., 2016) or a torus (Maron et al., 2017), in which CNN machinery is be applied to pushed-forward geometric signals. Planar mapping induces distortion and potentially requires cutting; furthermore, most parameterization methods do not consistently map the same parts of a surface to the same planar locations, creating artificial challenges.

Point cloud-based methods like PointNet (Qi et al., 2016a; 2017) operate without topological information, receiving as input an unordered collection of disconnected points. While point convolution (Atzmon et al., 2018), continuous convolution (Wang et al., 2018a), dynamic graph layers (Wang et al., 2018c), and other methods define convolution-like operations on point clouds to transfer information between samples, they discard topological cues afforded by meshes.

Operator approach. Laplacian shape analysis algorithms use the intrinsic *Laplace–Beltrami* (Laplacian) operator or its eigenfunctions and eigenvalues (which generalize Fourier bases and frequencies, resp.), achieving state-of-the-art shape classification results in non-learning settings. This methodology has been successfully applied to tasks including segmentation (Reuter et al., 2009), description (Reuter et al., 2006; Kokkinos et al., 2012), retrieval (Bronstein et al., 2011), and correspondence (Ovsjanikov et al., 2012). By construction, the Laplacian is invariant to rigid and nonrigid isometries and robust to remeshing; these properties make it a desirable choice for mesh processing.

Surface Networks (Kostrikov et al., 2018) generalize Graph Neural Networks (Scarselli et al., 2009) by replacing the adjacency matrix with the Laplacian or Dirac operators; the Dirac operator is used to add sensitivity to extrinsic deformation ignored by the (isometry-invariant) Laplacian operator. This method in some sense generalizes 2D CNNs to curved meshes using *fixed* kernels (stencils), which limits their representation capacity. Generalizing these methods, our method provides a framework to make the kernel *learnable*.

While Surface Networks apply operators directly as sparse matrices, they also can be applied in the spectral domain. This idea was proposed for graphs in (Bruna et al., 2014; Defferrard et al., 2016), and was extended to 3D shapes in (Yi et al., 2016); it allows for efficient application of certain operators and the application of learned filters to operator eigenvalues. Pointed out in e.g. (Bronstein et al., 2017), spectral coefficients do not adapt across domains when shapes are from different categories and cannot be put into pointwise correspondence. This issue aside, the Optimal Spectral Descriptor (Litman & Bronstein, 2014) learns spectral filter coefficients, and Deep Functional Maps (Litany et al., 2017) learn to refine the SHOT descriptor (Tombari et al., 2010); they are specifically for correspondence.

B Proposition 1 using an Operator View of Geometric Learning

We represent surfaces as triangle meshes $(\mathbf{X}, \mathbf{T})$, where $\mathbf{X} \in \mathbb{R}^{n\times 3}$ contains the (x, y, z) coordinates of one vertex per row and $\mathbf{T} \in \{1, \ldots, n\}^{f\times 3}$ contains a triplet of indices into the rows of $\mathbf{X}$ for each triangle in the mesh. Here, n is the number of vertices, and f is the number of triangles.

We consider learning from collections of meshes as $(\mathbf{X}, \mathbf{T})$ pairs, where the $\mathbf{T}$'s are not necessarily the same—that is, meshes can be triangulated differently. While one can endow the shape with additional per-vertex properties in $\mathbf{X}$ (e.g. texture, normal, or descriptors), we assume no extra information is provided beyond connectivity and coordinates.

B.1 Operator Representation

Just as a graph can be described using its adjacency or Laplacian matrices, there are many ways in which a shape can be expressed as an *operator*, i.e., a matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ assembled from $(\mathbf{X}, \mathbf{T})$.

Operator constructions can provide a bijection between meshes $(\mathbf{X}, \mathbf{T})$ and operators $\mathbf{A}$. That is, given a mesh $(\mathbf{X}, \mathbf{T})$, we can construct an operator $\mathbf{A}$ by some deterministic procedure, and conversely given the operator $\mathbf{A}$ we can recover $(\mathbf{X}, \mathbf{T})$, possibly up to rigid/nonrigid isometry, from $\mathbf{A}$. For example, the mesh Laplacian encodes edge lengths up to global scaling (Zeng et al., 2012), providing a means of recovering shape from an operator; (Boscaini et al., 2015; Corman et al., 2017; Chern et al., 2018) recover vertex coordinates up to isometry from Laplacians and related objects, providing a (nonconvex) way to invert this encoding. Another example is the single layer potential, which encodes the inverse distance between pairs of vertices (Wang et al., 2018b). The discrete Dirac operator (Crane et al., 2011; Liu et al., 2017; Ye et al., 2018; Kostrikov et al., 2018), though potentially not square, is another example.

In the language of operators, our two learning tasks can be equivalently written as finding functions of operators $\mathbf{f}(\mathbf{A}) : \mathbb{R}^{n\times n} \to \mathbb{R}^k$ and $\mathbf{F}(\mathbf{A}) : \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times k}$, respectively.

Our use of the term *operator* (rather than, e.g., "matrix") is suggestive of a long-standing duality in spectral geometry. A key realization in modern geometry is that operators like the Laplacian acting on functions—the Laplacian in $\mathbb{R}^n$ takes a function $f(x)$ to the second derivative $-\sum_{k=1}^{n} \partial^2 f/\partial x_i^2(x)$—encode geometric information in their eigenvalues and eigenfunctions. As is common in geometry processing, we design operators as matrices acting as functions represented using one value per mesh vertex. Many operators we will consider have sparsity determined by vertex adjacency, similar to a derivative operator in the smooth case.

Typical geometric deep learning architectures employ layers of the form $\mathbf{F}(\boldsymbol{\Delta})$, where $\boldsymbol{\Delta}$ is the Laplacian and $\mathbf{F}$ is a learnable function interpretable as a spectral filter applied to the eigenvalues: $\mathbf{F}(\boldsymbol{\Delta}) = \mathbf{V}\mathbf{F}(\boldsymbol{\Lambda})\mathbf{V}^\top$. If $\mathbf{F}$ is expressible in terms of matrix-matrix/matrix-vector products, it can be applied with linear complexity without explicit eigedecomposition. Popular choices for $\mathbf{F}$ include polynomial (Defferrard et al., 2016) or rational functions (Levie et al., 2017) with learnable coefficients.

B.2 Implications of Invariance

We now explore the implications of the desiderata in §2 on the operator representation from §B.1.

Given any permutation π of $\{1,\ldots,n\}$, denote the corresponding permutation matrix as $\mathbf{P} \in \{0,1\}^{n\times n}$. Applying π to our mesh $(\mathbf{X},\mathbf{T})$ yields a mesh $(\mathbf{P}^{-1}\mathbf{X},\pi(\mathbf{T}))$, where $\pi(\mathbf{T})$ substitutes entries in $\mathbf{F}$ with value k with $\pi(k)$; $\mathbf{P}^{-1}\mathbf{A}\mathbf{P}$ is its corresponding operator representation.

In operator language, requiring the learned functions $\mathbf{f},\mathbf{F}$ to be permutation-invariant is equivalent to constraining:

$$\begin{aligned}\mathbf{f}(\mathbf{A}) &= \mathbf{f}(\mathbf{P}^{-1}\mathbf{A}\mathbf{P})\\ \mathbf{P}^{-1}\mathbf{F}(\mathbf{A}) &= \mathbf{F}(\mathbf{P}^{-1}\mathbf{A}\mathbf{P})\end{aligned} \tag{2}$$

For settings with a fixed $\mathbf{A}$ (e.g. the graph Laplacian), permutation invariance leads to some theoretical properties; for example, if $\mathbf{f}$ is convex it can only depend on the "pseudo-eigenvalues" of $\mathbf{A}$ (Chandrasekaran et al., 2012).

Now let us consider the more general case. We apply the language of *functional maps* (Ovsjanikov et al., 2012), in which maps between meshes are expressed as operators taking functions on one mesh to functions on another. To establish notation, given the original mesh $(\mathbf{X},\mathbf{T})$ and remeshed version $(\tilde{\mathbf{X}},\tilde{\mathbf{T}})$, denote their operator representations as $\mathbf{A}$ and $\tilde{\mathbf{A}}$, resp.; let $\mathbf{M}$ and $\tilde{\mathbf{M}}$ denote the respective (diagonal) mass matrices, representing the local area elements on the two meshes. A functional map is a linear operator $\mathbf{C} \in \mathbb{R}^{\tilde{n}\times n}$ mapping a per-vertex function $\mathbf{u} \in \mathbb{R}^n$ on $(\mathbf{X},\mathbf{T})$ to a function $\tilde{\mathbf{u}} = \mathbf{C}\mathbf{u} \in \mathbb{R}^{\tilde{n}}$ on $(\tilde{\mathbf{X}},\tilde{\mathbf{T}})$.

Given an operator representation $\mathbf{A}$ of $(\mathbf{X},\mathbf{T})$, we can generate an operator representation $\tilde{\mathbf{A}}$ of $(\tilde{\mathbf{X}},\tilde{\mathbf{T}})$ by mapping a function to $(\mathbf{X},\mathbf{T})$, applying $\mathbf{A}$, and mapping it back: $\tilde{\mathbf{A}} = \mathbf{C}\mathbf{A}\mathbf{C}^{-1}$. Suppose $\mathbb{S}$ denotes some set of functional maps, e.g. those that preserve areas or angles. Invariance to the transformations in $\mathbb{S}$ can be written symbolically as

$$\begin{aligned}\mathbf{f}(\mathbf{A}) &= \mathbf{f}(\mathbf{C}\mathbf{A}\mathbf{C}^{-1})\\ \mathbf{F}(\mathbf{A}) &= \mathbf{C}^{-1}\mathbf{F}(\mathbf{C}\mathbf{A}\mathbf{C}^{-1})\end{aligned} \quad \text{whenever } \mathbf{C}\in\mathbb{S}. \tag{3}$$

A consequence of (Rustamov et al., 2013, Theorem 1) is that $\mathbf{C}$ is area-preserving exactly when the $\mathbf{C}$ is orthogonal with respect to the mass matrices $(\mathbf{M},\tilde{\mathbf{M}})$, that is, when $\mathbf{C}^\top\tilde{\mathbf{M}}\mathbf{C} = \mathbf{M}$. The challenge in our framework is that $(\mathbf{M},\tilde{\mathbf{M}})$ are not known; we only receive $\mathbf{A}$ as input. Enforcing (3) for *any* semidefinite $(\mathbf{M},\tilde{\mathbf{M}})$ is too strong for $\mathbb{S}$ and implies extremely restrictive constraints on $\mathbf{f}$ and $\mathbf{F}$.

As an alternative generalizing (2), we consider the case $\mathbf{M},\tilde{\mathbf{M}} = \mathbf{I}$, that is, when the mass matrices are both the identity; in this case $\mathbf{C}^\top\mathbf{C} = \mathbf{I}$, that is $\mathbf{C} \in O(n)$. This is roughly the case after isotropic remeshing, since typical triangulation algorithms lead to nearly identical masses assigned to all the vertices. As a way to understand the implications of assumptions like invariance under area-preserving transformation, we consider the following result for symmetric operators like the Laplacian:

Proposition 2 *Suppose $\mathbb{S} = O(n)$ in* (3)*, and apply eigendecomposition to write $\mathbf{A} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^\top$. Then,*

$$\mathbf{f}(\mathbf{A}) = \mathbf{f}(\mathbf{\Lambda}) \qquad \textit{and} \qquad \mathbf{F}(\mathbf{A}) = \mathbf{Q}\mathbf{F}(\mathbf{\Lambda}). \tag{4}$$

This proposition follows from straightforward arguments about orthogonal matrices.

Roughly this proposition indicates that invariance to simple geometric transformations and permutation implies that the learned functions involve just the eigenstructure of $\mathbf{A}$. Indeed, functions made from the spectra of discretized operators such as Laplacian have been proven in practice to be robust to mesh discretization and perturbation. Furthermore, deep learning architectures on graphs such as those in §B.1 explicitly learn functions of the Laplacian eigenvalues.

Several results from spectral geometry support some species of the converse to the discussion above. A landmark article in differential geometry poses the question, "Can one hear the shape of a drum?" (Kac, 1966), that is, given the eigenvalues of an operator like the Laplacian, can the shape be reconstructed? While the Laplacian does not have this property thanks to existence of isospectral non-isometric shapes (Gordon et al., 1992), in practice one can approximately reconstruct shapes from Laplacian spectra under mild assumptions (Cosmo et al., 2018).

Besides the Laplacian, there are other operators whose spectra determine a shape completely up to rigid motion. For example, 2D curves (polylines) can be recovered from eigenvalues of the Poincaré-Steklov operator (Levitin et al., 2017). As a corollary, cylindrical shapes (i.e. with straight parallel sides) can be recovered from eigenvalues of certain related operator. As another corollary, for shapes (e.g. convex polygon meshes) that can be recovered from projected outlines from finitely many views, the shape can be recovered from the Steklov eigenvalues of the projected outlines.

In these cases, if $\mathbf{f}$ inputs eigenvalues of the corresponding operator, it in effect has access to *all possible* information about the shape. These results support the intuition that eigendecomposition provides the relevant information about a shape after converting to an operator representation.

Our discussion implies a limit on the learning capacity of algorithms designed to be invariant to changes like permutation and deformation. Learning from quantities other than the spectrum is likely indicative of overfitting to the representation rather than the underlying geometry.

B.3 Why should the operator be learned?

Our formulation suggests that learning architectures on meshes should learn functions of operator spectra; common practice is to explicitly parametrize such functions e.g. as polynomials (§B.1) and apply them to a fixed operator such as the Laplacian. Here, we argue for the value of degrees of freedom afforded by making the operator learnable.

The best operator is problem-specific; different operators are sensitive to different features. Our analysis should be understood as showing that $\mathbf{f}$ *information-theoretically* depends largely on the spectrum of any invertible operator representation, but it implies nothing *computationally*. As indicated by the diverse set of operator representations applied in previous work, there are many ways to represent a shape as an operator that are all in some sense equivalent. The optimal function $\mathbf{f}$, however, may take an extremely complicated form for one operator while being simple for another. As an example, the diameter (largest pairwise distance) of a shape can be trivially obtained as the reciprocal of the smallest eigenvalue of the single layer potential operator, but it is less straightforward to obtain this quantity from the Dirac operator even if technically it can be recovered.

Individual fixed operators have limitations. The Laplacian is intrinsic, making it insensitive to bending without stretching. While the Dirac operator introduces extrinsic terms useful for learning (Kostrikov et al., 2018) and shape analysis (Liu et al., 2017), its spectrum is sensitive to high-frequency detail, suitable for tasks like texture detection but not higher-level tasks like classification (Wang et al., 2018b).

Fixed operators imply irreversible assumptions about the data: The Laplacian is intrinsic, and the Dirac is rigid-motion invariant. It is not clear, e.g., how to design operators invariant only to horizontal motion or specific directions of deformation. Ideally, the level of invariance to assorted transformations should be learned from data. Also, as mentioned earlier, fixed operators are analogous to 2D CNN with fixed stencils, whose representation capacity is limited.

C On the representation capacity of learnable operators

Consider the learnable operators with the formula in Eq. 1:

$$\mathbf{H}_{j,i} = \mathrm{MLP}_{\boldsymbol{\theta}}(\mathbf{C}_i, \mathbf{E}_{j,i}) \in \mathbb{R}^c,$$

A number of indexing/slicing operations is needed to capture our construction efficiently. There are two types of tensors in our implementation: real-valued feature tensors and integer-valued index tensors. Index tensors never undergo floating point computations but rather are used as indices to slice into the rows and columns of other tensors; they play the role of *pointers*. In this sense, our operator layer can be thought of as a careful application of pointer networks or structured learning (Vinyals et al., 2015), with structures prescribed by the mesh topology. The $\mathrm{MLP}_{\boldsymbol{\theta}}(\cdot)$ essentially bypasses and learns the combined steps of the quadrature points selection, bilinear form evaluation at quadrature points, and averaging by quadrature weights, which are variable and tweakable parts in finite element methods.

As mentioned earlier that $\mathbf{E}_{j,i}$ can store all features in triangle j including features correspond to the three directed edges of triangle j, these features are stored in an ordered fashion: $\mathbf{E}_{j,i}$ stacks features correspond to the directed edge $O(i, j)$, features correspond to the directed edge $P(i, j)$, and features correspond to the directed $S(i, j)$, in order. Here $O(i, j)$ refers to the directed edge in triangle j that is opposite to vertex i, and $P(i, j)$ and $S(i, j)$ refer to direct edges anticlockwisely preceding and succeeding to $O(i, j)$ within triangle j, respectively.

C.1 Relationship to Geometry Processing

Our construction above is by no means heuristic, but rather is reflective of typical operator constructions in geometry processing. Below we document how our model for $\mathbf{H}$ encapsulates previous fixed-operator representations of meshes, including those appearing in geometric deep learning. Beyond demonstrating the flexibility of our approach, these examples show how we can expect our architecture to learn operators that match the performance or outperform previous work prescribing a single operator for all tasks.

Dirac operator. The Dirac operator, used to build Surface Networks (Kostrikov et al., 2018), can be written in the form (1) by taking:

$$\mathbf{H}_{j,i} = -\frac{1}{2}\mathbf{e}_{ij}, \tag{5}$$

where $\mathbf{e}_{ij} \in \mathbb{R}^3$ is the directed edge vector opposite vertex i in triangle j. The Dirac operator is valued in the complex quaternions, explaining the right-hand side in $\mathbb{R}^3$.

Gradient operator. We can interpolate per-vertex features to the interiors of triangles linearly to obtain a function along the entire triangulated surface. This piecewise-linear interpolant—written in the "hat function" basis from the finite element method (FEM) (Brenner & Scott, 2007)—can be differentiated; since the interpolant is linear on each face, the gradient is a constant vector per triangle. Hence, the (x, y, z) components of the gradient each can be obtained using a linear operator that takes in a value per vertex and outputs vector per triangle

$$\mathbf{H}_{j,i} = \text{NORMALIZE}\left(\frac{\mathbf{e}_{ij,2} \times \mathbf{e}_{ij,3}}{\mathbf{e}_{ij,1}}\right), \tag{6}$$

where $\mathbf{e}_{ij,\ell}$ gives the edge vector opposite vertex i in triangle j ($\ell = 1$), its successor edge vector ($\ell = 2$), and its predecessor edge vector ($\ell = 3$). This operator appears in directional field design (Vaxman et al., 2016) and discretization of partial differential equations.

Cotangent Laplacian. The cotangent Laplacian (Pinkall & Polthier, 1993) is the best-known operator in geometry processing; Zeng et al. (2012) show that it encodes a shape up to rigid motion and global scaling. This $n \times n$ operator can be obtained as $\boldsymbol{\Delta} = \sum_{k=1}^{3} \mathbf{H}^{k\top}\mathbf{M}\mathbf{H}^k$, where $\mathbf{H}^k \in \mathbb{R}^{f\times n}$ denotes k-th dimension of the gradient operator (6). This example provides an operator that is captured by taking products of operators in our parameterized form for $\mathbf{H}^k$.

FEM operator. The finite element method (FEM) discretizes a bilinear form (i.e. linear operator) $H : (\mathcal{L}_2(\mathcal{M}), \mathcal{L}_2(\mathcal{M})) \to \mathbb{R}$ operating on functions over a surface $\mathcal{M}$ by representing them in a *weak* (integrated) fashion. Take $\{\phi_i\}_{i=1}^n$ and $\{\psi_j\}_{j=1}^f$ to be two bases for functions on the mesh, indexed by vertices and triangles, resp. Then, Galerkin FEM approximates

$$\mathbf{H}_{j,i} = \int H[\phi_i, \psi_j](\mathbf{x})\, d\mathbf{x}.$$

For our matrix dimensions, we can take the ϕ_i's to be piecewise-linear "hat" basis functions per vertex and the ψ_j's to be piecewise-constant functions per triangle.

D Future Work

This departure from conventional deep learning shows practical benefit in our experiments, but more importantly it suggests a breadth of research directions in geometric learning. Some directions are straightforward; we can incorporate additional geometric features, introduce additional layers, and train on larger datasets. Better interpreting of the learned operators can possibly inspire the design of new fixed operators for shape analysis. More speculative extensions may yield larger jumps in performance and interpretability. For instance, currently our receptive field is limited by the number of layers; each layer induces communication between a vertex and its neighboring elements. Classical methods employ sparse linear algebra (e.g. inverting the operator rather than applying it); incorporating sparse linear algebra into the deep learning pipeline using methods like (Rennich et al., 2014) would enable a full receptive field but requires challenging algorithmic developments incorporating operations like sparse factorization into back propagation. It may also be valuable to develop a stronger link between our operator construction and methods for constructing operators in geometry processing, e.g. FEM (Brenner & Scott, 2007) or discrete exterior calculus (DEC) (Hirani, 2003).

Combining insight from data with best practices from geometry processing holds potential to formulate well-posed techniques for geometric deep learning. By acknowledging the challenges specific to mesh data, we can develop theory and practice of geometric deep learning, articulating its relationship to and differences from 2D vision.