Graph Convolutional Networks for Learning Laplace-Beltrami Operators

Yingying Wu [1] **Roger Fu** [1] **Richard Peng** [2] **Qifeng Chen** [3]

Abstract

Recovering a high-level representation of geometric data is a fundamental goal in geometric modeling and computer graphics. In this paper, we introduce a data-driven approach to computing the spectrum of the Laplace-Beltrami operator of triangle meshes using graph convolutional networks. Specifically, we train graph convolutional networks on a large-scale dataset of synthetically generated triangle meshes, encoded with geometric data consisting of Voronoi areas, normalized edge lengths, and the Gauss map, to infer eigenvalues of 3D shapes. We attempt to address the ability of graph neural networks to capture global shape descriptors–including spectral information–that were previously inaccessible using existing methods from computer vision, and our paper exhibits promising signals suggesting that Laplace-Beltrami eigenvalues on discrete surfaces can be learned. Additionally, we perform ablation studies showing the addition of geometric data leads to improved accuracy.

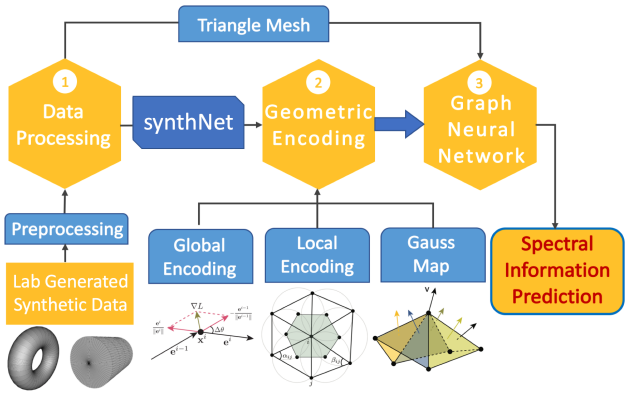


Figure 1. The overview of our approach: Our input is preprocessed synthetically generated datasets whose geometric data (Voronoi areas, normalized edge lengths, and Gauss map) are computed and encoded. We then use a graph neural network to predict spectral information from the triangle mesh and its geometric data.

1. Introduction

Identifying the shape of an object by geometric invariants is a critical task in a range of shape-analysis tasks such as segmentation, retrieval, parametrization, correspondence, and deformation. It also provides a basic model for understanding the properties of physical systems, including the heat equation that models how temperature diffuses over time. The spectrum (i.e., sequence of eigenvalues) of the Laplace–Beltrami operator of a surface in Euclidean space is isometry invariant, in the sense that it is not affected by rigid motions (translations and rotations), reflections, and changes in parametrization. Hence, it characterizes surfaces and solids, providing a global shape descriptor which efficiently encodes the information of geometric objects for retrieval and reconstruction purposes (Lévy, 2006).

Eigenvalue problems appear in several mathematical and geometric settings: the eigenvalues of the Laplace-Beltrami operator are connected to Ricci curvature and the heat kernel (Schoen & Yau, 1994), and furthermore, studying the connection between eigenvalues and geometric quantities such as the genus and the Cheeger constants is a fundamental problem in geometry (Yau, 2000). Furthermore, it is shown in (Reuter et al., 2006) that it is computationally feasible to extract certain geometric data, such as the volume, boundary length, and Euler characteristic, from the Laplace-Beltrami operator's spectrum. As the analytic solutions to eigenvalue problems are only known in certain special cases and typical require difficult analysis of systems of partial differential equations, numerical algorithms have become the dominant tool in discovering or verifying mathematical results involving eigenvalue problems. The computation of the eigenvalues of the Laplacian has a long history, going back to MacNeal (1949). The classical approach for computing the surface Laplacian involves computing the

[1]Department of Mathematics, University of Houston, Houston, USA [2]School of Computer Science, Carnegie Mellon University, Pittsburgh, USA [3]Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong, China. Correspondence to: Yingying Wu <ywu68@uh.edu>.

Proceedings of the Geometry-grounded Representation Learning and Generative Modeling at the 41st International Conference on Machine Learning, Vienna, Austria. PMLR Vol Number, 2024.
Copyright 2024 by the author(s).

cotangent formula of a triangle mesh and then deriving the eigenvalues of the matrix representing the cotangent formula, which involves a computationally intensive iterative numerical solver.

An additional difficulty is that the eigenvalues of the Laplace–Beltrami operator of a surface typically exhibit a large dynamic range and the magnitude of the spectrum may vary significantly from surface to surface, even when the triangle mesh is normalized to fit into a unit sphere. Furthermore, quantifying the accuracy of estimates of eigenvalues is also challenging since a good metric should be scaling-invariant, as rescaling a shape by a factor of s results in its eigenvalues being rescaled by a factor of $1/s^2$ (Reuter et al., 2009). Finally, surface reconstruction is carried out using the top K eigenvalues (Lévy, 2006), which are the smallest eigenvalues. This suggests a metric should be more sensitive to inaccuracies in smaller eigenvalues, thus further discouraging the use of metrics based on absolute measures of accuracy. For these reasons, the classical ℓ_p norms are not very effective, suggesting the need for an alternative approach.

In this article, we explore the learnability of spectral information from pre-existing and synthetically generated triangle mesh datasets using a graph neural network. Our approach is illustrated in Figure 1: we first generate a synthetic dataset, then perform geometric encoding before finally predicting spectral information through the use of a Graph Convolutional Network (GCN).

Preliminary studies suggest that the graph convolutional network is more efficient than classical algorithms when tested on an NVIDIA A100 GPU with 256GB memory and an AMD EPYC 9654 CPU with 1.5TB of memory. We start with 14,726 synthetic meshes and split them up into batches of 200 meshes each (there will be 1 batch that has < 200 meshes). The GCN receives all of the meshes in each batch simultaneously as input and predicts the first 10 eigenvalues for every mesh in parallel. This approach takes on average 0.1 seconds ($\sigma = 0.013$) to predict a batch of 200 meshes in parallel on GPU and 1.0 seconds ($\sigma = 0.081$) to predict each batch of 200 meshes in parallel on CPU. In comparison, the iterative method (Lanczos algorithm) on the same set of 14,726 meshes (without batch split) takes on average 2.5 seconds ($\sigma = 0.35$) on each mesh on CPU.

The main contribution of this paper is the introduction of encoded geometric data, specifically the encoding of Voronoi areas, normalized edge lengths, and the Gauss map, specifically, the weighted average of the face normals. Using the Peak-Signal-to-Noise Ratio (PSNR), which measures the ratio between the maximum possible power of a signal and the power of noise, we consider a prediction with PSNR > 20 as accurate (see Appendix B for an illustration of the PSNR metric). Encoding geometric data gives an improvement to 86.84% of meshes with PSNR > 20 compared to 46% of meshes with no geometric data encoded. More details can be found in Table 2.

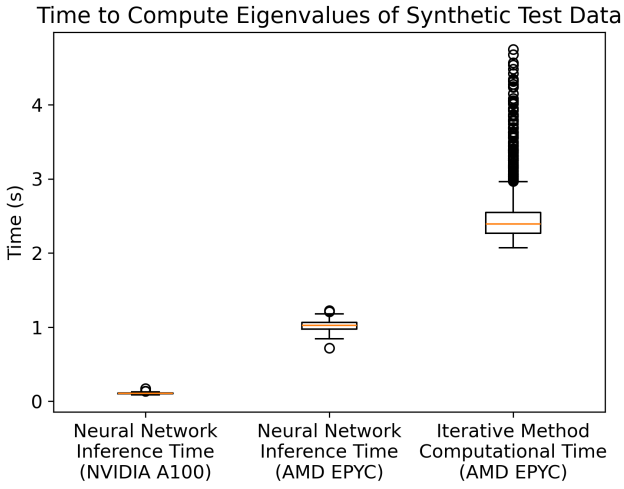


Figure 2. The average time taken to predict the first 10 eigenvalues of a batch of 200 meshes via the GCN is 0.11 seconds ($\sigma = 0.013$) on an NVIDIA A100 with 256GB memory (NVIDIA A100), and 1.0 seconds ($\sigma = 0.081$) on an AMD EPYC 9654 CPU with 1.5TB memory (AMD EPYC), while the average time taken to compute the first 10 eigenvalues for one mesh via LaPy is 2.5 seconds ($\sigma = 0.35$) on an AMD EPYC.

We introduce a new evaluation metric, Inverse Laplacian, designed specifically for spectral tasks, in addition to introducing novel loss functions PolarLoss and RPDLoss that are more stable than ℓ_1 loss for spectral problems. Additionally, we show empirically that RPDLoss is more effective than ℓ_1 loss for learning spectral information. Finally, we provide the large-scale refined triangle mesh synthetic dataset synthNet used for our experiments, for further research in geometric processing at `https://github.com/eigenGCN/synthnet`.

2. Related Work

2.1. Deep Learning Methods in Geometric Characterization

Convolutional Neural Networks (CNNs) have achieved strong performance when it comes to feature representation, including local and global features in 2D images for low-level and high-level image tasks (Krizhevsky et al., 2017). Unlike 2D images, extracting features by directly applying CNNs on 2D surfaces in $\mathbb{R}^3$ is inconvenient due to its non-Euclidean geometry and memory consumption. 2D surfaces are typically processed as triangle meshes or point clouds. Mesh-based implementations represent geometric and topological properties by vertices, edges, and surfaces, which can be more robust to non-rigid deformation.

Mesh-based networks aim to build convolutional and pooling layers directly on meshes (Hanocka et al., 2019; Sharp et al., 2022) or geodesic polar coordinates (Masci et al., 2015). Boscaini introduced a CNN architecture based on anisotropic diffusion kernels, which does not rely on triangle meshes only (2016). GCNs (Milano et al., 2020) treat meshes as graph-structured data and design variants of GCN directly on the graphs.

Most deep learning methods described above are designed for specific high- or low-level vision tasks (e.g., shape classification, semantic segmentation, shape retrieval) or for extracting local shape descriptors in a task-driven mode (Leng et al., 2015; Guo et al., 2020). Intrinsic global geometric shape descriptors such as the spectrum of the Laplacian-Beltrami operator and curvatures (Larios-Cárdenas & Gibou, 2021) are rarely discovered by deep learning methods, despite their importance for shape analysis. Learning intrinsic shape features is challenging since they are hard to implement as an end-to-end model using existing deep learning architectures. Additionally, the intrinsic geometric shape features are non-trivial to evaluate quantitatively. Learning-based methods (Litman & Bronstein, 2013) have been employed to generate shape descriptors mostly using synthetic and simple scanned shapes. Learning global geometric shape descriptors from real-world complex shapes is more challenging. Marin et al. (2021) compute a shape descriptor by focusing on the smoothness and orthogonality properties of the Laplace-Beltrami eigenvectors, and perform experiments on higher-dimensional embeddings with loss functions that optimize one or more of: orthogonality, sparsity, and the Dirichlet energy, which is a condition equivalent to smoothness in this context.

2.2. Computing the Laplacian

Computing the surface Laplacian can be traced back to MacNeal (1949). Notable later works include Reuter et al. (2006) on the spectrum of the Laplace-Beltrami operator as a numerical signature of a 2D or 3D manifold; Hamidian et al. (2016) presented a novel surface registration technique by mapping eigenvalues and eigenvectors of the Laplace-Beltrami of the shapes through optimizing an energy function; Wu et al. (2022) computed the eigenvalues of the Laplacian of a surface represented by a point cloud, and computed the eigenvalues of the Laplacian on a surface by computing the graph Laplacian of the ϵ-neighborhood of the surface in a lattice. Also working with point clouds, Marin et al. (2020) use an autoencoder-based method to learn the relationship between point clouds and the spectrum of the Laplace-Beltrami operator, allowing them to predict the eigenvalues of point clouds. A thorough review of computational methods for the analysis of the Laplacian is found in Solomon et al. (2014) and Crane et al. (2013).

The main drawback of the widely used cotangent formula-based approaches involves the time-consuming process of solving for eigenvalues of the constructed matrix. While constructing the sparse cotangent matrix for a mesh with n vertices and m edges can be done in $\mathcal{O}(n+m)$, the issue lies in solving for the eigenvalues of this matrix. In particular, the process of computing the eigenvalues of said cotangent matrix reduces to matrix multiplication if viewed as a general diagonalization problem (Banks et al., 2022). Thus, the complexity of the cotangent formula-based classical approach is around $O(n^\omega)$, where ω is the matrix multiplication constant, which at the time of writing is $\omega < 2.373$ (Le Gall, 2014).

The spectrum of the Laplace-Beltrami operator typically includes a very wide range of numerical values, as well as among the spectrum of different shapes. This variability makes the training of a learning-based algorithm more difficult. In this paper, we address these challenges using a graph-based neural network by encoding geometric data, alongside novel loss functions and evaluation metrics.

3. Methodology

3.1. Problem Formulation

Suppose $(\mathcal{M}, g)$ is a compact smooth Riemannian surface embedded in $\mathbb{R}^3$, possibly with boundary, and $\mathcal{L}_M$ is the Laplace-Beltrami operator on $\mathcal{M}$. We consider the eigenvalue problem of finding $(\lambda, f) \in \mathbb{R} \times C^2(\mathcal{M})$ such that

$$\mathcal{L}_M f = -\lambda f$$

where f satisfies the Neumann boundary condition if $\mathcal{M}$ has boundary. Computing eigenvalues of surfaces is closely related to the construction of discrete Laplacians. Given a piecewise Euclidean triangle mesh (V, E, F), a Laplace-Beltrami operator L is often represented as

$$(Lf)_i = \frac{1}{A_i} \sum_{j:e_{ij} \in E} w_{ij}(f_j - f_i)$$

with edge weights

$$w_{ij} = \frac{\cot \alpha_{ij} + \cot \beta_{ij}}{2},$$

where α_{ij} and β_{ij} are the two inner angles in T that face the edge e_{ij} and A_i is the Voronoi area normalization. In this study, we focus on using a graph neural network to infer the eigenvalues of the Laplace-Beltrami operator L on a compact smooth Riemannian surface embedded in $\mathbb{R}^3$ represented by a triangle mesh. We adopt the graph structure of triangle meshes and encode the coordinates in $\mathbb{R}^3$ of each vertex as 3 channels.

3.2. Geometric encoding

In this section, we propose geometric encoding schemes based on Riemannian geometry to augment the extracted surface information. In particular, we propose encoding the (mixed) Voronoi areas, normalized edge lengths, and a discrete analogue of the Gauss map.

Edge length encoding. Given a piecewise Euclidean triangle mesh (V, E, F), if $e_{ij} \in E$ is an edge with endpoints $v_i, v_j \in V$, we define the length of e_{ij} to be $\|v_i - v_j\|_2$, where

$$\|(x_1, x_2, x_3)\|_2 = \sqrt{x_1^2 + x_2^2 + x_3^2}$$

is the standard Euclidean norm. We can then assign e_{ij} its normalized length as the edge weight, i.e.,

$$\frac{\|v_i - v_j\|_2}{\max_{e_{ij} \in E} \|v_i - v_j\|_2}.$$

Voronoi area. We compute the mixed Voronoi cell area via the libigl library (Jacobson et al., 2018). Let v be a vertex, and F a triangular face incident to v whose other two vertices are x and y. If F is not an obtuse triangle, we define the Voronoi area of v in F to be

$$\frac{1}{8}\left(\|x - v\|_2^2 \cot \alpha + \|y - v\|_2^2 \cot \beta\right)$$

where α and β are the angles of F at x and y. If F is an obtuse triangle, we define the Voronoi area of v in F to be $\frac{\text{area}(F)}{2}$ if the obtuse angle is at vertex v and $\frac{\text{area}(F)}{4}$ otherwise where $\text{area}(F)$ denotes the area of the triangle F. Finally, the Voronoi area for v is given by the sum of the Voronoi areas of v in F for all F incident to v (Meyer et al., 2002).

Gauss Map. The Gauss map is a crucial tool in differential geometry that endows desirable properties. The Gaussian curvature K at a point on a surface is defined by the determinant of the differential of the Gauss map at that point:

$$K = \det(dG_p)$$

This relationship allows the Gaussian curvature to be interpreted as the area distortion factor by which the Gauss map stretches or compresses infinitesimal areas around a point when mapped onto the unit sphere. The degree of the Gauss map is related to the total curvature of the surface through the Gauss-Bonnet theorem (Gauss, 1827; Bonnet, 1853; Chern, 1944):

$$\int_S K \, dA = 2\pi\chi(S)$$

where K is the Gaussian curvature and dA is the area element on the surface. The differential of the Gauss map, dG_p, is closely connected to the shape operator S of the surface at point p:

$$S_p = -dG_p.$$

All these factors provide a strong motivation for encoding the Gauss map in surface characterization, and our experimental results support our argument (see 4.3.3).

Formally, the Gauss map is defined as the following: Let $S \subset \mathbb{R}^3$ be a smooth, orientable surface. The Gauss map, denoted as G, is a function defined as follows:

$$\begin{aligned} G : S &\to \mathbb{S}^2 \\ p &\mapsto N(p) \end{aligned}$$

where:

- p is a point on the surface S,
- $N(p)$ is the unit normal vector at p,
- $\mathbb{S}^2$ represents the unit sphere in $\mathbb{R}^3$.

We propose the area-weighted Gauss map for triangle meshes as follows: if $v \in V$ is incident to faces $F_1, \ldots, F_k \in F$ with areas $\text{area}(F_1), \ldots, \text{area}(F_k)$ and normal vectors $N_1, \ldots, N_K$ oriented outwards, then we assign v the normal vector

$$N_v = \frac{\sum_{i=1}^k \text{area}(F_i) N_i}{\sum_{i=1}^k \text{area}(F_i)}$$

encoded as a vector in $\mathbb{R}^3$, at each vertex. This is illustrated by Figure 3.

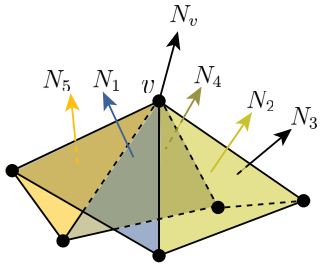


Figure 3. Area-weighted Gauss map for triangle meshes. For a given vertex v, we take the weighted average of the normal vectors of the faces incident to v.

3.3. Models

Spatial surfaces can be represented as triangle meshes, on which we can train graph neural networks. Since GCNs process directly on the spectral domain, they are a natural choice for computing the spectrum of the Laplace-Beltrami operator. These networks preserve topological and geometric properties. An illustration of our models is displayed in Figure 4.

3.3.1. Graph Convolution Network for Mesh

Graphs can represent an object in low dimensions while preserving geometric and topological properties. For surfaces represented by triangle meshes, we can construct an undirected graph $G = (V, E, A)$, where V and E are the sets of vertices and edges with $|V| = N$ and $A \in \mathbb{R}^{N\times N}$ is the weighted adjacency matrix denoting the weights of the edges. The essential operator in GCNs is the graph Laplacian (Chung, 1997), which is defined in the normalized form as $L = I_n - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$, where $D \in \mathbb{R}^{n\times n}$ is a diagonal degree matrix with $D_{ii} = \sum_j A_{ij}$. Since the eigenvalues of L represent the graph's frequencies and contain essential information, the convolution of the graph is generally applied to the spectral rather than on the vertex domain. We apply a GCN as follows:

Graph convolutional layer: The convolutional kernel of a signal $X \in \mathbb{R}^{N\times C}$ in GCN (Kipf & Welling, 2016) is

$$X' = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}XW),$$

where $W \in \mathbb{R}^{C\times m}$ is a matrix of convolutional kernel parameters, $X' \in \mathbb{R}^{N\times m}$ is the convolved signals, and σ is the activation function. Since convolutional layers of the graph will smooth out the signals (Nt & Maehara, 2019; Li et al., 2018; Xu et al., 2018), the depth of GCN models is limited. Thus, for the spectral problem of the Laplace-Beltrami Operator, the architecture of the network is given by three graph blocks with 64, 128, and 256 hidden channels, respectively. Each graph block is composed of a GraphConv layer, followed by a LeakyReLU, and then a Linear layer, in that order.

Multi-Layer Perception (MLP): After 3 graph convolution layers, we use a 5-layer MLP, which can aggregate features extracted by previous graph convolution layers. Since our task is to predict the smallest K eigenvalues, the number of output channels of the MLP layer is K as the expected number of eigenvalues.

3.4. Novel Loss Functions

One major challenge in learning the spectrum of Laplace-Beltrami operators is the selection of the loss function. ℓ_1 loss and MSE loss perform poorly for the eigenvalue problem since, for each object, the eigenvalues can take on a wide range of magnitudes. For instance, in the Thingi10K dataset (Zhou & Jacobson, 2016), the maximum value of the 10th eigenvalue can be as large as 2915.95, and the minimum can be as small as -2.89×10^{-16}. Furthermore, loss functions like the ℓ_1 and MSE loss can achieve a local minimum by focusing on accurately predicting the larger eigenvalues, while neglecting the smaller ones. However, the smaller eigenvalues are the ones that contribute more to the global shape appearance (Cosmo et al., 2019), which suggests they are unsuitable loss functions. Previous work (Cosmo et al., 2019; Rampini et al., 2019) overcame this deficiency by using weighted ℓ_2 norms, reweighing the ith difference by either $\frac{1}{i}$, or $\frac{1}{\mu_i}$, where μ_i is the ith eigenvalue of the Hamiltonian operator.

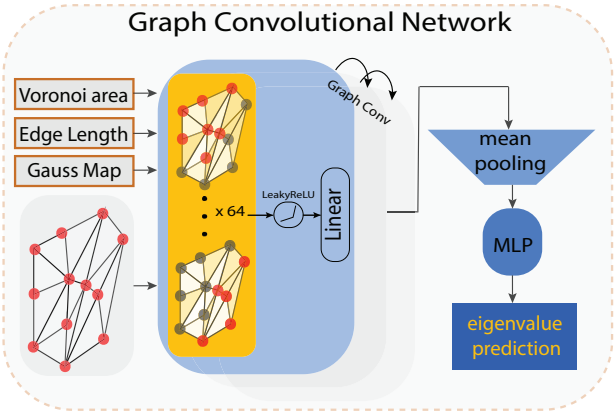


Figure 4. A schematic depiction of our GCN architecture for spectral inference with geometric encoding. The GCN takes as input a triangle mesh and geometric encoding, including the Voronoi area, normalized edge lengths, and Gauss map. The GCN then passes the mesh and geometric encoding through 3 graph convolutional layers, each consisting of a GraphConv, LeakyReLU, and a Linear layer, followed by a mean pooling layer and an MLP layer.

To address this challenge, we propose to use the Relative Percent Difference Loss (RPDL) and introduce a novel loss function PolarLoss (PL).

PolarLoss is motivated by the both cosine similarity (Nguyen & Bai, 2011) and normalized error. Like polar coordinates in 2D, where every point is specified by a distance from the origin (norm) and angle, PolarLoss measures the difference in both the angle and norm between two vectors.

$$\mathrm{PL}(\vec{p}, \vec{g}) = 1 - \langle \hat{p}, \hat{g} \rangle + \frac{\|\vec{p} - \vec{g}\|_2}{\|\vec{p}\|_2 + \|\vec{g}\|_2}, \hat{p} = \frac{\vec{p}}{\|\vec{p}\|_2}, \hat{g} = \frac{\vec{g}}{\|\vec{g}\|_2}.$$

Let $\vec{p} = \{p_1, \ldots, p_{10}\}$ and $\vec{g} = \{g_1, \ldots, g_{10}\}$ be vectors corresponding to the first 10 predictions and labels, respectively. The first term $1 - \langle \hat{p}, \hat{g} \rangle$ can be interpreted as the cosine similarity between the two vectors. Since we know the vector of an object's first 10 eigenvalues will be in ascending order, the predicted eigenvalues should also have the same trend. The cosine similarity term is sensitive to every eigenvalue regardless of its magnitude. Hence, the large eigenvalues will not dominate the loss. However, solely capturing the trend is inadequate as it fails to account for potential discrepancies among large eigenvalues, despite their similar trends. To address this limitation, the addition of the second term accounts for the normalized difference between

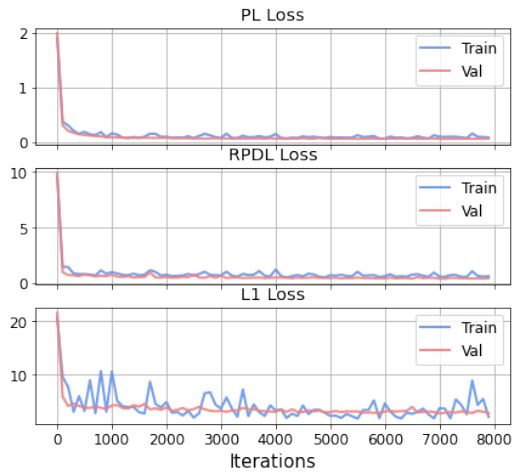


Figure 5. Comparison of PolarLoss (PL), Relative Percent Difference Loss (RPDL), and ℓ_1 loss functions. The graphs show that PolarLoss and RPDLoss are more stable than ℓ_1 loss.

eigenvalues. This term effectively corrects for any discrepancies introduced by the large eigenvalues. The first term will be in $[0, 2]$, and the second term will be in $[0, 1]$. Hence, for every object, the loss will be of the same magnitude.

Relative Percent Difference Loss is another choice of loss function for the situation where the magnitude of different eigenvalues of an object has small variation. As in the synthetic data, there is rarely a jump between consecutive eigenvalues. Define

$$\text{RPDL} = \sum_{i}^{10} \frac{|p_i - g_i|}{|p_i| + |g_i| + \varepsilon}.$$

where $p_1, \ldots, p_N$ are the predictions, $g_1, \ldots, g_N$ the ground truth, and ε is a small non-zero constant to prevent division by 0. Relative loss measures the ratio of the difference between predicted and real value. Here, for the first K eigenvalue prediction, we calculate the relative losses of each eigenvalue and calculate the ℓ_1 norm of the relative error of the predicted spectrum, which is commonly used in numerical analysis. Then we compare our proposed Polar Loss and RPDL loss with ℓ_1 loss in Figure 5. The figure shows that PolarLoss and RPDL loss are significantly more stable than ℓ_1 loss since they are not sensitive to the magnitude when the size of the eigenvalues has a large variation and they handle all eigenvalues equally.

3.5. Inverse Laplacian: A Novel Evaluation Metric

When computing the ℓ_2 norm between the predicted eigenvalues $\vec{p}$ and the ground truth eigenvalues $\vec{g}$, small eigenvalues have little influence on the difference, while large eigenvalues have great influence. Nevertheless, the effect of each eigenvalue should contribute equally regardless of its magnitude (Cosmo et al., 2019). As such, we propose the new evaluation metric Inverse Laplacian:

$$\text{InvLap} = \sqrt{\sum_{i}^{10} \left(\frac{\frac{1}{p_i} - \frac{1}{g_i}}{\frac{1}{g_i}} \right)^2} = \sqrt{\sum_{i}^{10} \left(\frac{g_i}{p_i} - 1 \right)^2}$$

This metric measures the difference between the predicted value and the ground truth with the effect caused by the magnitude of the eigenvalue canceling out each other on the magnitude. Each summand is the relative error of reciprocals. If $g_i \neq 0$ and $\frac{g_i}{p_i} - 1 = \varepsilon$, then $\frac{p_i - g_i}{g_i} = \frac{\varepsilon}{1+\varepsilon}$ (see Appendix A). This implies that each term of the summation behaves like a negative reciprocal of relative error shifted by 1 on the denominator, i.e., the Inverse Laplacian appears to be bigger than relative error, but penalizes more heavily the prediction of small eigenvalues. The behaviour of the Inverse Laplacian converges to relative error as the prediction p approaches the ground truth g.

4. Experiments

4.1. Dataset

For the synthetic dataset, we generated 16,550 triangle meshes. We computed the eigenvalues of these triangle meshes using the LaPy solver (Reuter et al., 2006; Wachinger et al., 2015) to construct the LBO matrix and compute its first 10 eigenvalues. The LaPy solver calls the eigsh method of scipy (Virtanen et al., 2020), which invokes an implementation of the Implicitly Restarted Lanczos Method (Lehoucq et al., 1998). We then filtered out any meshes whose eigenvalues have numerical abnormalities, yielding 16,361 meshes. The breakdown of the original 16,550 meshes is as follows. For further details, see Appendix C.

Standard shapes. We generate triangle meshes for 2,000 rectangular prisms, 1,000 cones, 1,000 cylinders, 50 spheres, and 2,000 tori using Open3D's built-in methods (Zhou et al., 2018) in addition to manually generating 2,000 tetrahedra and 1,000 conical frustums.

Split shapes and inverted cones. To ensure our proposed model are learning the underlying geometry of triangle meshes, as opposed to graph connectivity information, we generate 2,500 split spheres and 1,000 split cones, in addition to 4,000 inverted cones.

Data augmentation. To improve training, we augment the data set by performing rotations and reflections. Specifically, for each triangle mesh, we generate two rotations and two reflections of the mesh, and then 4 instances of the mesh under both rotations and reflections, in addition to the original mesh. The rotations and reflections are randomly generated using the trimesh library (Dawson-Haggerty et al.).

4.2. Training

Datasets were divided into training, validation, and test sets (70:20:10 split). For Table 1 and Table 4, all models are trained for 200 epochs with the SGD optimizer (lr=10^{-3}, momentum=0.9, weight decay=10^{-5}) and batch size 32.

To perform the geometric data comparison in Table 2 and Table 3, we train for 400 epochs with the SGD optimizer (lr=10^{-4}, momentum=0.9, weight decay=10^{-5}) and batch size 16.

4.3. Results

In this section, we present our experimental results on training a graph convolutional network to predict eigenvalues on synthetic data, including a shape-by-shape breakdown. We also compare the effects encoding geometric data has on the accuracy of predictions. We define accuracy using PSNR (Hore & Ziou, 2010). PSNR is computed by the formula

$$\mathrm{PSNR}(I, J) = 10 \log_{10} \left(\frac{\mathrm{MAX}_I^2}{\mathrm{MSE}} \right)$$

where $I, J \in \mathbb{R}^n$, $I = (I_1, \ldots, I_n)$ and $J = (J_1, \ldots, J_n)$ corresponding to the ground truth and predictions, respectively. For each ground truth vector I, we define

$$\mathrm{MAX}_I = \max_{i=1,\ldots,n} I_i - \min_{j=1,\ldots,n} I_j$$

as the range of the vector I, and $\mathrm{MSE} = \frac{1}{n} \sum_{i=1}^{n} (I_i - J_i)^2$ is the mean squared error. We find empirically that if PSNR > 20, the prediction is reasonably accurate and closely matches the ground truth. Hence, we consider such predictions to be correct. We include a selection of representative examples along with their respective evaluation metrics in Appendix B.

4.3.1. RPDL Loss Function

We conducted experiments on the augmented synthetic data using our GCN trained with PolarLoss, RPDL, and ℓ_1 loss without geometric encoding. Experimental results on the synthetic data are displayed in Table 1, where we find the model trained by RPDLoss outperforms the models trained by PolarLoss and ℓ_1 loss under the Inverse Laplacian metric: RPDLoss achieves 0.2816, smaller than ℓ_1's 0.3716 and PolarLoss's 0.9458. Similarly for the PSNR > 20 metric, RPDLoss achieves 84.5%, better than ℓ_1's 77.2% and PolarLoss's 54.3%.

Our experiments show that the GCN is capable of predicting the eigenvalues of the Laplace-Beltrami operator on the augmented synthetic dataset, and that the RPDLoss function achives the best performance with under the Inverse Laplacian, PolarLoss, RPD, and PSNR metrics. Furthermore, as can be seen in Table 4, the model trained with RPDLoss outperforms the models trained with PolarLoss and ℓ_1 respectively on 8/10 types of shapes, only outperformed by the model trained with ℓ_1 loss on the truncated cone and tetrahedron. As such, we use the RPDLoss function as a baseline when comparing the effects of geometric encoding on accuracy.

4.3.2. Learning Underlying Geometries

Table 4 gives a breakdown of the accuracy of the model on each class of shape. Of particular interest is that the GCN trained with RPDLoss is capable of recognizing shapes with similar geometries.

The dataset includes two types of triangle meshes, named split spheres and spheres, both of which correspond to spheres. Thus, if the model is learning geometric, as opposed to connectivity information, it should have comparable performance on the two types of meshes. Table 4 shows that the model trained with RPDLoss achieves Inverse Laplacian 0.0319 on spheres and 0.0309 on split spheres. These are comparable, suggesting the model doesn't have particular difficulty learning one over the other. Similarly, split cones and cones also encode the same underlying surface, and there too the model trained with RPDLoss has comparable performance on both, with Inverse Laplacian 0.1735 on the former and 0.2204 on the latter. This suggests that the model is using more than just the connectivity of the underlying mesh to determine how to compute the eigenvalues, and as such is learning the geometry of the underlying mesh.

4.3.3. Comparisons with Geometric Encoding

We perform comparisons among GCN models trained with RPDLoss, comparing the effect of encoding the Voronoi area, edge lengths, Gauss map, and adding a virtual node, which is a special vertex that is connected to all other vertices in the graph, as introduced by Gilmer et al. (2017). Incorporating a virtual node allows for the convolution operators to propagate information between vertices that would otherwise be apart in the graph.

From Table 2, we observe that encoding geometric data (area, Gauss map, distances) improves the performance of the 3-layer model: the number of meshes with PSNR > 20 increases from 46% with no geometric data to 72.98% when area is encoded to 86.84% when area, Gauss map, and the normalized edge lengths are encoded.

Additionally, taking the 3-layer model as our baseline, we find that encoding more geometric information (area, normalized edge lengths, Gauss map) improves the accuracy more than increasing the number of layers to 4 or 5. This can be seen by Table 3: the 3-layer model without virtual node that encodes area, distances, and the Gauss map has 80.9%

Table 1. Experiment results on synthetic data with different evaluation metrics. The table shows the model trained with RPDLoss as the loss function outperforms the models trained with PolarLoss and ℓ_1 loss in all metrics other than ℓ_1 and ℓ_2.

Model	Inverse Laplacian	PolarLoss	RPD	PSNR > 20	ℓ_1 Norm	ℓ_2 Norm
GCN_{PL}	0.9458	0.0985	0.985	54.3%	83.40	33.91
GCN_{RPDL}	**0.2816**	**0.0370**	**0.3378**	**84.5%**	36.59	16.66
GCN_{ℓ_1}	0.3716	0.0463	0.4515	77.2%	**31.01**	**14.20**

$\mathbf{GCN_{PL}}$: PolarLoss as loss function, $\mathbf{GCN_{RPDL}}$: RPDLoss as loss function, $\mathbf{GCN}_{\ell_1}$: ℓ_1 loss.

Table 2. Comparison of models with different amounts of geometric information encoded. The table shows that encoding the area, edge lengths, and Gauss map outperforms the model that has access to no encoded geometric data, and the model where only the Voronoi area is encoded in all metrics, with or without the presence of a virtual node (a special vertex added to the graph to connect all vertices).

Geometric Encoding	Virtual Node	PSNR > 20	PolarLoss	RPD	Inverse Laplacian	ℓ_1 Norm	ℓ_2 Norm
None	Yes	46.00%	1.98×10^{0}	1.87×10^{0}	3.64×10^{0}	1.25×10^{2}	4.75×10^{1}
A	No	73.79%	6.00×10^{-2}	4.98×10^{-1}	4.98×10^{-1}	5.81×10^{1}	2.57×10^{1}
A	Yes	72.98%	6.29×10^{-2}	5.29×10^{-1}	5.14×10^{-1}	5.96×10^{1}	2.62×10^{1}
ALG	No	80.90%	3.92×10^{-2}	4.00×10^{-1}	2.77×10^{-1}	$\mathbf{4.04 \times 10^{1}}$	$\mathbf{1.62 \times 10^{1}}$
ALG	Yes	**86.84%**	$\mathbf{3.70 \times 10^{-2}}$	$\mathbf{3.88 \times 10^{-1}}$	$\mathbf{2.46 \times 10^{-1}}$	5.35×10^{1}	2.06×10^{1}

A: area, **L**: edge length, **G**: Gauss map.

Table 3. Comparison of the effectiveness of adding more layers. The table shows that the highest accuracy is achieved by the 3 layer model with more geometric encoding, rather than by increasing the number of layers to 4 or 5.

Virtual Node	Layers	Geometric Encoding	PSNR > 20	PolarLoss	RPD	Inverse Laplacian	ℓ_1 Norm	ℓ_2 Norm
No	3	ALG	**80.90%**	$\mathbf{3.92 \times 10^{-2}}$	$\mathbf{4.00 \times 10^{-1}}$	$\mathbf{2.77 \times 10^{-1}}$	$\underline{\mathbf{4.04 \times 10^{1}}}$	$\underline{\mathbf{1.62 \times 10^{1}}}$
	3	A	73.79%	5.29×10^{-1}	5.29×10^{-1}	4.98×10^{-1}	5.81×10^{1}	2.57×10^{1}
	4	A	79.90%	5.19×10^{-2}	4.38×10^{-1}	4.33×10^{-1}	5.39×10^{1}	2.41×10^{1}
	5	A	78.31%	5.11×10^{-2}	4.38×10^{-1}	4.08×10^{-1}	4.92×10^{1}	2.23×10^{1}
Yes	3	ALG	$\underline{\mathbf{86.84}\%}$	$\underline{\mathbf{3.70 \times 10^{-2}}}$	$\underline{\mathbf{3.88 \times 10^{-1}}}$	$\underline{\mathbf{2.46 \times 10^{-1}}}$	5.35×10^{1}	$\mathbf{2.06 \times 10^{1}}$
	3	A	72.98%	6.29×10^{-2}	5.29×10^{-1}	5.14×10^{-1}	5.96×10^{1}	2.62×10^{1}
	4	A	78.93%	5.15×10^{-2}	4.42×10^{-1}	4.20×10^{-1}	$\mathbf{5.24 \times 10^{1}}$	2.35×10^{1}
	5	A	67.14%	8.32×10^{-2}	6.74×10^{-1}	6.44×10^{-1}	6.76×10^{1}	2.89×10^{1}

A: area, **L**: edge length, **G**: Gauss map.

Table 4. Experiment results on different geometric objects using Inverse Laplacian. The table shows that RPDLoss gives the most accurate predictions for all shapes other than truncated cones and tetrahedra, where it only performs slightly worse than ℓ_1 loss.

Model	Box	Cone	Truncated Cone	Cylinder	Sphere
GCN_{PL}	0.9597	0.4764	1.451	0.8435	0.3391
GCN_{RPDL}	**0.3971**	**0.1735**	0.8993	**0.4035**	**0.0319**
GCN_{ℓ_1}	0.6287	0.3151	**0.7157**	0.4945	0.0753

Model	Tetrahedron	Torus	Split Cone	Inverted Cone	Split Sphere
GCN_{PL}	3.092	0.8795	0.5591	0.4671	0.1669
GCN_{RPDL}	0.5925	**0.1291**	**0.2204**	**0.2282**	**0.0309**
GCN_{ℓ_1}	**0.5498**	0.3212	0.3642	0.3540	0.0753

$\mathbf{GCN_{PL}}$: PolarLoss as loss function, $\mathbf{GCN_{RPDL}}$: RPDLoss as loss function, $\mathbf{GCN}_{\ell_1}$: ℓ_1 loss.

of meshes with PSNR > 20, compared to 79.9% and 78.31% for 4 and 5 layer models that encode only area, respectively. The case with a virtual node, the 3-layer mode that encodes area, normalize edge lengths, and Gauss map has 86.84% of meshes with PSNR > 20, compared to 78.93% and 67.14% for the 4- and 5-layer models that encode only area, respectively.

4.3.4. Comparisons with More Layers

We also observe the effect of adding more layers. It follows from Table 3, that in the case where there is a virtual node, using 5 layers results worse accuracy than three and four layers, performing worse in the PSNR, PolarLoss, RPD, and Inverse Laplacian metrics.

We also compare the result of adding more layers in the case when the graph is not augmented with a virtual node. Comparing the PSNR and PolarLoss metrics suggests that using 4 layers is competitive with using 5 layers.

5. Conclusion

In this study, we explore the learnability of the Laplace-Beltrami operator of triangle meshes through graph neural networks, and investigate the effectiveness of encoding geometric data for a synthetic dataset. We find that even without encoding geometric data, the network is capable of predicting eigenvalues, with over 80% of meshes achieving PSNR > 20. Furthermore, the addition of geometric data provides significant improvements to accuracy, increasing the accuracy by upwards of 10%. This suggests that the encoded geometric data is playing a non-trivial role in the prediction of the eigenvalues. Our result also shows the effectiveness of our novel loss function RPDLoss: When trained with RPDLoss, 7% more meshes have PSNR > 20 compared to training with ℓ_1 loss, and it performs better than the ℓ_1 trained model on the Inverse Laplacian metric on all but two shapes (truncated cones and tetrahedra). This suggests that RPDLoss is more effective than ℓ_1 loss for spectral problems, as it accounts for relative rather than absolute error.

Future work will be dedicated to predicting the eigenvalues of point clouds and generalizing the geometric encoding to point clouds. This would allow for the loss functions, evaluation metrics, and geometric encoded to be tested on a related problem, while generalizing their use beyond just point clouds. Furthermore, this would allow for a comparison with the autoencoder-based methodology proposed in Marin et al (2020), where they train an autoencoder and inevitable module simultaneously such that the module learns the association between the spectrum of the LBO operator and latent vector.

Limitations. In this study, we primarily focused on learnability and studying the effect encoding geometric data has on spectral prediction. As such, our results have yet to be improved upon, both on a discrete scale such as PSNR, and continuous scales, such as Inverse Laplacian. Additionally, our dataset contains only meshes of genus 0 and 1, and no real-world meshes.

Acknowledgements

The authors would like to thank Cliff Taubes, Etienne Vouga, and Justin Solomon for their input. This work used the Anvil system at Purdue University and the Delta system at the National Center for Supercomputing Applictions through allocation MTH230008 from the Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (ACCESS) program, which is supported by National Science Foundation grants #2138259, #2138286, #2138307, #2137603, and #2138296.

Richard Peng was partially supported by NSF CAREER Award CCF-2330255, and the Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant RGPIN-2022-03207.

References

Banks, J., Garza-Vargas, J., Kulkarni, A., and Srivastava, N. Pseudospectral shattering, the sign function, and diagonalization in nearly matrix multiplication time. *Foundations of Computational Mathematics*, pp. 1–89, 2022.

Bonnet, P. O. Mémoire sur la théorie générale des surfaces. *Journal de l'École Polytechnique*, 19:1–146, 1853.

Boscaini, D., Masci, J., Rodolà, E., and Bronstein, M. Learning shape correspondence with anisotropic convolutional neural networks. *Advances in neural information processing systems*, 29, 2016.

Chern, S.-S. A simple intrinsic proof of the Gauss-Bonnet formula for closed riemannian manifolds. *Annals of Mathematics*, 45(4):747–752, 1944.

Chung, F. R. *Spectral graph theory*, volume 92. American Mathematical Soc., 1997.

Cosmo, L., Panine, M., Rampini, A., Ovsjanikov, M., Bronstein, M. M., and Rodola, E. Isospectralization, or how to hear shape, style, and correspondence. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 7529–7538, 2019.

Crane, K., De Goes, F., Desbrun, M., and Schröder, P. Digital geometry processing with discrete exterior calculus. In *ACM SIGGRAPH 2013 Courses*, pp. 1–126. 2013.

Dawson-Haggerty et al. trimesh. URL https://trimsh.org/.

Gauss, C. F. *Disquisitiones Generales Circa Superficies Curvas*. Dieterich, Göttingen, 1827. Commentatio mathematica, quae invitationi respondet, ad locum decimum tertium ordinis primi.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pp. 1263–1272. JMLR.org, 2017.

Guo, J., Wang, H., Cheng, Z., Zhang, X., and Yan, D.-M. Learning local shape descriptors for computing non-rigid dense correspondence. *Computational Visual Media*, 6: 95–112, 2020.

Hamidian, H., Hu, J., Zhong, Z., and Hua, J. Quantifying shape deformations by variation of geometric spectrum. In *Medical Image Computing and Computer-Assisted Intervention-MICCAI 2016: 19th International Conference, Athens, Greece, October 17-21, 2016, Proceedings, Part III 19*, pp. 150–157. Springer, 2016.

Hanocka, R., Hertz, A., Fish, N., Giryes, R., Fleishman, S., and Cohen-Or, D. Meshcnn: a network with an edge. *ACM Transactions on Graphics (TOG)*, 38(4):1–12, 2019.

Hore, A. and Ziou, D. Image quality metrics: PSNR vs. SSIM. In *2010 20th international conference on pattern recognition*, pp. 2366–2369. IEEE, 2010.

Jacobson, A., Panozzo, D., et al. libigl: A simple C++ geometry processing library, 2018. https://libigl.github.io/.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

Larios-Cárdenas, L. Á. and Gibou, F. A deep learning approach for the computation of curvature in the level-set method. *SIAM Journal on Scientific Computing*, 43(3): A1754–A1779, 2021.

Le Gall, F. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th international symposium on symbolic and algebraic computation*, pp. 296–303, 2014.

Lehoucq, R. B., Sorensen, D. C., and Yang, C. *ARPACK Users' Guide*. Society for Industrial and Applied Mathematics, 1998. doi: 10.1137/1.9780898719628. URL https://epubs.siam.org/doi/abs/10.1137/1.9780898719628.

Leng, B., Guo, S., Zhang, X., and Xiong, Z. 3d object retrieval with stacked local convolutional autoencoder. *Signal Processing*, 112:119–128, 2015.

Lévy, B. Laplace-Beltrami eigenfunctions towards an algorithm that "understands" geometry. In *IEEE International Conference on Shape Modeling and Applications 2006 (SMI'06)*, pp. 13–13. IEEE, 2006.

Li, Q., Han, Z., and Wu, X.-M. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Litman, R. and Bronstein, A. M. Learning spectral descriptors for deformable shape correspondence. *IEEE transactions on pattern analysis and machine intelligence*, 36(1): 171–180, 2013.

MacNeal, R. H. *The solution of partial differential equations by means of electrical networks*. PhD thesis, California Institute of Technology, 1949.

Marin, R., Rampini, A., Castellani, U., Rodolà, E., Ovsjanikov, M., and Melzi, S. Instant recovery of shape from spectrum via latent space connections. In *2020 International Conference on 3D Vision (3DV)*, pp. 120–129, 2020. doi: 10.1109/3DV50981.2020.00022.

Marin, R., Attaiki, S., Melzi, S., Rodolà, E., and Ovsjanikov, M. Smoothness and effective regularizations in learned embeddings for shape matching. *arXiv preprint arXiv:2112.07289*, 2021.

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

Meyer, M., Desbrun, M., Schröder, P., and Barr, A. H. Discrete differential-geometry operators for triangulated 2-manifolds. In *International Workshop on Visualization and Mathematics*, 2002. URL https://api.semanticscholar.org/CorpusID:11850545.

Milano, F., Loquercio, A., Rosinol, A., Scaramuzza, D., and Carlone, L. Primal-dual mesh convolutional neural networks. *Advances in Neural Information Processing Systems*, 33:952–963, 2020.

Nguyen, H. V. and Bai, L. Cosine similarity metric learning for face verification. In *Computer Vision–ACCV 2010: 10th Asian Conference on Computer Vision, Queenstown, New Zealand, November 8-12, 2010, Revised Selected Papers, Part II 10*, pp. 709–720. Springer, 2011.

Nt, H. and Maehara, T. Revisiting graph neural networks: All we have is low-pass filters. *arXiv preprint arXiv:1905.09550*, 2019.

Rampini, A., Tallini, I., Ovsjanikov, M., Bronstein, A. M., and Rodolà, E. Correspondence-free region localization for partial shape similarity via Hamiltonian spectrum alignment. In *2019 International Conference on 3D Vision (3DV)*, pp. 37–46, 2019. doi: 10.1109/3DV.2019.00014.

Reuter, M., Wolter, F.-E., and Peinecke, N. Laplace–Beltrami spectra as 'Shape-DNA' of surfaces and solids. *Computer-Aided Design*, 38(4):342–366, 2006.

Reuter, M., Wolter, F.-E., Shenton, M., and Niethammer, M. Laplace–Beltrami eigenvalues and topological features of eigenfunctions for statistical shape analysis. *Computer-Aided Design*, 41(10):739–755, 2009. ISSN 0010-4485. Selected Papers from the 2007 New Advances in Shape Analysis and Geometric Modeling Workshop.

Schoen, R. M. and Yau, S.-T. *Lectures on differential geometry*, volume 1. International press Cambridge, MA, 1994.

Sharp, N., Attaiki, S., Crane, K., and Ovsjanikov, M. Diffusionnet: Discretization agnostic learning on surfaces. *ACM Transactions on Graphics (TOG)*, 41(3):1–16, 2022.

Solomon, J., Crane, K., and Vouga, E. Laplace-Beltrami: The swiss army knife of geometry processing. In *Symposium on Geometry Processing Graduate School (Cardiff, UK, 2014)*, volume 2, 2014.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

Wachinger, C., Golland, P., Kremen, W., Fischl, B., and Reuter, M. BrainPrint: A discriminative characterization of brain morphology. *NeuroImage*, 109:232–248, 2015. ISSN 1053-8119.

Wang, Y. and Solomon, J. Intrinsic and extrinsic operators for shape analysis. In *Handbook of Numerical Analysis*, volume 20, pp. 41–115. Elsevier, 2019.

Wu, Y., Wu, T., and Yau, S.-T. Surface eigenvalues with lattice-based approximation in comparison with analytical solution. *arXiv preprint arXiv:2203.03603*, 2022.

Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K.-i., and Jegelka, S. Representation learning on graphs with jumping knowledge networks. In *International conference on machine learning*, pp. 5453–5462. PMLR, 2018.

Yau, S.-T. Review of geometry and analysis. *Asian Journal of Mathematics*, 4(1):235–278, 2000.

Zhou, Q. and Jacobson, A. Thingi10K: A dataset of 10,000 3d-printing models. *arXiv preprint arXiv:1605.04797*, 2016.

Zhou, Q.-Y., Park, J., and Koltun, V. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018.

A. Inverse Laplacian and Asymptotics

Each summand of Inverse Laplacian is similar to the shifted negative reciprocal of the relative error, and as the prediction approaches the ground truth, each summand behaves similarly to the relative error:

Lemma A.1. *Given ground truth g_i and its respective prediction p_i, if $g_i \neq 0$ and $\frac{g_i}{p_i} - 1 = \varepsilon$ then $\frac{p_i - g_i}{g_i} = \frac{\varepsilon}{1+\varepsilon}$.*

Proof. Note that because $g_i \neq 0$, $\varepsilon \neq -1$. Then

$$\frac{g_i}{p_i} = 1 + \varepsilon \implies \frac{g_i - p_i}{g_i} = \frac{\varepsilon}{1+\varepsilon}. \qquad \square$$

Note that as $p \to g$, $\varepsilon \to 0$ and so the $1 + \varepsilon$ term in the denominator is dominated by the 1. This is similar to the shifted negative reciprocal of relative error. To see this, rewrite $\frac{\varepsilon}{1+\varepsilon} = \frac{1+\varepsilon-1}{1+\varepsilon} = 1 - \frac{1}{1+\varepsilon}$. This is a shifted negative reciprocal function. We illustrate this relationship by examining the asymptotes of this function. There are two asymptotes: the limit as $\varepsilon \to \pm\infty$ is 1, and the expression is unbounded as $\varepsilon \to -1$. Note that $\left|\frac{g}{p} - 1\right| \to \infty$ if and only if $\frac{|g|}{|p|} \to \infty$, i.e. $|g| \gg |p|$. This implies $|g - p| \approx |g|$ so $\frac{|g-p|}{|g|} \approx 1$, i.e. relative error converges to 1. On the other hand, if $\varepsilon \to -1$ then $\frac{|g|}{|p|} \to 0$, i.e. $|g| \ll |p|$, thus $|g - p| \approx |p|$ and so the relative error is given by $\frac{|g-p|}{|g|} \approx \frac{|p|}{|g|} \to \infty$, i.e. the relative error is unbounded.

B. Examples of Predicted Eigenvalues

$\mathrm{PSNR} > 20$ is used as an evaluation metric for model predictions. To provide an intuition of what $\mathrm{PSNR} > 20$ means in the context of a vector of ten eigenvalues, we exhibit a few examples of predicted eigenvalues versus ground truth eigenvalues, with two examples of predictions that are considered "unsuccessful," i.e., $\mathrm{PSNR} < 20$, and four examples of predictions that are considered "successful," i.e., $\mathrm{PSNR} > 20$. We also supply the respective inverse Laplacian with each chart, to provide a visual intuition for inverse Laplacian as well.

In general, when the PSNR is large and the Inverse Laplacian is small, it indicates that the prediction is very close to the ground truth. We find PSNR is relatively forgiving to outliers if predictions of most other eigenvalues are accurate. This highlights the difference between PSNR and Inverse Laplacian: the former provides a holistic view of accuracy while the latter is concerned with the accuracy of each individual eigenvalue.

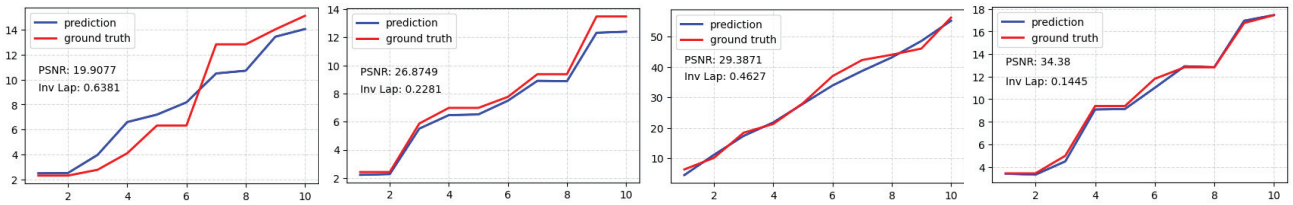


Figure 6. Some samples of the evaluation metrics on eigenvalues. In particular, when $\mathrm{PSNR} > 20$, the predicted eigenvalues, in blue, approximate the true eigenvalues well.

C. Synthetic Dataset

The synthetic dataset starts with 16,550 triangle meshes. This consists of 2,000 rectangular prisms, 1,000 cones, 1,000 cylinders, 50 spheres, 2,000 tori, 2,000 tetrahedra, 1,000 conical frustrums, 2,500 split spheres, 1,000 split cones, and 4,000 inverted cones. We compute the eigenvalues using Reuter's LaPy solver (Reuter et al., 2006; Wachinger et al., 2015), and then filter out any meshes whose eigenvalues have numerical abnormalities. This leaves 16,361 meshes before augmentations.

Standard shapes The triangle meshes for the rectangular prisms, cones, cylinders, spheres, and tori are generated using Open3D's built-in methods (Zhou et al., 2018). For each mesh, the parameters are sampled from the distribution given in Table 5.

Table 5. Dataset Parameters. The radius of all applicable shapes (i.e., all but the rectangular prism and the tetrahedron) is 1. For tori, this means that the parameter torus_radius is always 1. $N(\mu, \sigma^2)$ denotes a normal distribution with mean μ and standard deviation σ and $U(a, b)$ denotes a uniform distribution with range $[a, b]$. As the normal distribution can generate arbitrary large / small values, we clamp all values to the range $[\mu - 3\sigma, \mu + 3\sigma]$ as there is an approximately 99.7% chance a randomly generated sample naturally falls in this range. For split meshes, the resolution and split parameters are sampled twice two create two meshes.

Parameters	Height	Subdivisions	Resolution	Split	Other
Rectangular Prism	$10^{N\left(0,(2/3)^2\right)}$	$U(3, 5)$	N/A	N/A	Width: 1, Depth: $10^{N\left(0,(2/3)^2\right)}$
Cone	$10^{N\left(0,(2/3)^2\right)}$	1	$U(20, 100)$	$U(1, 20)$	N/A
Split Cone	$10^{N\left(0,(2/3)^2\right)}$	1	$U(20, 100)$	$U(1, 20)$	N/A
Conical Frustum	$10^{N\left(0,(2/3)^2\right)}$	0	$U(20, 100)$	$U(1, 20)$	Scaling factor: $10^{N\left(0,(2/3)^2\right)}$
Cylinder	$10^{N\left(0,(2/3)^2\right)}$	0	$U(20, 100)$	$U(4, 20)$	N/A
Sphere	N/A	0	$U(20, 100)$	N/A	N/A
Split Sphere	N/A	0	$U(20, 100)$	N/A	N/A
Tetrahedron	$10^{N\left(0,(2/3)^2\right)}$	$U(3, 5)$	N/A	N/A	Width: 1, Depth: $10^{N\left(0,(2/3)^2\right)}$
Torus	N/A	0	N/A	N/A	Tube radius: $10^{N\left(1/2,(1/6)^2\right)}$
Inverted Cone	$10^{N\left(0,(2/3)^2\right)}$	1	$U(20, 100)$	$U(10, 30)$	Flip: $U(0.5, 1)$

For tetrahedra, the parameters width, height, and depth are sampled from the distributions given in Table 5. A triangle mesh is then constructed with vertices at $(0, 0, 0), (w, 0, 0), (0, h, 0), (0, 0, d)$, with w being the width, h being the height, and d being the depth. All triples of vertices are joined as faces.

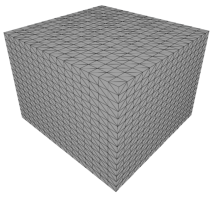
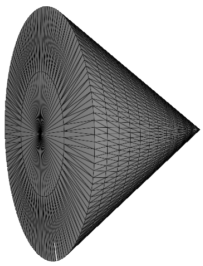
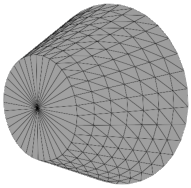
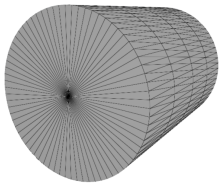
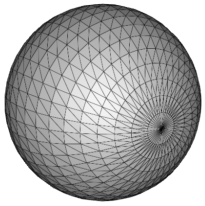
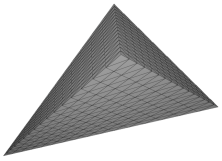
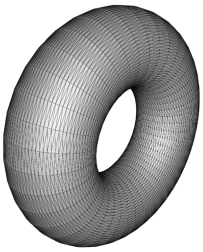
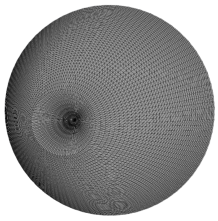
Finally, conical frustums are generated by first generating a cylinder by sampling the appropriate parameters. One of the radii is then scaled by a multiplicative scaling factor, sampled from the distribution given in Table 5, and this scaling is linearly interpolated along the height of the cylinder.

Split shapes and inverted cones The split shapes, i.e., the split cones and split spheres, are cones and spheres, respectively, generated by constructing two meshes of the same underlying shape of different resolutions using Open3D's built-in methods. In particular, the degree to which curved surfaces were approximated differed between the two meshes. The two meshes are each cut in half by the plane $x = 1$, and the two halves of the meshes are joined together by converting to point clouds and then merged using Open3D's PointCloud.compute_convex_hull.

Inverted cones are generated by generating a cone and then inverting its tip. This operation preserves the intrinsic geometry of the mesh and so the eigenvalues of the Laplacian operation are invariant under this operation (Wang & Solomon, 2019). The inverted cone is generated by generating a cone using Open3D's built-in method with parameters sampled from the distributions given in Table 5. The tip of the cone is inverted by sampling some $t \in [0.5, 1]$ from the distribution labelled 'flip' in Table 5, before rounding for mesh integrity reasons and reflecting all points with $z > th$ over the plane $z = th$ where h is the height of the cone.

Additionally, to better approximate the shapes for the FEM solver, we iteratively subdivide the meshes by the midpoints of their edges. The number of times this process is done is sampled from the distribution given in Table 5. We present representative sample meshes for each shape in Table 6.

Table 6. A representative mesh from each type of shape. We do not display split shapes (shapes sampled with different resolution) since they are visually indistinguishable to their non-split counterparts given the displayed sizes.

Rectangular Prism	Cone	Conical Frustum	Cylinder
			
Sphere	**Tetrahedron**	**Torus**	**Inverted Cone**