EPFL

Thèse n° 7398

Leveraging topology, geometry, and symmetries for efficient Machine Learning

Présentée le 31 mai 2022

Faculté des sciences et techniques de l'ingénieur
Laboratoire de traitement des signaux 2
Programme doctoral en génie électrique

pour l'obtention du grade de Docteur ès Sciences

par

Michaël DEFFERRARD

Acceptée sur proposition du jury

Prof. P. Frossard, président du jury
Prof. P. Vandergheynst, directeur de thèse
Prof. M. Welling, rapporteur
Prof. Y. LeCun, rapporteur
Prof. M. Jaggi, rapporteur

École polytechnique fédérale de Lausanne

2022

...all progress, both theoretical and practical,
has resulted from a single human activity:
the quest for what I call good explanations.
— David Deutsch

Dedicated to the loving memory of my father
Christian Defferrard
1960–2003

Acknowledgments

First and foremost, I want to thank Pierre Vandergheynst for giving me the opportunity to pursue a PhD in his laboratory and for providing me with an environment where my creativity could flourish. Pierre gave me resources, freedom, and trust without asking for anything in return. Next, I want to thank Xavier Bresson for guiding me through both technical and non-technical matters. Xavier was an adviser and mentor until he left for his own lab. I would not be who I am without him. I am grateful to my thesis committee—Martin Jaggi, Pascal Frossard, Max Welling, Yan LeCun—for their interest and enthusiasm: it was an honor to discuss my research with you.

Over the years, I had the pleasure to interact with and learn from many people. I thank Nathanaël Perraudin for his patience teaching me: I had so many questions! A unique collaborator and friend. Along with Vassilis Kalofolias and Johan Paratte, I thank this trio for bringing me into this journey at the LTS2 laboratory and getting me started. To them and to all the marvelous companions at the LTS2 and beyond, I thank you for the time we shared turning these years into a memorable experience. It was wonderful to grow together. Special thanks go to my office-mates, Nauman Shahid, Rodrigo Pena, Volodymir Miz, Konstantinos Pitas, and Youngjoo Seo: they made our office a lovely place and the table football a source of much after-lunch excitement. I thank my collaborators, including Andreas Loukas, Stefania Ebli, Gard Spreemann, Gionata Ghiggi, Katharina Glomb, Erik Bekkers, Tomasz Kacprzak, Sharada P. Mohanty, Kirell Benzi, Frank de Morsier, Bruno Correia, and Zander Harteveld, for the many fruitful discussions and learning opportunities. I thank the many talented students I have advised, in particular Yoann Ponti, Martino Milani, Jelena Banjac, and Hugo Aguettaz: it was a privilege.

Finally, I want to express my gratitude to my parents for nurturing my curiosity and independence, and to my family and friends for providing love, fun, and support. And most importantly, I thank my wife, Jessica, who always believed in me, and my daughter, Julia, who lighted up the last few months of this adventure.

Thank you.

Abstract

When learning from data, leveraging the symmetries of the domain the data lies on is a principled way to combat the curse of dimensionality: it constrains the set of functions to learn from. It is more data efficient than augmentation and gives a generalization guarantee. Symmetries might however be unknown or expensive to find; domains might not be homogeneous.

From the building blocks of space (vertices, edges, simplices), an incidence structure, and a metric—the domain's topology and geometry—a linear operator naturally emerges that commutes with any known and unknown symmetry action. We call that equivariant operator the *generalized convolution operator*. And we use it, designed or learned, to transform data and embed domains. In our generalized setting involving unknown and non-transitive symmetry groups, our convolution is an inner-product with a kernel that is localized instead of moved around by group actions like translations: a bias–variance tradeoff that paves the way to efficient learning on arbitrary discrete domains.

We develop convolutional neural networks that operate on graphs, meshes, and simplicial complexes. Their implementation amounts to the multiplications of data tensors by sparse matrices and pointwise operations, with linear compute, memory, and communication requirements. We demonstrate our method's efficiency by reaching state-of-the-art performance for multiple tasks on large discretizations of the sphere. DeepSphere has been used for studies in cosmology and shall be used for operational weather forecasting—advancing our understanding of the world and impacting billions of individuals.

Keywords. Convolutional neural networks, graph neural networks, network embedding, graph signal processing, discrete calculus, equivariance, symmetry groups.

Résumé

Lors de l'apprentissage à partir de données, l'exploitation des symétries du domaine sur lequel reposent les données est une manière raisonnée de lutter contre le fléau de la dimension : elle contraint l'ensemble des fonctions à partir desquelles apprendre. C'est plus efficace en termes de données que l'augmentation et ça donne une garantie de généralisation. Les symétries peuvent cependant être inconnues ou coûteuses à trouver ; les domaines peuvent ne pas être homogènes.

À partir des éléments constitutifs de l'espace (sommets, arêtes, simplexes), d'une structure d'incidence et d'une métrique – la topologie et la géométrie du domaine – un opérateur linéaire émerge naturellement qui commute avec toute action de symétrie, connue et inconnue. Nous appelons cet opérateur équivariant l'*opérateur de convolution généralisé*. Et nous l'utilisons, conçu ou appris, pour transformer des données et plonger des domaines. Dans notre cadre généralisé impliquant des groupes de symétrie inconnus et non-transitifs, notre convolution est un produit scalaire avec un noyau qui est localisé au lieu d'être déplacé par les actions d'un groupe telles que les translations : un compromis biais-variance qui ouvre la voie à un apprentissage efficace sur des domaines discrets arbitraires.

Nous développons des réseaux de neurones convolutifs qui opèrent sur des graphes, des maillages et des complexes simpliciaux. Leur implémentation revient à multiplier des tenseurs de données par des matrices creuses et des opérations ponctuelles, avec des exigences de calcul, de mémoire et de communication linéaires. Nous démontrons l'efficacité de notre méthode en atteignant des performances de pointe pour de multiples tâches sur de grandes discrétisations de la sphère. DeepSphere a été utilisé pour des études en cosmologie et devrait être utilisé pour la prévision météorologique, faisant progresser notre compréhension du monde et impactant des milliards d'individus.

Mots-clés. Réseaux de neurones convolutifs, réseaux de neurones sur graphe, plongement de réseaux, traitement du signal sur graphe, calcul différentiel discret, équivariance, groupes de symétrie.

Contents

1 Introduction

The goal of Science is to learn from data: we seek explanations that are consistent with observations. An explanation can be a human-designed theory—formalized in a natural, mathematical, or computer language. Or it can be a machine-learned model.

The goal of machine learning is to automate the task of finding models from data. At least to help the humans doing it. During my doctoral studies, I have tackled and contributed to solving the following problems: from observations, estimate the free parameters of a human-designed theory of the largest-scale structures and dynamics of our universe, the ΛCDM cosmological model (Chapter 5). From observations, learn a model of the dynamics of the Earth's atmosphere, towards the goal of predicting its evolution given its current state, i.e., weather forecasting ([62], in preparation). Those were the problem-driven guides to my research.

1.1 Solution space: constraints and biases

Learning is equivalent to searching for a model $f \in \mathcal{F}$, or a distribution over $\mathcal{F}$, where the hypothesis class $\mathcal{F}$ is the set of models under consideration (Figure 1.1). Typically, a model f_θ is parameterized by a finite number of parameters θ that are optimized to achieve an objective. Multi-layer perceptrons (MLPs) are a kind of artificial neural network whose layers are fully connected, i.e., implement an unconstrained linear transformation. MLPs are powerful models: they are universal approximators [82]. The flip side of this flexibility is that they suffer from the curse of dimensionality: they require

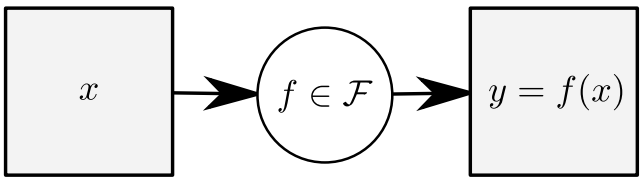


Figure 1.1: Learning is finding a model $f \in \mathcal{F}$, or a distribution over the solution space $\mathcal{F}$, from data $\{(x_i, y_i)\}$.

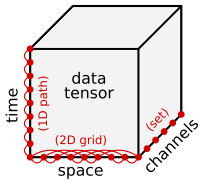


Figure 1.2: A data tensor that is structured along 2 of its 3 dimensions.

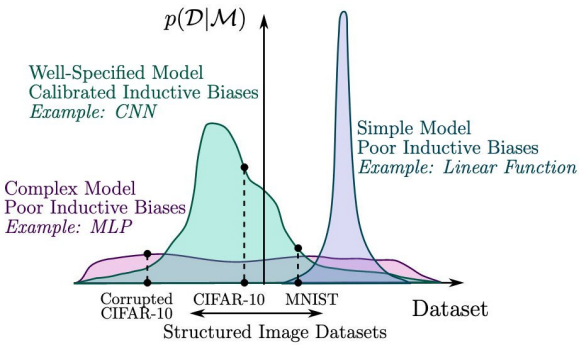


Figure 1.3: A well-designed solution space $\mathcal{F}$ promotes a priori likely solutions and forbids those that violate the problem's constraints. Figure from [169].

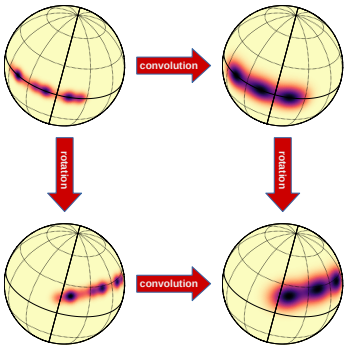


Figure 1.4: Symmetry constraints (i.e., equivariance) yield convolutions.

exponentially many examples to learn from high-dimensional data. One way to combat that curse is to gather more data. One could collect more data or augment the collected data. The first costs human resources, the second computer resources.

A more efficient alternative is to design the solution space $\mathcal{F}$. Its support should implement the constraints of the problem; for example, fluid dynamics does not create mass. Those gives the designer a generalization guarantee. Its inductive biases should promote a priori likely solutions; for example, the identity to predict tomorrow's weather given today's. Those are not hard constraints, but prior knowledge that makes likely solutions easier to find while allowing unlikely ones given enough data. Solution spaces are illustrated in Figure 1.3.

Machine learning moves design a level of abstraction up: from that of solutions f to that of solution spaces $\mathcal{F}$. While that was possible in theory before, it was a practical revolution enabled by deep learning. The resulting increase in leverage of human creativity impacted most sciences and arts, and disrupted many industries. That might well turn out to be paradigm shift [103] in Science.

1.2 Structure and Symmetries

Data is multi-dimensional: it often has a space, time, and feature dimension. Data is either inherently discrete or is a discretization (because of finite measurements and computers). Finally, the dimensions are most often structured, i.e., data is a function of a set with additional structure, such as a topology and a geometry. Figure 1.2 shows a typical data tensor. That structure yields a discrete calculus (§2.1) that we ought to

leverage (§2.2) to learn more efficiently. Figuring out how to make machines efficiently leverage structure was the curiosity-driven guide to my research.

Equivariance to symmetries is one way that structure constrains the solution space as

$$\mathcal{F} = \left\{ f \,\middle|\, f(\sigma \cdot x) = \sigma \cdot f(x) \; \forall x \in X, \sigma \in \mathrm{Sym}(X) \right\}.$$

Figure 1.4 depicts the commutative diagram for $X = \mathbb{S}^2$ and $\mathrm{Aut}(\mathbb{S}^2) = SO(3)$. A special case of equivariance is invariance, defined as $f(\sigma \cdot x) = f(x) \, \forall \, \sigma$. That is typically desired in global tasks, where the space dimension is squashed towards the later layers (as in the regression setting used in Chapter 5).

In practice, the functional space $\mathcal{F}$ is determined by the architecture of the neural network. Mostly by the kind of linear transformations its layers can perform. The classic recipe to build equivariant linear transformations is to leverage global symmetries and perform a convolution as $\langle x, P_\sigma g \rangle$, the inner product between the data and a filter g moved around by the action P_σ of the symmetry $\sigma \in \mathrm{Sym}(X)$. Convolution on groups and homogeneous spaces through group action is well understood. The most common example of that recipe is the classical convolutional neural network, where X is the plane and $\mathrm{Sym}(X)$ are translations. After Euclidean spaces, the sphere $\mathbb{S}^2$ is certainly the most practically relevant space to convolve on (Chapter 4). The generalization of the recipe to any symmetry group has been popularized in machine learning by Cohen, Kondor, and others, in a line of work developed concurrently to that presented here. See for example [33] and [98]. A convolutional structure is not only a sufficient condition for equivariance, but also a necessary one [98]. The imposed convolutional structure sparsifies layer connectivity, makes the connections local, and shares parameters across the domain. For example, the matrix representation of a 1D convolution, which is equivariant to the action of the cyclic group, is circulant.

The main limitation of convolutions by symmetry action is that the domain must be homogeneous for parameters to be shared around it. Figure 1.5b shows a common case: a graph made of a large asymmetric core and many small symmetric motifs. Even a finite discretization of the plane does not have enough symmetries to be homogeneous (Figure 1.5a). Gauge equivariance has been proposed to learn on non-homogeneous domains [35], although it yields a locally connected rather than convolutional layer, i.e., there is no parameter sharing. Sharing parameters on non-homogeneous domains require an additional constraint.

Another limitation of leveraging symmetry action is that it presumes that the symmetries are known. While the symmetries of common data domains like Euclidean space

(a) Grid. (b) Asymmetric core of 7 vertices and 2 symmetric motifs.

Figure 1.5: Two domains that have not enough symmetries to move data around. Symmetries can only move data between vertices of the same color (but black), i.e., equivalent vertices who are in the same orbit of the domain's symmetry group. Another mechanism is required to share weights around non-homogeneous (non-transitive) domains.

and the sphere are known and well studied, symmetries might be unknown, expensive to find (e.g., large social or biological networks), or exist only approximately (e.g., sampled spheres).

1.3 Research question

My research question is the following:

> How to leverage the topological and geometrical structure of the data's domain to learn efficiently without the help of symmetry action?

I answer it by proposing to learn the parameters of a linear transformation that is a priori equivariant to symmetry action (Chapters 2 and 3). Its use to tackle real-world problems with spherical data (Chapters 4 and 5) shows that the constraint introduced to enable parameter sharing does not prevent DeepSphere from reaching state-of-the-art performance, while being more memory and compute efficient than alternatives. Here joins the curiosity- and problem-driven approaches to my research.

1.4 A bias–variance tradeoff

While it remains to be determined when (and why) dropping the parameter sharing constraint might be worth paying in compute, memory, and data; we should note that it is all a tradeoff between bias and variance—the most fundamental lesson of machine learning [168].

One might believe that data is enough: we shall just throw more data at a larger MLP. Or one might believe that data is not enough: we still need to design the hypothesis class

to learn from. Needless to write that I stand firmly in the second camp. That is, until data proves me wrong.

1.5 Contributions

This document is made of the following manuscripts:

Chapter 2 [41] M. Defferrard. "Generalized convolutions". In: *In preparation*. 2022

Chapter 3 [43] M. Defferrard, X. Bresson, and P. Vandergheynst. "Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering". In: *Advances in Neural Information Processing Systems (NIPS)*. 2016. arXiv: 1606.09375. URL: https://arxiv.org/abs/1606.09375

Chapter 4 [45] M. Defferrard, M. Milani, F. Gusset, and N. Perraudin. "DeepSphere: a graph-based spherical CNN". in: *International Conference on Learning Representations (ICLR)*. 2020. arXiv: 2012.15000. URL: https://arxiv.org/abs/2012.15000

Chapter 5 [132] N. Perraudin, M. Defferrard, T. Kacprzak, and R. Sgier. "DeepSphere: Efficient spherical Convolutional Neural Network with HEALPix sampling for cosmological applications". *Astronomy and Computing* 27, 2019, pp. 130–146. ISSN: 2213-1337. DOI: 10.1016/j.ascom.2019.03.004. arXiv: 1810.12186. URL: https://arxiv.org/abs/1810.12186

While not forming their own chapter, ideas from the following manuscripts are scattered throughout:

- [53] S. Ebli, M. Defferrard, and G. Spreemann. "Simplicial Neural Networks". In: *Topological Data Analysis and Beyond workshop at NeurIPS*. 2020. arXiv: 2010.03633. URL: https://arxiv.org/abs/2010.03633
- [48] M. Defferrard, N. Perraudin, T. Kacprzak, and R. Sgier. "DeepSphere: towards an equivariant graph-based spherical CNN". in: *ICLR Workshop on Representation Learning on Graphs and Manifolds*. 2019. arXiv: 1904.05146. URL: https://arxiv.org/abs/1904.05146
- [150] Y. Seo, M. Defferrard, P. Vandergheynst, and X. Bresson. "Structured Sequence Modeling with Graph Convolutional Recurrent Networks". In: *International Conference on Neural Information Processing (ICONIP)*. 2017. arXiv: 1612.07659. URL: https://arxiv.org/abs/1612.07659

During my doctoral studies, I also co-authored the following papers, which do not appear in this document:

- G. Ghiggi, M. Defferrard, W. Feng, Y. Y. Haddad, N. Bolón Brun, I. Lloréns Jover, and P. Dueben. "DeepSphere-Weather: Deep Learning on spherical unstructured grids for weather/climate applications". *In preparation*, 2022. URL: https://github.com/deepsphere/deepsphere-weather

- H. Aguettaz, E. J. Bekkers, and M. Defferrard. "ChebLieNet: Invariant spectral graph NNs turned equivariant by Riemannian geometry on Lie groups". In: 2021. arXiv: 2111.12139. URL: https://arxiv.org/abs/2111.12139

- Z. Harteveld, J. Southern, M. Defferrard, A. Loukas, P. Vandergheynst, M. M. Bronstein, and B. E. Correia. "Deep sharpening of topological features for de novo protein design". In: *Machine Learning for Drug Discovery Workshop at ICLR*. 2022. URL: https://openreview.net/forum?id=H5g7W-6Oec

- A. Scheck, S. Rosset, M. Defferrard, A. Loukas, J. Bonet, P. Vandergheynst, and B. E. Correia. "RosettaSurf—A surface-centric computational design approach". *PLOS Computational Biology* 18:3, 2022. bioRxiv: 2021.06.16.448645, pp. 1–23. DOI: 10.1371/journal.pcbi.1009178

- J. Banjac, L. Donati, and M. Defferrard. "Learning to recover orientations from projections in single-particle cryo-EM". in: 2021. arXiv: 2104.06237. URL: https://arxiv.org/abs/2104.06237

- K. Glomb, J. Rué Queralt, D. Pascucci, M. Defferrard, S. Tourbier, M. Carboni, M. Rubega, S. Vulliémoz, G. Plomp, and P. Hagmann. "Connectome spectral analysis to track EEG task dynamics on a subsecond scale". *NeuroImage* 221, 2020. bioRxiv: 2020.06.22.164111, pp. 117–137. ISSN: 1053-8119. DOI: 10.1016/j.neuroimage.2020.117137

- M. Defferrard, S. P. Mohanty, S. F. Carroll, and M. Salathé. "Learning to Recognize Musical Genre from Audio. Challenge Overview". In: *The 2018 Web Conference Companion*. ACM Press, 2018. ISBN: 9781450356404. DOI: 10.1145/3184558.3192310. arXiv: 1803.05337. URL: https://arxiv.org/abs/1803.05337

- M. Defferrard, K. Benzi, P. Vandergheynst, and X. Bresson. "FMA: A Dataset for Music Analysis". In: *18th International Society for Music Information Retrieval Conference (ISMIR)*. 2017. arXiv: 1612.01840. URL: https://arxiv.org/abs/1612.01840

Finally, robust and useful results for my research were consolidated in the following stable and maintained software packages with thought-out APIs:

- M. Defferrard, L. Martin, R. Pena, and N. Perraudin. *PyGSP: Graph Signal Processing in Python*. DOI: 10.5281/zenodo.1003157. URL: https://github.com/epfl-lts2/pygsp/
- F. Gusset, L. Vancauwenberghe, M. Allemann, J. Fluri, N. Perraudin, and M. Defferrard. *DeepSphere: learning on the sphere*. URL: https://github.com/deepsphere/
- M. Defferrard, R. Pena, and N. Perraudin. *PyUNLocBoX: Optimization by Proximal Splitting*. DOI: 10.5281/zenodo.1199081. URL: https://github.com/epfl-lts2/pyunlocbox/

As software is in my opinion not yet recognized enough as core contributions to Science, I take this opportunity to acknowledge the use of the following software packages in pursuing my research and producing this document: PyGSP [44], healpy [172], matplotlib [84], SciPy [165], NumPy [166], TensorFlow [117], PyTorch [129].

2 Generalized convolutions

2.1 A discrete calculus

In this section, we will build a discrete calculus from the foundations of space: its topology and geometry. The presented material follows standard works in discrete calculus, such as that presented in [19, 49, 69], which is also referred to as discrete exterior calculus (DEC) or discrete differential geometry (DDG). A discrete calculus analogue of the Laplacian operator first appeared in [54]. I will however take my own perspective and sometimes diverge from the standard exposition to best suit our ultimate goal of building an efficient tool to learn from structured data. For example, I will use a linear algebra notation, instead of the traditional notation inherited from the continuous setting.

2.1.1 Space

To accommodate our computers' finite memory and for algorithms to finish in a finite amount of time, we must deal with finite representations. Discrete spaces are easier to deal with than parameterizations of continuous ones (e.g., by splines or Bézier curves). Either way, continuous spaces can be modeled to an arbitrary precision by finite discrete spaces through their discretization. Most methods for solving differential equations, such as the finite element, volume, or difference methods, operate in that way. That said, discrete spaces are more than merely approximations of continuous spaces: they have special properties of their own. As we shall see, discrete topology, geometry, and calculus reduce to linear algebra. That is not only useful for their representation and manipulation on computers, but also for simplicity of exposition.

Simplices are the building blocks of discrete spaces: a d-simplex is a set of $d+1$ vertices that represents a d-dimensional subspace. An oriented simplex is an ordered set. Simplices are so-named because they represent the simplest polytope in any given dimension: vertices for $d=0$, edges for $d=1$, triangles for $d=2$, tetrahedra for $d=3$, and so forth (see Figure 2.1).

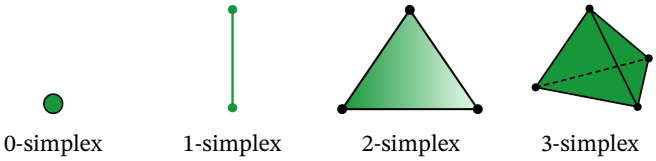


Figure 2.1: Simplices.

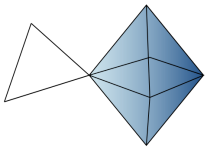


Figure 2.2: Simplicial complex.

A simplicial complex is a collection of simplices that form a discrete space (see Figure 2.2). While the notion emerged in geometry, a purely combinatorial definition suffices: an abstract[1] simplicial complex is a collection of sets closed under taking subsets, i.e., every subset of a set in the collection is also in the collection. We call a member of a simplicial complex K a simplex of dimension d if it has cardinality $d+1$, and denote the set of all such d-simplices K_d. A d-simplex has $d+1$ faces of dimension $d-1$, namely the subsets omitting one element.

Graphs are simplicial complexes made of vertices and edges only. A mesh is a special case of a simplicial complex. Simplicial complexes are special cases of cell complexes (whose d-cells, which must be homeomorphic to d-dimensional simplices, can be made of more than $d+1$ vertices) and hypergraphs (whose collection of sets needs not to be closed under taking subsets). While there are tradeoffs in the choice of discrete structure to represent relationships, simplicial complexes might be a sweet spot. Compared to graphs, they enable the representation of higher-order relations and data (such as vector and tensor fields). Compared to meshes, they are not limited to geometric applications and can represent purely combinatorial spaces. They are easier to represent and manipulate than cell complexes (though the presented theory generalizes to cell complexes). Compared to hypergraphs, they yield a natural discrete calculus.

2.1.2 Data

Unlike in the continuous, where coordinate-free treatments are more elegant, the simplices form a natural basis in which to represent data. Let us call it the spatial basis. To define that basis, simplices must be oriented and ordered (arbitrarily, as those are not properties of space). Simplices are oriented by ordering the vertices they are made of. Simplices are ordered by ordering the sets $\{K_d\}$. A simple way to do both is to arbitrarily order the simplices, then use the ordering of K_0 to orient the higher-order simplices.

Our data are fields in the physics sense, i.e., functions that take a value at every simplex. A vertex-valued function ($d=0$) is a scalar field, for example the temperature. An

[1] A simplicial complex without an associated geometry is said to be abstract.

edge-valued function ($d = 1$) is a vector field, for example the wind, whose evaluation yields the signed magnitude of the wind along an edge (i.e., at a given position and direction). An example tensor field is the fluid flux ($d = 2$), whose evaluation yields the signed magnitude of the flux through a surface. But data needs not be physical measurements. It could be the number of citations attributed to a d-simplex that represents a collaboration between $d + 1$ authors, as we did in [53]. Or it could be an indicator function of a subdomain.

We call d-dimensional data that co-vary with the space's metric d-chains and contravariant data d-cochains. These are dual quantities: the pairing of a d-chain with a d-cochain yields a scalar. Representing data as vectors whose i^{th} component is the value of the chain or cochain at the i^{th} simplex, the dual pairing of the chain $x_d \in \mathbb{R}^{|K_d|}$ with the cochain $f_d \in \mathbb{R}^{|K_d|}$ is denoted by the dot product $\langle x_d, f_d \rangle = {x_d}^{\mathsf{T}} f_d$. Note how the ordering and orientation of simplices enable that representation. The orientation defines what a positive value means (the direction for $d = 1$, the (counter-)clockwise rotation for $d = 2$, the right- or left-handed helix for $d = 3$, etc.).

EXAMPLE. Consider an electrical circuit with voltage information represented as a 1-cochain. Pairing the voltage 1-cochain[2] with a subdomain, represented as a 1-chain, computes a path integral that yields the voltage between the two end vertices. Pairing it with current information, represented as another 1-chain, yields the power dissipated by the circuit.

2.1.3 TOPOLOGY: AN INCIDENCE STRUCTURE

While a set of sets representation, derived from the combinatorial definition of a simplicial complex, is elegant[3], it is not convenient to store and manipulate complexes on computers. As hinted above, we will resort to an algebraic representation. We define $B_d \in \{0, 1, -1\}^{|K_d| \times |K_{d-1}|}$, in the spatial basis, such that

$$(B_d)_{ij} = \begin{cases} 0 & \text{if the } j^{\text{th}} (d-1)\text{-simplex is not a face of the } i^{\text{th}} d\text{-simplex,} \\ +1 & \text{if the } j^{\text{th}} (d-1)\text{-simplex is a face of the } i^{\text{th}} d\text{-simplex with agreeing orientation,} \\ -1 & \text{if the } j^{\text{th}} (d-1)\text{-simplex is a face of the } i^{\text{th}} d\text{-simplex with disagreeing orientation,} \end{cases} \tag{2.1}$$

B_1 is the classic graph incidence matrix whose columns represent the (ordered) vertices K_0 and rows the (ordered and oriented) edges K_1. The set $\{B_d\}$ is a complete represen-

[2] As an integrand, it is also known as a discrete differential 1-form.
[3] It clearly states the single axiom underpinning that object.

tation of the simplicial complex: it records the incidence structure and captures the topology of the space.

We call ${B_d}^\mathsf{T}$ the boundary operator: given a d-chain x_d that represents a subdomain, it computes its boundary ${B_d}^\mathsf{T} x_d$, represented by a $(d-1)$-chain. We call B_d the differential operator[4]: given a $(d-1)$-cochain f_{d-1} that represents some covariant data, it computes the finite difference $B_d f_{d-1}$, represented by a d-cochain.

We observe that

$$\left\langle {B_d}^\mathsf{T} x_d, f_{d-1} \right\rangle = \left\langle x_d, B_d f_{d-1} \right\rangle, \tag{2.2}$$

hence that the boundary operator ${B_d}^\mathsf{T}$ is adjoint to the differential operator B_d with respect to the dual pairing of chains and cochains. That is an expression of the generalized Stokes theorem[5]: a topological result that says that taking the boundary is adjoint to taking the derivative with respect to integration. The result is typically written as $\int_{\partial\Omega} \omega = \int_\Omega \mathrm{d}\omega$, where Ω is a d-dimensional manifold and ω a $(d-1)$-differential form, which is easily identified with (2.2).

2.1.4 Geometry: an inner product

To enable the computation of norms and distances between chains and cochains, we add a geometrical structure to our topological space. We define the inner-product between the cochains f_d and h_d as $\langle f_d, h_d \rangle_{M_d} = {f_d}^\mathsf{T} M_d h_d$, where M_d is a positive (semi-)definite matrix that we will call the metric. Naturally assuming that the simplices are independent, i.e., that the spatial basis is orthogonal, the metric is diagonal and $(M_d)_{ii}$ can be interpreted as the weight of the i^{th} d-simplex. By convention, I choose these weights to represent similarities (affinities), which contra-vary with cochains. The set $\{M_d\}$ records the inner product structure and captures the geometry of the space.

The metric M_d induces an isomorphism between the spaces of chains and cochains[6]: given a cochain f_d, one gets the chain $M_d f_d$ through the induced map $\langle f_d, \cdot \rangle_{M_d} = \langle M_d f_d, \cdot \rangle$. For this isomorphism to preserve the inner product, we must have $\langle f_d, h_d \rangle_{M_d} = \langle M_d f_d, M_d h_d \rangle_{M_d^{-1}}$. Hence, ${M_d}^{-1}$ is the metric for chains. It represents edge lengths, triangle areas, and simplex volumes in general, which, as expected, contra-vary with chains. In the other direction, one gets the cochain ${M_d}^{-1} x_d$ from the chain x_d.

[4] It is also known as the coboundary operator or the exterior derivative.
[5] A generalization of the fundamental theorem of calculus, the divergence theorem, and Green's theorem.
[6] In the continuous, that role is played by the musical isomorphism, also known as index raising/lowering in tensor analysis.

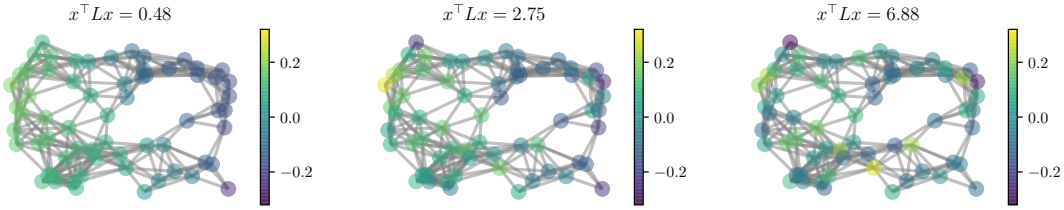


Figure 2.3: Dirichlet energy of a 0-cochain. The energy $E(f_0) = \langle f_0, L_0 f_0 \rangle_{M_0} = \|B_1 f_0\|^2_{M_1}$ measures the smoothness, or variation, of the 0-cochain f_0 as the norm of its gradient 1-cochain $B_1 f_0$.

By equipping the vector space of cochains (and so chains) with an inner product, it becomes an Hilbert space. As expected, the inner product induces the norm $\|f_d\|^2_{M_d} = \langle f_d, f_d \rangle_{M_d}$ and distance $d(f_d, h_d) = \|f_d - h_d\|_{M_d}$.

2.1.5 A DISCRETE CALCULUS

With the topological and geometrical structures in B_d and M_d, we have everything to build a discrete calculus. Combining both structures, we observe that

$$\langle B_d f_{d-1}, h_d \rangle_{M_d} = \left\langle f_{d-1}, {B_d}^\dagger h_d \right\rangle_{M_{d-1}}, \tag{2.3}$$

which implies that the adjoint of the difference operator B_d with respect to the inner product is the codifferential operator ${B_d}^\dagger = {M_{d-1}}^{-1} {B_d}^\top M_d$. We can again identify classical notions: B_1 as the gradient, ${B_1}^\dagger$ as the divergence, and B_2 as the curl.

The Dirichlet energy is given by

$$\left\langle {B_d}^\dagger f_d, {B_d}^\dagger h_d \right\rangle_{M_{d-1}} + \langle B_{d+1} f_d, B_{d+1} h_d \rangle_{M_{d+1}} = \langle f_d, L_d h_d \rangle_{M_d}, \tag{2.4}$$

which defines the Laplacian as the second-order differential operator $L_d = B_d {B_d}^\dagger + {B_{d+1}}^\dagger B_{d+1}$. We can again recognize the Laplace–de Rham operator from Hodge–de Rham theory.

Apart from yielding the Laplacian, the Dirichlet energy is useful by itself as a measure of variation or smoothness. Writing it as

$$E(f_d) = \langle f_d, L_d f_d \rangle_{M_d} = \left\| {B_d}^\dagger f_d \right\|^2_{M_{d-1}} + \|B_{d+1} f_d\|^2_{M_{d+1}} \tag{2.5}$$

shows that it takes into account two notions of smoothness, across down $(d-1)$ and up $(d+1)$ relations. For a 1-cochain f_1, the energy looks at the divergence 0-cochain ${B_1}^\dagger f_1$

and the curl 2-cochain $B_2 f_1$. Figure 2.3 shows an example for a 0-cochain. One could also write $E(f_d) = \|f_d\|^2_{M_d L_d} = \|M_d L_d f_d\|_2^2$, using the fact that all finite-dimensional Hilbert spaces are isomorphic, including Euclidean space. Because the Laplacian is positive semi-definite, the Dirichlet energy is a semi-norm, i.e., it follows the axioms of a norm but there exists cochains $f_d \neq 0$ such that $E(f_d) = 0$. Such d-cochains are in the nullspace of the Laplacian, and are actually indicators of d-dimensional holes. For $d = 0$, they are the 0-cochains that are constant across connected components.

The set of Laplacians $\{L_d\}$ records both the incidence and inner product structures, and captures the topology and geometry of the space. It is also a useful representation to compute topological properties. For example, the number of zero-eigenvalues of L_d corresponds to the number of d-dimensional holes in the simplicial complex.

We recover the graph Laplacian $L_0 = {B_1}^\dagger B_1 = {M_0}^{-1} {B_1}^\top M_1 B_1$ (as $B_0 = 0$) and its standard variants: the combinatorial Laplacian with edge weights $M_1 = W$ and without vertex weights, i.e., $M_0 = I$. The normalized symmetric Laplacian with $M_0 = I$ and $M_1 = D^{-1/2} W D^{-1/2}$, where D is the diagonal degree matrix such that $D_{ii} = \sum_j W_{ij}$. The random-walk Laplacian with $M_0 = D$ and $M_1 = W$.

2.2 Generalized convolutions

In the previous section, we built the Laplacian L_d from the fundamentals of space—its topology and geometry. In this section, we will study its properties with respect to the symmetries of that space. We will build a convolution operator that (i) enables efficient learning and (ii) that is equivariant to symmetry actions. Note that we took a constructive approach rather than a deductive one (say from a desired behavior with respect to symmetries) because symmetries are not necessarily known.

To simplify our discussion, we will focus on the most common case: graphs with weighted edges and unweighted vertices (i.e., $M_0 = I$). It allows to lighten the notation by dropping the dimension subscript and write $B = B_1$, $M = M_1$, $L = L_0 = B^\dagger B = B^\top M B$, and $n = |K_0|$. The discussion however translates to higher-dimensional simplices and non-identity metrics.

2.2.1 Symmetries

A symmetry is an automorphism σ, i.e., a map from the space to itself that preserves its structure. For a graph G, it is a permutation of its vertices that preserves edges and non-edges. The set of automorphisms forms a group, the graph's automorphism group

$\mathrm{Aut}(G)$. It is a subgroup of the symmetric group S_n, the group of all permutations of n elements. All in all, we have $\sigma \in \mathrm{Aut}(G) \subset S_n$. The number of symmetries, $|\mathrm{Aut}(G)|$, is anywhere between 0 (for asymmetric graphs) and $|S_n| = n!$ (for empty and complete graphs). In the spatial basis, the group representation is the permutation matrix P_σ, whose entry $(P_\sigma)_{ij}$ is 1 if a vertex is moved from the j^{th} to the i^{th} position and is 0 otherwise. It acts as $P_\sigma f$ on data (chains and cochains) by moving them across vertices.

While any permutation $\sigma \in S_n$ induces a change-of-basis (because vertex order is arbitrary, see §2.1.3), they are not necessarily symmetries. Similarly, while any reorientation of the simplices induces a change-of-basis, they are not symmetries. Both are artifacts of the algebraic representation.

2.2.2 Laplacian and equivariance

Because a symmetry preserves the adjacency structure by definition, its action on the Laplacian leaves it unchanged, i.e., ${P_\sigma}^\mathsf{T} L P_\sigma = L$. In other words, the Laplacian commutes with symmetry group actions, i.e., $LP_\sigma = P_\sigma L$. It is well-known that the Laplacian is, by construction as an intrinsic operator, an equivariant operator (see for example the discussion in [167]).

As symmetries leave L unchanged, they must act as rotations within its eigenspaces. The basis that diagonalizes the Laplacian as $L = U\Lambda U^{-1}$ hence jointly block-diagonalizes the symmetry group actions—without knowing them. Figure 2.4 illustrates it.

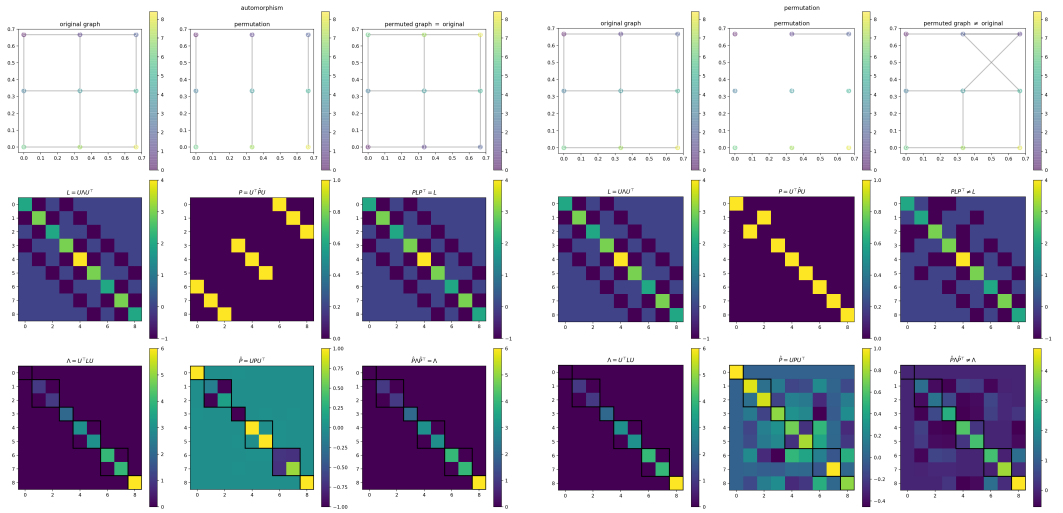
Because permutation groups are compact and Abelian, the result that the Fourier transform diagonalizes the action of symmetry groups is a special case of the Peter-Weyl theorem [134] and Pontryagin duality.

2.2.3 Spectral basis

We call the change-of-bases U and U^{-1} the graph Fourier transform (GFT) because it is reminiscent of the classical Fourier transform [155]. Indeed, we get the discrete cosine transform (DCT) from a path graph and the discrete Fourier transform (DFT) from a cycle (ring) graph. In the same way the transform was historically invented to simplify the study of heat diffusion by diagonalizing the heat kernel, it simplifies our study by (block-)diagonalizing our operators of interest.

Without loss of generality, we assume that the matrices $\Lambda = \mathrm{diag}(\lambda_1, \dots, \lambda_n)$ and $U = [u_1, \dots, u_n]$ are assembled such that the eigenvalues are ordered as $0 = \lambda_1 \leq \dots \leq \lambda_n$[7]. The eigenvalue λ_i is associated to the eigenvector u_i.

[7]The smallest eigenvalue is equal to zero because the graph is made of at least one connected component.



(a) A permutation that is an automorphism, i.e., $PLP^\mathsf{T} = L$.

(b) A permutation that is not an automorphism, i.e., $PLP^\mathsf{T} \neq L$.

Figure 2.4: Two permutations of a 3×3 grid graph ($n = 9$ vertices). Top: an embedding of the initial graph, permutation, and permuted graph with ordered vertices for illustration. Middle: initial Laplacian L, permutation matrix P, and permuted Laplacian PLP^T in the spatial basis. Bottom: initial Laplacian Λ, permutation matrix Π, and permuted Laplacian $\Pi\Lambda\Pi^\mathsf{T}$ in the spectral basis, whose eigen-subspaces are highlighted by boxes. Permutations that are automorphisms act within the Laplacian's eigenspaces as roto-translations (a); permutations that are not automorphisms do not (b). By definition and construction, permutations P that are symmetry actions preserve and commute with L.

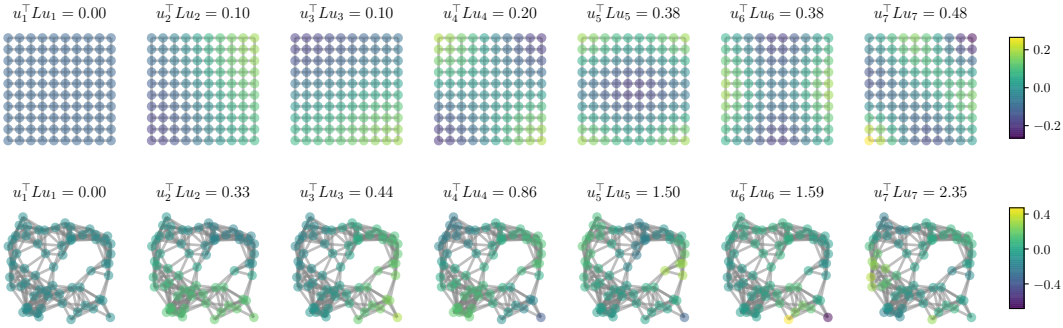


Figure 2.5: Eigenvectors that form the spectral (Fourier) basis.

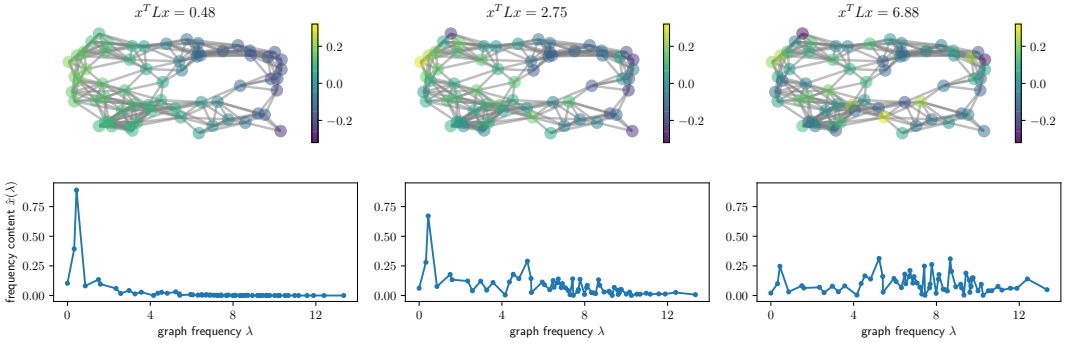


Figure 2.6: Data in the spatial and spectral bases. Spatial: the data x is shown as x_i at vertex v_i. Spectral: the data $\hat{x} = U^{-1}x$ is shown as $(\lambda_i, \hat{x}_i)$. The smoother the data, measured by the Dirichlet energy $E(x) = x^\mathsf{T} Lx = \hat{x}^\mathsf{T} \Lambda \hat{x}$, the more concentrated in the lower frequencies is the energy.

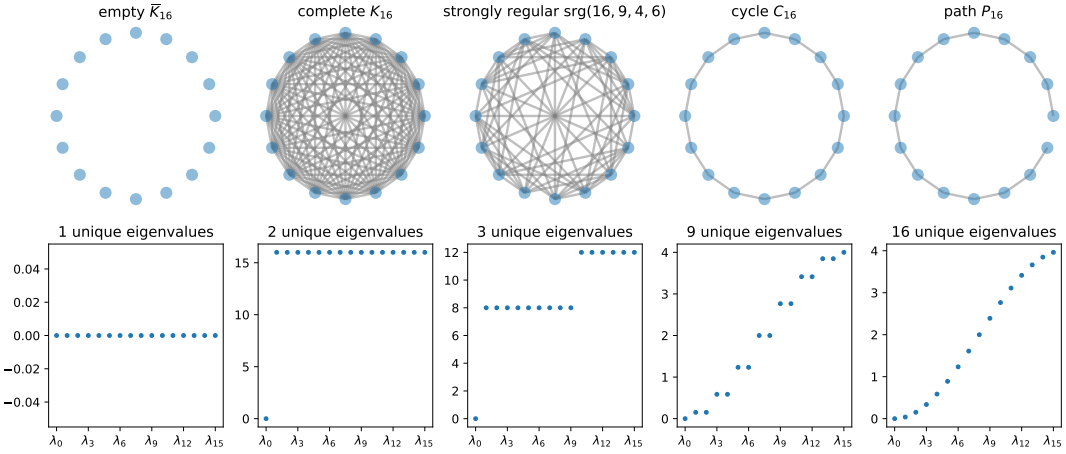


Figure 2.7: Eigenvalues of graph Laplacians. All graphs are made of $n = 16$ vertices but their connectivity yields very different eigenspaces.

The eigenspaces are the subspaces that are spanned by eigenvectors that share the same eigenvalue. Note that the basis chosen by the eigensolver within each eigenspace is arbitrary and up to rotation and reflection. For example, u_i and $-u_i$ are two equally valid choices.

The number of eigenspaces is the number of distinct eigenvalues, and their dimensionality is the multiplicity of the associated eigenvalue. At the extremes, the spectral basis of path graphs is made of n eigenspaces of dimensionality 1 while that of empty graphs is made of a single eigenspace of dimensionality n. See Figure 2.7. While it is known that a graph is strongly regular if and only if it has 3 distinct eigenvalues, and that the number of eigenspaces for paths and cycles correspond to the classical DCT and DFT, I do not know of and could not find a general rule. That is a direction for future research.

2.2.4 Generalized convolutions

The spectral basis is decomposed into eigenspaces, which are the invariant subspaces of both the Laplacian and symmetry actions. It hence suffices to look at their action in these subspaces. The representation of permutations in the spectral basis is the block-diagonal matrix $\Pi_\sigma = U^{-1}P_\sigma U$, with one block per eigenspace. Each block implements a roto-reflection.

The unique operation that is orthogonal to roto-reflections is scaling: a multiplication operator. As each eigenspace is associated with a distinct eigenvalue, we can implement the multiplication operator without loss of generality as a kernel evaluated at the eigenvalues:

$$g(\Lambda) = \operatorname{diag}\big(g(\lambda_1), \dots, g(\lambda_n)\big). \tag{2.6}$$

We easily verify that $g(\Lambda)\Pi_\sigma = \Pi_\sigma g(\Lambda)\ \forall\ g, \sigma$. The orthogonal actions of $g(\Lambda)$ and Π_σ are illustrated in Figure 2.11b. Any operator that is to commute with symmetry actions without knowing them must have the form of $g(\Lambda)$. While any permutation in the spatial basis is a rotation in the eigenspaces of the spectral basis, the converse is not true: most of these rotations are not representations of permutations, let alone symmetries. Hence, if symmetries are known, one can build more general equivariant operators that not only scale but also rotate.

In the spatial basis, the multiplication operator $g(\Lambda)$ becomes

$$g(L) = Ug(\Lambda)U^{-1}, \tag{2.7}$$

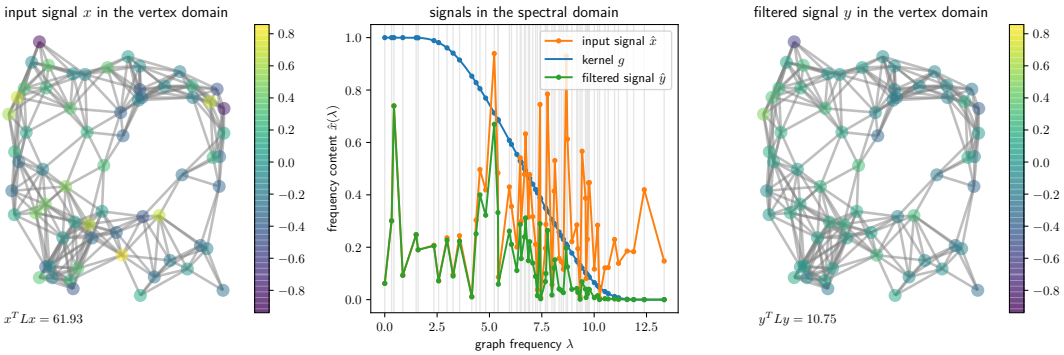


Figure 2.8: Filtering. Left: data x in the spatial basis. Middle: data $\hat{x} = U^{-1}x$, concrete filter $\mathrm{diag}(g(\Lambda))$, and filtered data $\hat{y} = g(\Lambda)\hat{x}$ in the spectral basis. Right: filtered data $y = U\hat{y}$ in the spatial basis. The low-pass filter smooths the data while respecting the domain's topology and geometry.

which we call the *generalized convolution operator*. I see two justifications for calling it a convolution: first, as in classical signal processing and harmonic analysis, the multiplication operator $g(\Lambda)$ in the spectral basis becomes the convolution operator $g(L)$ in the spatial basis. Second, it is an equivariant operator that commutes with symmetries, the defining property of convolutions. It is however a generalized convolution because it commutes with more than symmetries, although I could not (yet) precisely figure out what that operation might be.

We have used an inconsistent notation for the homomorphisms $\sigma \to P_\sigma$ and $g \to g(L)$ (and there equivalent in the spectral basis, $\sigma \to \Pi_\sigma$ and $g \to g(\Lambda)$). The first is standard in group theory; the second is standard in functional calculus. The group theory notation emphasizes that the representation of σ is a permutation matrix, while the functional calculus notation emphasizes that $g(L)$ depends on L. Indeed, $g(L)$ is a concrete representation—on a space specified by L—of the abstract kernel g. The matrix function notation makes sense because L is diagonalizable, hence the kernel g can be evaluated at its eigenvalues Λ. That is the continuous functional calculus, a generalization of the polynomial functional calculus and a special case of the Borel functional calculus.

2.2.5 Filtering

The most obvious use of the generalized convolution $g(L)$ is to linearly transform data while respecting the space they lie on, known as filtering in signal processing. Figure 2.8 shows a filtering operation $y = g(L)x$ decomposed as the forward Fourier transform U^{-1}, multiplication $g(\Lambda)$ in the spectral basis, and backward transform U to the spatial

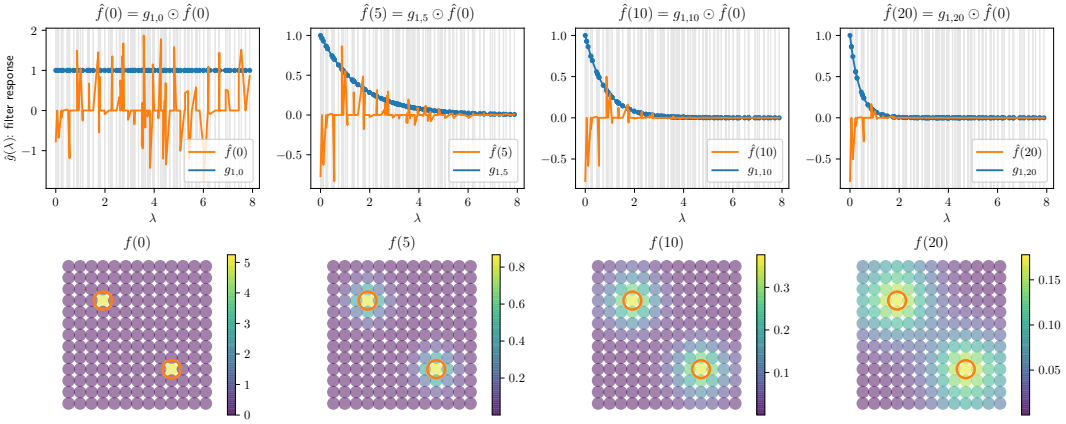


Figure 2.9: Heat diffusion. Solving the partial differential equation $-\tau L f(t) = \partial_t f(t)$ as $f(t) = g_{\tau t}(L) f(0)$, where $f(0)$ is the initial heat distribution and $g_{\tau t}(\lambda) = \exp(-\tau t \lambda)$ is the heat kernel. Top: process in the spectral basis. Bottom: process in the spatial basis.

domain. The kernel g can be designed to perform known linear transformations, for example to remove noise from data or solve partial differential equations. Figure 2.9 shows the diffusion of heat at different times t as an initial heat distribution that is filtered by the heat kernel $g_{\tau t}(\lambda) = \exp(-\tau t \lambda)$.

While P_σ, Π_σ, and $g(\Lambda)$ are rather simple operators, depending on the complexity of the space, $g(L)$ might not be. By looking at the weak form $\langle y, \delta_i \rangle = \langle g(L)x, \delta_i \rangle = \langle x, g(L)\delta_i \rangle$, we observe that the result of the convolution at the i^{th} vertex is an inner product of the data with $g(L)\delta_i$. We recognize here the form of a classic convolution, which is an inner product with a kernel that is moved around by group actions like translations. We will say that $g(L)\delta_i$ is the kernel g *localized* at the i^{th} vertex. Figure 2.10 shows the localization of the heat kernel on two graphs: a vertex-transitive one and an asymmetric one. Localization is a generalization of symmetry action to non-homogeneous spaces. It is important to note that, contrary to the classical case where two signals are convolved through symmetry group actions, the kernel g and the data are objects of a different nature.

2.2.6 Spectral embedding

We can write

$$E_g(f) = \langle f, g(L)f \rangle = \left\| g^{1/2}(\Lambda) U^{-1} f \right\|_2^2 \tag{2.8}$$

as a generalization of the Dirichlet energy (2.4) to $g \neq \mathrm{id}$ and view $g^{1/2}(\Lambda)U^{-1}f$ as an embedding of f in Euclidean space. By construction, that embedding reproduces the

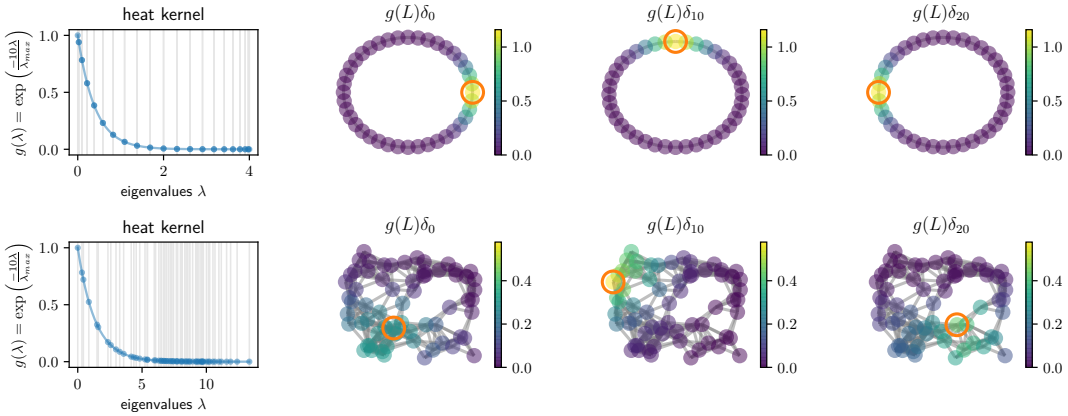


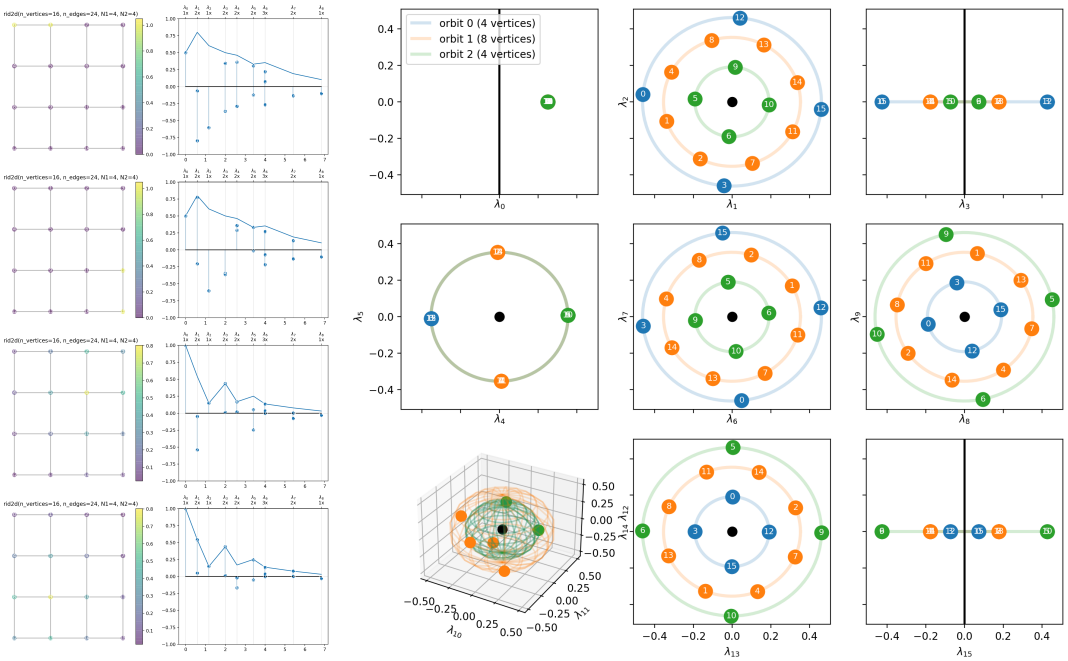
Figure 2.10: Localization. On homogeneous spaces (e.g., vertex-transitive graphs), symmetry actions suffice to move convolution kernels anywhere. On non-homogeneous and asymmetric spaces however, we require another operation: localization. When the space has symmetries, the localization of a convolution kernel reduces to symmetry actions.

symmetries of the space encoded in L and a notion of distance set by g. See Figure 2.11. From that energy and semi-norm, we also get the distance $E_g(f-h)$ between f and h.

Viewing the set $\{q_i\}_{i=1}^N$ of N embedded cochains $q_i = g^{1/2}(\Lambda)U^{-1}f_i$ as particles in Euclidean space, we define by $J(q) = \Sigma_i \|q - q_i\|_2^2$ the moment of inertia at a point q. The moment of inertia at the center of mass, the origin, is $J(0) = \Sigma_i \|q_i\|^2$. The moment of inertia at the i^{th} cochain is $J(q_i) = \Sigma_j \|q_j - q_i\|_2^2 = \Sigma_j \|q_j\|_2^2 + \|q_i\|_2^2 - 2\langle q_i, q_j\rangle = J(0) + N\|q_i\|_2^2 \propto \|q_i\|_2^2 = \|g^{1/2}(\Lambda)U^{-1}f_i\|_2^2 = E_g(f_i)$. The energy E_g is hence a measure of centrality under the notion of distance set by g: the closest a cochain is to all other cochains, the closest it must be to the origin.

A special case of interest, for the extraction of structural features for downstream learning tasks, is the embedding of indicator 0-cochains δ_i. The vertices of a graph are embedded as $Q = [q_1, \dots, q_n] = g^{1/2}(\Lambda)U^{-1}$ and the centrality of the i^{th} vertex is given by $E_g(\delta_i) = \big(g(L)\big)_{ii}$. In that embedding, a well-connected vertex is literally centered. The moment of inertia at the origin $J(0) = \Sigma_i \|q_i\|_2^2 = \frac{1}{2n}\Sigma_{i,j}\|q_i - q_j\|_2^2$ measures the variance in centrality, hence the regularity of the g-modulated graph. $J(0) = \Sigma_k g(\lambda_k)$ is indeed the total variance of the embedding, as seen from the full covariance $Q^\top Q = Ug(\Lambda)U^{-1} = g(L)$. The g-modulated graph (hence energy and distance) can be approximated from a lower-rank embedding $\Sigma_{k\in\mathcal{K}} g^{1/2}(\lambda_k)u_k{}^\top \cong Q$ that captures $\frac{\Sigma_{k\in\mathcal{K}} g(\lambda_k)}{\Sigma_k g(\lambda_k)}$ of the variance.

Multiple methods to embed graphs or measure centrality can be mapped in this framework. As demonstrated in Figure 2.12, which shows the embedding of vertex indicator



(a) Cochain embedding. Left: data in the spatial basis. Right: data in the spectral basis. The first two and the last two have the same power spectral density (PSD), i.e., the same energy in each eigenspace, because they are equivalent under symmetry action.

(b) Vertex embedding. Black dots and lines show the origin. Colored dots numbered i show the spectral embedding $g^{1/2}(\Lambda)U^{-1}\delta_i$ of the i^{th} vertex represented by the indicator 0-chain δ_i. The action of g, represented by the multiplication operator $g(\Lambda)$, independently scales the eigenspaces, while the orthogonal action of symmetries σ, represented by Π_σ, rotates and reflects. The 2D embedding in (a) makes it evident that the 4×4 grid is made of 3 equivalent classes of vertices, represented by 3 colors and linked by symmetry actions on colored orbits.

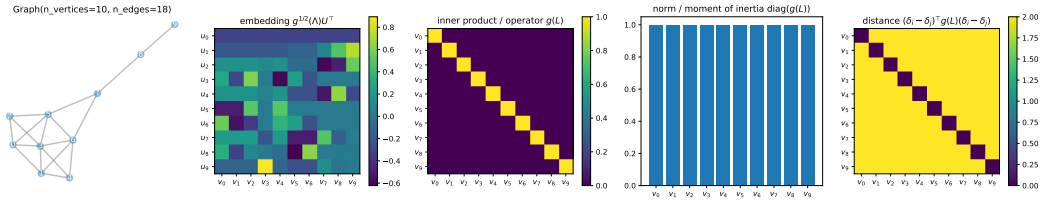
Figure 2.11: Spectral embedding of some 0-cochains on a 4×4 grid graph ($n = 16$ vertices). The 16-dimensional spectral basis of that graph is decomposed into 9 eigenspaces made of 1 to 3 eigenvectors.

cochains under different choices of g, there is no single notion of centrality [101]. We already saw that the Dirichlet energy was a special case with $g(\lambda) = \lambda$. That corresponds to the degree of a vertex as its measure of centrality, while the pseudo-inverse $g(\lambda) = \lambda^+$ corresponds to the resistance [95] or commute-time [66] distance [58]. Laplacian eigenmaps [17] is obtained with $g(\lambda) = 1$. Diffusion kernels [99] and diffusion maps [37] are obtained with the heat kernel $g(\lambda) = e^{-t\lambda}$. In the computer graphics literature, the heat kernel signature (HKS) [161] and the wave kernel signature (WKS) [7] were proposed to extract features for the analysis of shapes. $g(\lambda) = (a - \lambda)^p, a \geq \lambda_n$ corresponds to the p-steps random-walk used in PageRank [126].

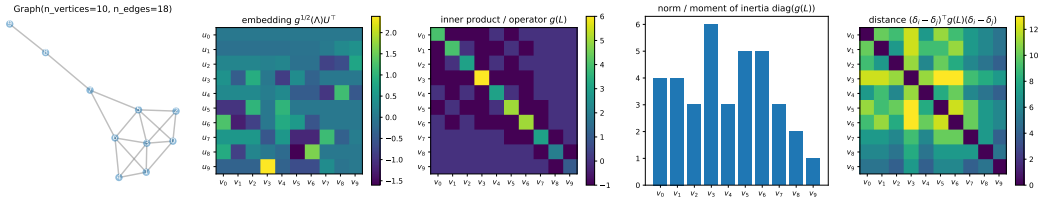
2.3 Conclusion

The kernel g defines a notion of distance. Its concrete representation, the generalized convolution $g(L) = g(B^\top MB)$, only depends on the space's topology B and geometry M, and is constrained by the symmetries and complexity of that space. Fulfilling our goal of constraining the functional space to learn from, $g(L)$ is a linear operator that is mostly constrained by the properties of the space: while an unconstrained linear operator would have n^2 degrees of freedom, $g(L)$ has from 1 to n degrees.

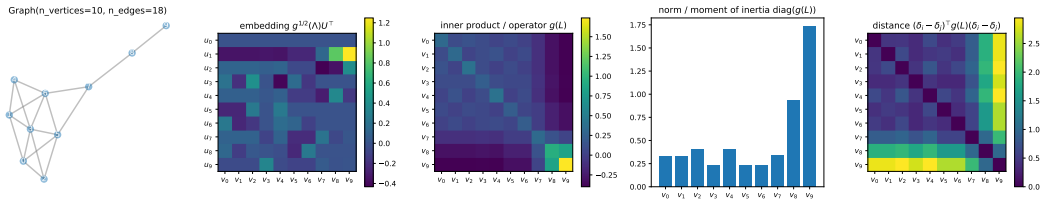
Let me close this chapter by emphasizing that filtering and embedding are one and the same operation—a generalized convolution. They are only different points of view: filtering emphasizes the linear transformation of data. Embedding emphasizes the extraction of features from the space. For both purposes, the kernel g can be designed if one knows what one wants, or learned from data if one does not.



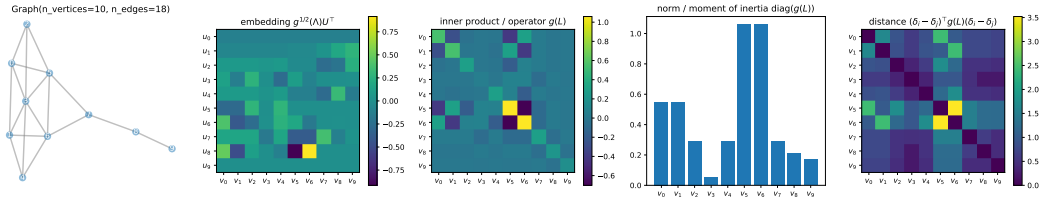
(a) $g(\lambda) = 1$ yields the Laplacian eigenmaps embedding.



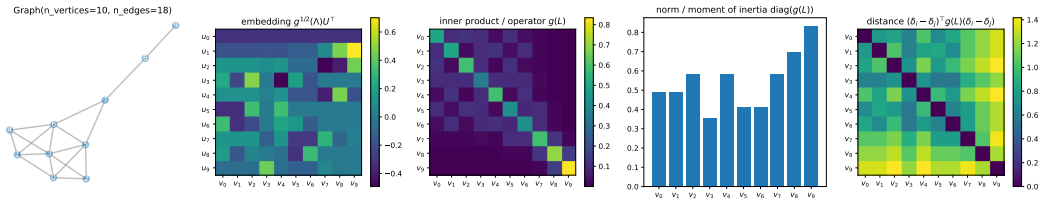
(b) $g(\lambda) = \lambda$ yields the degree centrality measure and Dirichlet energy.



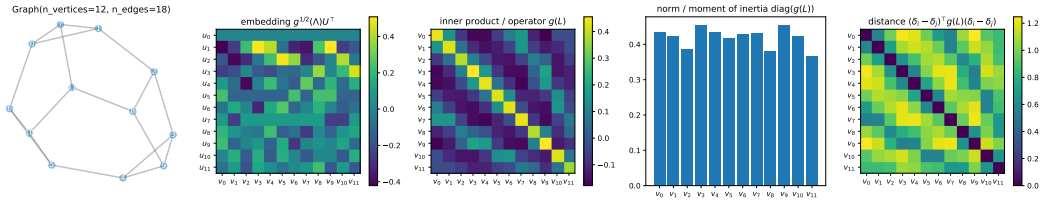
(c) $g(\lambda) = \lambda^+ \approx \lambda^{-1}$ (pseudo-inverse) yields the resistance (or commute-time) distance.



(d) $g(\lambda) = (a-\lambda)^p, a \geq \lambda_n$, yields the p-steps random-walk distance.



(e) $g(\lambda) = e^{-t\lambda}$ yields the diffusion distance and heat kernel signature (HKS) embedding.



(f) $g(\lambda) = \lambda^+$ on the Frucht graph.

Figure 2.12: Embedding and centrality of vertices, along inner product and distance between them. Different choices of g yields different notions. The Krackhardt kite [101] graph demonstrates that there is no single notion of centrality as multiple vertices qualify as central (a-e), while the asymmetric Frucht graph [60] is more regular and hardly has a central vertex (f).

3 Learning from structured data

In this work, we are interested in generalizing convolutional neural networks (CNNs) from low-dimensional regular grids, where image, video and speech are represented, to high-dimensional irregular domains, such as social networks, brain connectomes or words' embedding, represented by graphs. We present a formulation of CNNs in the context of spectral graph theory, which provides the necessary mathematical background and efficient numerical schemes to design fast localized convolutional filters on graphs. Importantly, the proposed technique offers the same linear computational complexity and constant learning complexity as classical CNNs, while being universal to any graph structure. Experiments on MNIST and 20NEWS demonstrate the ability of this novel deep learning system to learn local, stationary, and compositional features on graphs.

3.1 Introduction

Convolutional neural networks [106] offer an efficient architecture to extract highly meaningful statistical patterns in large-scale and high-dimensional datasets. The ability of CNNs to learn local stationary structures and compose them to form multi-scale hierarchical patterns has led to breakthroughs in image, video, and sound recognition tasks [105]. Precisely, CNNs extract the local stationarity property of the input data or signals by revealing local features that are shared across the data domain. These similar features are identified with localized convolutional filters or kernels, which are learned from the data. Convolutional filters are shift- or translation-invariant filters, meaning they are able to recognize identical features independently of their spatial locations. Localized kernels or compactly supported filters refer to filters that extract local features independently of the input data size, with a support size that can be much smaller than the input size.

User data on social networks, gene data on biological regulatory networks, log data on telecommunication networks, or text documents on word embeddings are important examples of data lying on irregular or non-Euclidean domains that can be structured

with graphs, which are universal representations of heterogeneous pairwise relationships. Graphs can encode complex geometric structures and can be studied with strong mathematical tools such as spectral graph theory [30].

A generalization of CNNs to graphs is not straightforward as the convolution and pooling operators are only defined for regular grids. This makes this extension challenging, both theoretically and implementation-wise. The major bottleneck of generalizing CNNs to graphs, and one of the primary goals of this work, is the definition of localized graph filters which are efficient to evaluate and learn. Precisely, the main contributions of this work are summarized below.

1. **Spectral formulation.** A spectral graph theoretical formulation of CNNs on graphs built on established tools in graph signal processing (GSP) [153].

2. **Strictly localized filters.** Enhancing [25], the proposed spectral filters are provable to be strictly localized in a ball of radius K, i.e., K hops from the central vertex.

3. **Low computational complexity.** The evaluation complexity of our filters is linear with respect to the filters support's size K and the number of edges $|\mathcal{E}|$. Importantly, as most real-world graphs are highly sparse, we have $|\mathcal{E}| \ll n^2$ and $|\mathcal{E}| = kn$ for the widespread k-nearest neighbor (NN) graphs, leading to a linear complexity with respect to the input data size n. Moreover, this method avoids the Fourier basis altogether, thus the expensive eigenvalue decomposition (EVD) necessary to compute it as well as the need to store the basis, a matrix of size n^2. That is especially relevant when working with limited GPU memory. Besides the data, our method only requires to store the Laplacian, a sparse matrix of $|\mathcal{E}|$ non-zero values.

4. **Efficient pooling.** We propose an efficient pooling strategy on graphs which, after a rearrangement of the vertices as a binary tree structure, is analog to pooling of 1D signals.

5. **Experimental results.** We present multiple experiments that ultimately show that our formulation is (i) a useful model, (ii) computationally efficient and (iii) superior both in accuracy and complexity to the pioneer spectral graph CNN introduced in [25]. We also show that our graph formulation performs similarly to a classical CNNs on MNIST and study the impact of various graph constructions on performance. The TensorFlow [1] code to reproduce our results and apply the model to other data is available as an open-source software.[1]

[1] https://github.com/mdeff/cnn_graph

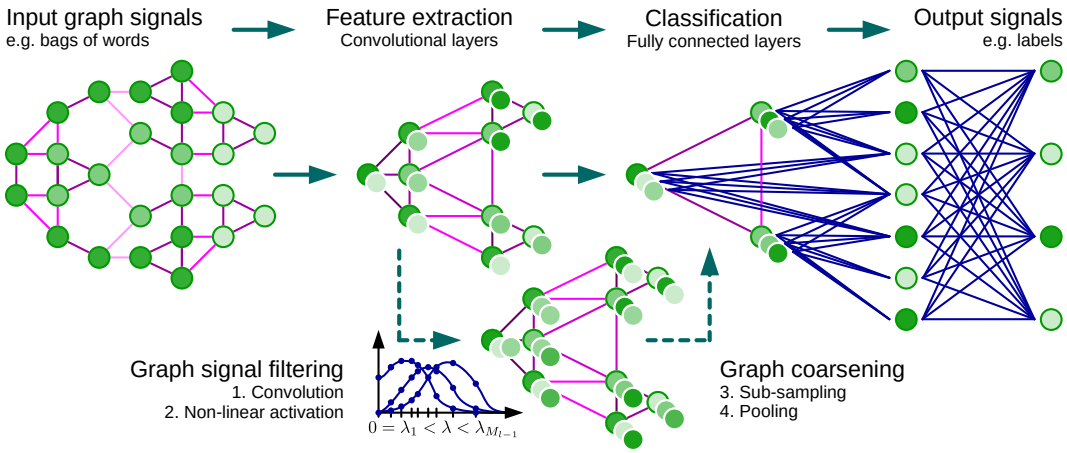


Figure 3.1: Architecture of a CNN on graphs and the four ingredients of a (graph) convolutional layer.

3.2 Proposed technique

Generalizing CNNs to graphs requires three fundamental steps: (i) the design of localized convolutional filters on graphs, (ii) a graph coarsening procedure that groups together similar vertices and (iii) a graph pooling operation that trades spatial resolution for higher filter resolution.

3.2.1 Learning fast localized spectral filters

There are two strategies to define convolutional filters; either from a spatial approach or from a spectral approach. By construction, spatial approaches provide filter localization via the finite size of the kernel. However, although graph convolution in the spatial domain is conceivable, it faces the challenge of matching local neighborhoods, as pointed out in [25]. Consequently, there is no unique mathematical definition of translation on graphs from a spatial perspective. On the other side, a spectral approach provides a well-defined localization operator on graphs via convolutions with a Kronecker delta implemented in the spectral domain [153]. The convolution theorem [115] defines convolutions as linear operators that diagonalize in the Fourier basis (represented by the eigenvectors of the Laplacian operator). However, a filter defined in the spectral domain is not naturally localized and translations are costly due to the $\mathcal{O}(n^2)$ multiplication with the graph Fourier basis. Both limitations can however be overcome with a special choice of filter parametrization.

GRAPH FOURIER TRANSFORM. We are interested in processing signals defined on undirected and connected graphs $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$, where $\mathcal{V}$ is a finite set of $|\mathcal{V}| = n$ vertices, $\mathcal{E}$ is a set of edges and $W \in \mathbb{R}^{n\times n}$ is a weighted adjacency matrix encoding the connection weight between two vertices. A signal $x : \mathcal{V} \to \mathbb{R}$ defined on the nodes of the graph may be regarded as a vector $x \in \mathbb{R}^n$ where x_i is the value of x at the i^{th} node. An essential operator in spectral graph analysis is the graph Laplacian [30], which combinatorial definition is $L = D - W \in \mathbb{R}^{n\times n}$ where $D \in \mathbb{R}^{n\times n}$ is the diagonal degree matrix with $D_{ii} = \sum_j W_{ij}$, and normalized definition is $L = I_n - D^{-1/2}WD^{-1/2}$ where I_n is the identity matrix. As L is a real symmetric positive semidefinite matrix, it has a complete set of orthonormal eigenvectors $\{u_l\}_{l=0}^{n-1} \in \mathbb{R}^n$, known as the graph Fourier modes, and their associated ordered real nonnegative eigenvalues $\{\lambda_l\}_{l=0}^{n-1}$, identified as the frequencies of the graph. The Laplacian is indeed diagonalized by the Fourier basis $U = [u_0, \dots, u_{n-1}] \in \mathbb{R}^{n\times n}$ such that $L = U\Lambda U^\top$ where $\Lambda = \mathrm{diag}([\lambda_0, \dots, \lambda_{n-1}]) \in \mathbb{R}^{n\times n}$. The graph Fourier transform of a signal $x \in \mathbb{R}^n$ is then defined as $\hat{x} = U^\top x \in \mathbb{R}^n$, and its inverse as $x = U\hat{x}$ [153]. As on Euclidean spaces, that transform enables the formulation of fundamental operations such as filtering.

SPECTRAL FILTERING OF GRAPH SIGNALS. As we cannot express a meaningful translation operator in the vertex domain, the convolution operator on graph $*_\mathcal{G}$ is defined in the Fourier domain such that $x *_\mathcal{G} y = U\big((U^\top x) \odot (U^\top y)\big)$, where $\odot$ is the element-wise Hadamard product. It follows that a signal x is filtered by g_θ as

$$y = g_\theta(L)x = g_\theta(U\Lambda U^\top)x = Ug_\theta(\Lambda)U^\top x. \tag{3.1}$$

A non-parametric filter, i.e., a filter whose parameters are all free, would be defined as

$$g_\theta(\Lambda) = \mathrm{diag}(\theta), \tag{3.2}$$

where the parameter $\theta \in \mathbb{R}^n$ is a vector of Fourier coefficients.

POLYNOMIAL PARAMETRIZATION FOR LOCALIZED FILTERS. There are however two limitations with non-parametric filters: (i) they are not localized in space and (ii) their learning complexity is in $\mathcal{O}(n)$, the dimensionality of the data. These issues can be overcome with the use of a polynomial filter

$$g_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k \Lambda^k, \tag{3.3}$$

where the parameter $\theta \in \mathbb{R}^K$ is a vector of polynomial coefficients. The value at vertex j of the filter g_θ centered at vertex i is given by $\left(g_\theta(L)\delta_i\right)_j = \left(g_\theta(L)\right)_{i,j} = \sum_k \theta_k \left(L^k\right)_{i,j}$, where the kernel is localized via a convolution with a Kronecker delta function $\delta_i \in \mathbb{R}^n$. By [74, Lemma 5.2], $d_\mathcal{G}(i,j) > K$ implies $\left(L^K\right)_{i,j} = 0$, where $d_\mathcal{G}$ is the shortest path distance, i.e., the minimum number of edges connecting two vertices on the graph. Consequently, spectral filters represented by K^{th}-order polynomials of the Laplacian are exactly K-localized. Besides, their learning complexity is $\mathcal{O}(K)$, the support size of the filter, and thus the same complexity as classical CNNs.

RECURSIVE FORMULATION FOR FAST FILTERING. While we have shown how to learn localized filters with K parameters, the cost to filter a signal x as $y = U g_\theta(\Lambda) U^\mathsf{T} x$ is still high with $\mathcal{O}(n^2)$ operations because of the multiplication with the Fourier basis U. A solution to this problem is to parametrize $g_\theta(L)$ as a polynomial function that can be computed recursively from L, as K multiplications by a sparse L costs $\mathcal{O}(K|\mathcal{E}|) \ll \mathcal{O}(n^2)$. One such polynomial, traditionally used in GSP to approximate kernels (like wavelets), is the Chebyshev expansion [74]. Another option, the Lanczos algorithm [162], which constructs an orthonormal basis of the Krylov subspace $\mathcal{K}_K(L,x) = \text{span}\{x, Lx, \dots, L^{K-1}x\}$, seems attractive because of the coefficients' independence. It is however more convoluted and thus left as a future work.

Recall that the Chebyshev polynomial $T_k(x)$ of order k may be computed by the stable recurrence relation $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$ with $T_0 = 1$ and $T_1 = x$. These polynomials form an orthogonal basis for $L^2([-1,1], dy/\sqrt{1-y^2})$, the Hilbert space of square integrable functions with respect to the measure $dy/\sqrt{1-y^2}$. A filter can thus be parametrized as the truncated expansion

$$g_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{\Lambda}), \tag{3.4}$$

of order $K-1$, where the parameter $\theta \in \mathbb{R}^K$ is a vector of Chebyshev coefficients and $T_k(\tilde{\Lambda}) \in \mathbb{R}^{n\times n}$ is the Chebyshev polynomial of order k evaluated at $\tilde{\Lambda} = 2\Lambda/\lambda_{\max} - I_n$, a diagonal matrix of scaled eigenvalues that lie in $[-1,1]$. The filtering operation can then be written as $y = g_\theta(L)x = \sum_{k=0}^{K-1} \theta_k T_k(\tilde{L})x$, where $T_k(\tilde{L}) \in \mathbb{R}^{n\times n}$ is the Chebyshev polynomial of order k evaluated at the scaled Laplacian $\tilde{L} = 2L/\lambda_{\max} - I_n$. Denoting $\bar{x}_k = T_k(\tilde{L})x \in \mathbb{R}^n$, we can use the recurrence relation to compute $\bar{x}_k = 2\tilde{L}\bar{x}_{k-1} - \bar{x}_{k-2}$ with $\bar{x}_0 = x$ and $\bar{x}_1 = \tilde{L}x$. The entire filtering operation $y = g_\theta(L)x = [\bar{x}_0, \dots, \bar{x}_{K-1}]\theta$ then costs $\mathcal{O}(K|\mathcal{E}|)$ operations.

LEARNING FILTERS. The j^{th} output feature map of the sample s is given by

$$y_{s,j} = \sum_{i=1}^{F_{\text{in}}} g_{\theta_{i,j}}(L) x_{s,i} \in \mathbb{R}^n, \tag{3.5}$$

where the $x_{s,i}$ are the input feature maps and the $F_{\text{in}} \times F_{\text{out}}$ vectors of Chebyshev coefficients $\theta_{i,j} \in \mathbb{R}^K$ are the layer's trainable parameters. When training multiple convolutional layers with the backpropagation algorithm, one needs the two gradients

$$\frac{\partial E}{\partial \theta_{i,j}} = \sum_{s=1}^{S} [\bar{x}_{s,i,0}, \dots, \bar{x}_{s,i,K-1}]^{\mathsf{T}} \frac{\partial E}{\partial y_{s,j}} \qquad \text{and} \qquad \frac{\partial E}{\partial x_{s,i}} = \sum_{j=1}^{F_{\text{out}}} g_{\theta_{i,j}}(L) \frac{\partial E}{\partial y_{s,j}}, \tag{3.6}$$

where E is the loss energy over a mini-batch of S samples. Each of the above three computations boils down to K sparse matrix-vector multiplications and one dense matrix-vector multiplication for a cost of $\mathcal{O}(K|\mathcal{E}|F_{\text{in}}F_{\text{out}}S)$ operations. These can be efficiently computed on parallel architectures by leveraging tensor operations. Eventually, $[\bar{x}_{s,i,0}, \dots, \bar{x}_{s,i,K-1}]$ only needs to be computed once.

3.2.2 GRAPH COARSENING

The pooling operation requires meaningful neighborhoods on graphs, where similar vertices are clustered together. Doing this for multiple layers is equivalent to a multi-scale clustering of the graph that preserves local geometric structures. It is however known that graph clustering is NP-hard [27] and that approximations must be used. While there exist many clustering techniques, e.g., the popular spectral clustering [114], we are most interested in multilevel clustering algorithms where each level produces a coarser graph which corresponds to the data domain seen at a different resolution. Moreover, clustering techniques that reduce the size of the graph by a factor two at each level offers a precise control on the coarsening and pooling size. In this work, we make use of the coarsening phase of the Graclus multilevel clustering algorithm [50], which has been shown to be extremely efficient at clustering a large variety of graphs. Algebraic multigrid techniques on graphs [143] and the Kron reduction [154] are two methods worth exploring in future works.

Graclus [50], built on Metis [91], uses a greedy algorithm to compute successive coarser versions of a given graph and is able to minimize several popular spectral clustering objectives, from which we chose the normalized cut [152]. Graclus' greedy rule consists, at each coarsening level, in picking an unmarked vertex i and matching it with one of its unmarked neighbors j that maximizes the local normalized cut $W_{ij}(1/d_i + 1/d_j)$. The

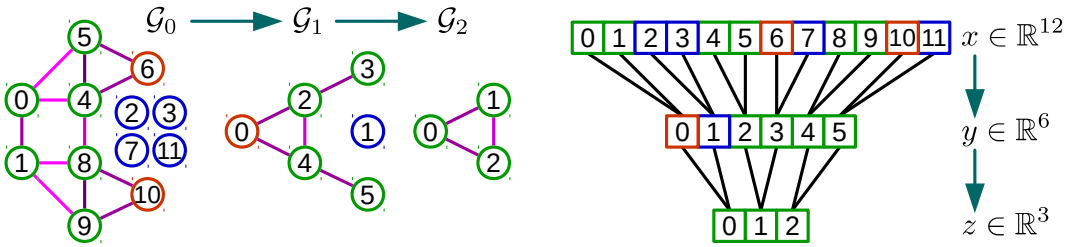


Figure 3.2: **Example of Graph Coarsening and Pooling.** Let us carry out a max pooling of size 4 (or two poolings of size 2) on a signal $x \in \mathbb{R}^8$ living on $\mathcal{G}_0$, the finest graph given as input. Note that it originally possesses $n_0 = |\mathcal{V}_0| = 8$ vertices, arbitrarily ordered. For a pooling of size 4, two coarsenings of size 2 are needed: let Graclus gives $\mathcal{G}_1$ of size $n_1 = |\mathcal{V}_1| = 5$, then $\mathcal{G}_2$ of size $n_2 = |\mathcal{V}_2| = 3$, the coarsest graph. Sizes are thus set to $n_2 = 3$, $n_1 = 6$, $n_0 = 12$ and fake nodes (in blue) are added to $\mathcal{V}_1$ (1 node) and $\mathcal{V}_0$ (4 nodes) to pair with the singeltons (in orange), such that each node has exactly two children Nodes in $\mathcal{V}_2$ are then arbitrarily ordered and nodes in $\mathcal{V}_1$ and $\mathcal{V}_0$ are ordered consequently. At that point the arrangement of vertices in $\mathcal{V}_0$ permits a regular 1D pooling on $x \in \mathbb{R}^{12}$ such that $z = [\max(x_0, x_1), \max(x_4, x_5, x_6), \max(x_8, x_9, x_{10})] \in \mathbb{R}^3$, where the signal components x_2, x_3, x_7, x_{11} are set to a neutral value.

two matched vertices are then marked and the coarsened weights are set as the sum of their weights. The matching is repeated until all nodes have been explored. This is an very fast coarsening scheme which divides the number of nodes by approximately two (there may exist a few singletons, non-matched nodes) from one level to the next coarser level.

3.2.3 Fast pooling of graph signals

Pooling operations are carried out many times and must be efficient. After coarsening, the vertices of the input graph and its coarsened versions are not arranged in any meaningful way. Hence, a direct application of the pooling operation would need a table to store all matched vertices. That would result in a memory inefficient, slow, and hardly parallelizable implementation. It is however possible to arrange the vertices such that a graph pooling operation becomes as efficient as a 1D pooling. We proceed in two steps: (i) create a balanced binary tree and (ii) rearrange the vertices. After coarsening, each node has either two children, if it was matched at the finer level, or one, if it was not, i.e the node was a singleton. From the coarsest to finest level, fake nodes, i.e., disconnected nodes, are added to pair with the singletons such that each node has two children. This structure is a balanced binary tree: (i) regular nodes (and singletons) either have two regular nodes (e.g., level 1 vertex 0 in Figure 3.2) or (ii) one singleton and a fake node as children (e.g., level 2 vertex 0), and (iii) fake nodes always have two fake nodes as chil-

dren (e.g., level 1 vertex 1). Input signals are initialized with a neutral value at the fake nodes, e.g., 0 when using a ReLU activation with max pooling. Because these nodes are disconnected, filtering does not impact the initial neutral value. While those fake nodes do artificially increase the dimensionality thus the computational cost, we found that, in practice, the number of singletons left by Graclus is quite low. Arbitrarily ordering the nodes at the coarsest level, then propagating this ordering to the finest levels, i.e., node k has nodes $2k$ and $2k+1$ as children, produces a regular ordering in the finest level. Regular in the sense that adjacent nodes are hierarchically merged at coarser levels. Pooling such a rearranged graph signal is analog to pooling a regular 1D signal. Figure 3.2 shows an example of the whole process. This regular arrangement makes the operation very efficient and satisfies parallel architectures such as GPUs as memory accesses are local, i.e., matched nodes do not have to be fetched.

3.3 Related works

3.3.1 Graph signal processing

The emerging field of GSP aims at bridging the gap between signal processing and spectral graph theory [15, 30, 114], a blend between graph theory and harmonic analysis. A goal is to generalize fundamental analysis operations for signals from regular grids to irregular structures embodied by graphs. We refer the reader to [153] for an introduction of the field. Standard operations on grids such as convolution, translation, filtering, dilatation, modulation or downsampling do not extend directly to graphs and thus require new mathematical definitions while keeping the original intuitive concepts. In this context, the authors of [36, 61, 74] revisited the construction of wavelet operators on graphs and techniques to perform mutli-scale pyramid transforms on graphs were proposed in [138, 154]. The works of [128, 131, 164] redefined uncertainty principles on graphs and showed that while intuitive concepts may be lost, enhanced localization principles can be derived.

3.3.2 CNNs on non-Euclidean domains

The Graph Neural Network framework [147], simplified in [109], was designed to embed each node in an Euclidean space with a RNN and use those embeddings as features for classification or regression of nodes or graphs. By setting their *transition function* f as a simple diffusion instead of a neural net with a recursive relation, their *state* vector becomes $s = f(x) = Wx$. Their point-wise *output function* g_θ can further be set as

$\hat{x} = g_\theta(s, x) = \theta(s - Dx) + x = \theta Lx + x$ instead of another neural net. The Chebyshev polynomials of degree K can then be obtained with a K-layer GNN, to be followed by a non-linear layer and a graph pooling operation. Our model can thus be interpreted as multiple layers of diffusions and node-local operations.

The works of [32, 70] introduced the concept of constructing a local receptive field to reduce the number of learned parameters. The idea is to group together features based upon a measure of similarity such as to select a limited number of connections between two successive layers. While this model reduces the number of parameters by exploiting the locality assumption, it did not attempt to exploit any stationarity property, i.e., no weight-sharing strategy. The authors of [25] used this idea for their spatial formulation of graph CNNs. They use a weighted graph to define the local neighborhood and compute a multiscale clustering of the graph for the pooling operation. Inducing weight sharing in a spatial construction is however challenging, as it requires to select and order the neighborhoods when a problem-specific ordering (spatial, temporal, or otherwise) is missing.

A spatial generalization of CNNs to 3D-meshes, a class of smooth low-dimensional non-Euclidean spaces, was proposed in [119]. The authors used geodesic polar coordinates to define the convolution on mesh patches, and formulated a deep learning architecture which allows comparison across different manifolds. They obtained state-of-the-art results for 3D shape recognition.

The first spectral formulation of a graph CNN, proposed in [25], defines a filter as

$$g_\theta(\Lambda) = B\theta, \tag{3.7}$$

where $B \in \mathbb{R}^{n \times K}$ is the cubic B-spline basis and the parameter $\theta \in \mathbb{R}^K$ is a vector of control points. They later proposed a strategy to learn the graph structure from the data and applied the model to image recognition, text categorization and bioinformatics [78]. This approach does however not scale up due to the necessary multiplications by the graph Fourier basis U. Despite the cost of computing this matrix, which requires an EVD on the graph Laplacian, the dominant cost is the need to multiply the data by this matrix twice (forward and inverse Fourier transforms) at a cost of $\mathcal{O}(n^2)$ operations per forward and backward pass, a computational bottleneck already identified by the authors. Besides, as they rely on smoothness in the Fourier domain, via the spline parametrization, to bring localization in the vertex domain, their model does not provide a precise control over the local support of their kernels, which is essential to learn

Model	Architecture	Accuracy
Classical CNN	C32-P4-C64-P4-FC512	99.33
Proposed graph CNN	GC32-P4-GC64-P4-FC512	99.14

Table 3.1: Classification accuracies of the proposed graph CNN and a classical CNN on MNIST.

localized filters. Our technique leverages on this work, and we showed how to overcome these limitations and beyond.

3.4 Numerical experiments

In the sequel, we refer to the non-parametric and non-localized filters (3.2) as *NonParam*, the filters (3.7) proposed in [25] as *Spline* and the proposed filters (3.4) as *Chebyshev*. We always use the Graclus coarsening algorithm introduced in §3.2.2 rather than the simple agglomerative method of [25]. Our motivation is to compare the learned filters, not the coarsening algorithms.

We use the following notation when describing network architectures: FCk denotes a fully connected layer with k hidden units, Pk denotes a (graph or classical) pooling layer of size and stride k, GCk and Ck denote a (graph) convolutional layer with k feature maps. All FCk, Ck and GCk layers are followed by a ReLU activation $\max(x, 0)$. The final layer is always a softmax regression and the loss energy E is the cross-entropy with an ℓ_2 regularization on the weights of all FCk layers. Mini-batches are of size $S = 100$.

3.4.1 Revisiting classical CNNs on MNIST

To validate our model, we applied it to the Euclidean case on the benchmark MNIST classification problem [106], a dataset of 70,000 digits represented on a 2D grid of size 28×28. For our graph model, we construct an 8-NN graph of the 2D grid which produces a graph of $n = |\mathcal{V}| = 976$ nodes ($28^2 = 784$ pixels and 192 fake nodes as explained in §3.2.3) and $|\mathcal{E}| = 3198$ edges. Following standard practice, the weights of a k-NN similarity graph (between features) are computed as

$$W_{ij} = \exp\left(-\frac{\|z_i - z_j\|_2^2}{\sigma^2}\right), \tag{3.8}$$

where z_i is the 2D coordinate of pixel i.

Model	Accuracy
Linear SVM	65.90
Multinomial Naive Bayes	68.51
Softmax	66.28
FC2500	64.64
FC2500-FC500	65.76
GC32	68.26

Table 3.2: Accuracies of the proposed graph CNN and other methods on 20NEWS.

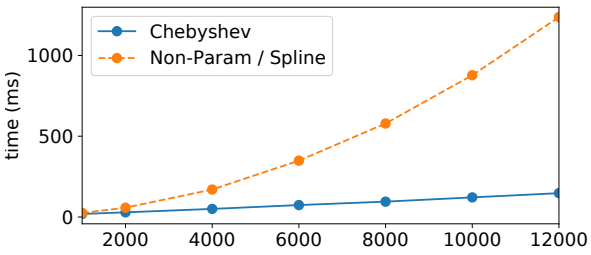


Figure 3.3: Time to process a mini-batch of $S = 100$ 20NEWS documents with respect to the number of words n.

This is an important sanity check for our model, which must be able to extract features on any graph, including the regular 2D grid. Table 3.1 shows the ability of our model to achieve a performance very close to a classical CNN with the same architecture. The gap in performance may be explained by the isotropic nature of the spectral filters, i.e., the fact that edges in a general graph do not possess an orientation (like up, down, right and left for pixels on a 2D grid). Whether this is a limitation or an advantage depends on the problem and should be verified, as for any invariance. Moreover, rotational invariance has been sought: (i) many data augmentation schemes have used rotated versions of images and (ii) models have been developed to learn this invariance, like the Spatial Transformer Networks [87]. Other explanations are the lack of experience on architecture design and the need to investigate better suited optimization or initialization strategies.

The LeNet-5-like network architecture and the following hyper-parameters are borrowed from the TensorFlow MNIST tutorial[2]: dropout probability of 0.5, regularization weight of 5×10^{-4}, initial learning rate of 0.03, learning rate decay of 0.95, momentum of 0.9. Filters are of size 5×5 and graph filters have the same support of $K = 25$. All models were trained for 20 epochs.

3.4.2 Text categorization on 20NEWS

To demonstrate the versatility of our model to work with graphs generated from unstructured data, we applied our technique to the text categorization problem on the 20NEWS dataset which consists of 18,846 (11,314 for training and 7,532 for testing) text documents associated with 20 classes [89]. We extracted the 10,000 most common words from the 93,953 unique words in this corpus. Each document x is represented using

[2] https://www.tensorflow.org/versions/r0.8/tutorials/mnist/pros

	Accuracy		
Architecture	Non-Param (3.2)	Spline (3.7) [25]	Chebyshev (3.4)
GC10	95.75	97.26	97.48
GC32-P4-GC64-P4-FC512	96.28	97.15	99.14

Table 3.3: Classification accuracies for different types of spectral filters ($K = 25$).

		Time (ms)		
Model	Architecture	CPU	GPU	Speedup
Classical CNN	C32-P4-C64-P4-FC512	210	31	6.77x
Proposed graph CNN	GC32-P4-GC64-P4-FC512	1600	200	8.00x

Table 3.4: Time to process a mini-batch of $S = 100$ MNIST images.

the bag-of-words model, normalized across words. To test our model, we constructed a 16-NN graph with (3.8) where z_i is the word2vec embedding [121] of word i, which produced a graph of $n = |\mathcal{V}| = 10,000$ nodes and $|\mathcal{E}| = 132,834$ edges. All models were trained for 20 epochs by the Adam optimizer [93] with an initial learning rate of 0.001. The architecture is GC32 with support $K = 5$. Table 3.2 shows decent performances: while the proposed model does not outperform the multinomial naive Bayes classifier on this small dataset, it does defeat fully connected networks, which require much more parameters.

3.4.3 Comparison between spectral filters and computational efficiency

Table 3.3 reports that the proposed parametrization (3.4) outperforms (3.7) from [25] as well as non-parametric filters (3.2) which are not localized and require $\mathcal{O}(n)$ parameters. Moreover, Figure 3.4 gives a sense of how the validation accuracy and the loss E converges with respect to the filter definitions.

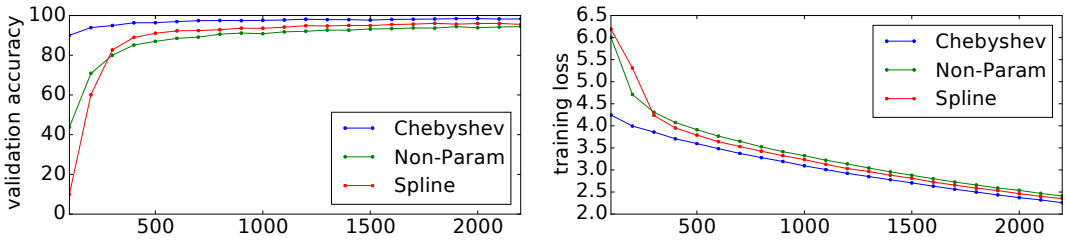


Figure 3.4: Plots of validation accuracy and training loss for the first 2000 iterations on MNIST.

Architecture	8-NN on 2D Euclidean grid	random
GC32	97.40	96.88
GC32-P4-GC64-P4-FC512	99.14	95.39

Table 3.5: Classification accuracies with different graph constructions on MNIST.

	word2vec			
bag-of-words	pre-learned	learned	approximate	random
67.50	66.98	68.26	67.86	67.75

Table 3.6: Classification accuracies of GC32 with different graph constructions on 20NEWS.

Figure 3.3 validates the low computational complexity of our model which scales as $\mathcal{O}(n)$ while [25] scales as $\mathcal{O}(n^2)$. The measured runtime is the total training time divided by the number of gradient steps. Table 3.4 shows a similar speedup as classical CNNs when moving to GPUs. This exemplifies the parallelization opportunity offered by our model, who relies solely on matrix multiplications. Those are efficiently implemented by cuBLAS, the linear algebra routines provided by NVIDIA.

3.4.4 Influence of graph quality

For any graph CNN to be successful, the statistical assumptions of locality, stationarity, and compositionality regarding the data must be fulfilled on the graph where the data resides. Therefore, the learned filters' quality and thus the classification performance critically depends on the quality of the graph. For data lying on Euclidean space, experiments in §3.4.1 show that a simple k-NN graph of the grid is good enough to recover almost exactly the performance of standard CNNs. We also noticed that the value of k does not have a strong influence on the results. We can witness the importance of a graph satisfying the data assumptions by comparing its performance with a random graph. Table 3.5 reports a large drop of accuracy when using a random graph, that is when the data structure is lost and the convolutional layers are not useful anymore to extract meaningful features.

While images can be structured by a grid graph, a feature graph has to be built for text documents represented as bag-of-words. We investigate here three ways to represent a word z: the simplest option is to represent each word as its corresponding column in the bag-of-words matrix while, another approach is to learn an embedding for each word with word2vec [121] or to use the pre-learned embeddings provided by the authors. For

larger datasets, an approximate nearest neighbors algorithm may be required, which is the reason we tried LSHForest [14] on the learned word2vec embeddings. Table 3.6 reports classification results which highlight the importance of a well constructed graph.

3.5 Conclusion

In this chapter, we have introduced the mathematical and computational foundations of an efficient generalization of CNNs to graphs using tools from GSP. Experiments have shown the ability of the model to extract local and stationary features through graph convolutional layers. Compared with the first work on spectral graph CNNs introduced in [25], our model provides a strict control over the local support of filters, is computationally more efficient by avoiding an explicit use of the Graph Fourier basis, and experimentally shows a better test accuracy. Besides, we addressed the three concerns raised by [78]: (i) we introduced a model whose computational complexity is linear with the dimensionality of the data, (ii) we confirmed that the quality of the input graph is of paramount importance, (iii) we showed that the statistical assumptions of local stationarity and compositionality made by the model are verified for text documents as long as the graph is well constructed.

Future works will investigate two directions. On one hand, we will enhance the proposed framework with newly developed tools in GSP. On the other hand, we will explore applications of this generic model to important fields where the data naturally lies on graphs, which may then incorporate external information about the structure of the data rather than artificially created graphs which quality may vary as seen in the experiments. Another natural and future approach, pioneered in [78], would be to alternate the learning of the CNN parameters and the graph.

4 Efficient learning on the sphere

Designing a convolution for a spherical neural network requires a delicate tradeoff between efficiency and rotation equivariance. DeepSphere, a method based on a graph representation of the sampled sphere, strikes a controllable balance between these two desiderata. This contribution is twofold. First, we study both theoretically and empirically how equivariance is affected by the underlying graph with respect to the number of vertices and neighbors. Second, we evaluate DeepSphere on relevant problems. Experiments show state-of-the-art performance and demonstrates the efficiency and flexibility of this formulation. Perhaps surprisingly, comparison with previous work suggests that anisotropic filters might be an unnecessary price to pay. Our code is available at https://github.com/deepsphere.

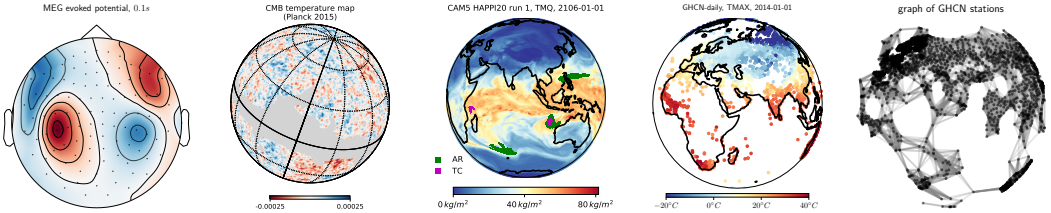
4.1 Introduction

Spherical data is found in many applications (Figure 4.1). Planetary data (such as meteorological or geological measurements) and brain activity are example of intrinsically spherical data. The observation of the universe, LIDAR scans, and the digitalization of 3D objects are examples of projections due to observation. Labels or variables are often to be inferred from them. Examples are the inference of cosmological parameters from the distribution of mass in the universe [132], the segmentation of omnidirectional images [92], and the segmentation of cyclones from Earth observation [125].

As neural networks (NNs) have proved to be great tools for inference, variants have been developed to handle spherical data. Exploiting the locally Euclidean property of the sphere, early attempts used standard 2D convolutions on a grid sampling of the sphere [20, 38, 159]. While simple and efficient, those convolutions are not equivariant to rotations. On the other side of this tradeoff, [34] and [56] proposed to perform proper spherical convolutions through the spherical harmonic transform. While equivariant to rotations, those convolutions are expensive (§4.2).

[2] https://martinos.org/mne/stable/auto_tutorials/plot_visualize_evoked.html

[2] https://www.ncdc.noaa.gov/ghcn-daily-description



(a) (b) (c) (d) (e)

Figure 4.1: Examples of spherical data: (a) brain activity recorded through magnetoencephalography (MEG),[1] (b) the cosmic microwave background (CMB) temperature from [135], (c) hourly precipitation from a climate simulation [88], (d) daily maximum temperature from the Global Historical Climatology Network (GHCN).[2] A rigid full-sphere sampling is not ideal: brain activity is only measured on the scalp, the Milky Way's galactic plane masks observations, climate scientists desire a variable resolution, and the position of weather stations is arbitrary and changes over time. (e) Graphs can faithfully and efficiently represent sampled spherical data by placing vertices where it matters.

As a lack of equivariance can penalize performance (§4.4.2) and expensive convolutions prohibit their application to some real-world problems, methods standing between these two extremes are desired. [35] proposed to reduce costs by limiting the size of the representation of the symmetry group by projecting the data from the sphere to the icosahedron. The distortions introduced by this projection might however hinder performance (§4.4.3).

Another approach is to represent the sampled sphere as a graph connecting pixels according to the distance between them [26, 92, 132]. While Laplacian-based graph convolutions are more efficient than spherical convolutions, they are not exactly equivariant [48]. In this work, we argue that graph-based spherical CNNs strike an interesting balance, with a controllable tradeoff between cost and equivariance (which is linked to performance). Experiments on multiple problems of practical interest show the competitiveness and flexibility of this approach.

4.2 Method

DeepSphere leverages graph convolutions to achieve the following properties: (i) computational efficiency, (ii) sampling flexibility, and (iii) rotation equivariance (§4.3). The main idea is to model the sampled sphere as a graph of connected pixels: the length of the shortest path between two pixels is an approximation of the geodesic distance between them. We use the graph CNN formulation introduced in [43] and a pooling strategy that exploits hierarchical samplings of the sphere.

SAMPLING. A sampling scheme $\mathcal{V} = \{x_i \in \mathbb{S}^2\}_{i=1}^n$ is defined to be the discrete subset of the sphere containing the n points where the values of the signals that we want to analyse are known. For a given continuous signal f, we represent such values in a vector $\boldsymbol{f} \in \mathbb{R}^n$. As there is no analogue of uniform sampling on the sphere, many samplings have been proposed with different tradeoffs. In this work, depending on the considered application, we will use the equiangular [51], HEALPix [67], and icosahedral [13] samplings.

GRAPH. From $\mathcal{V}$, we construct a weighted undirected graph $\mathcal{G} = (\mathcal{V}, w)$, where the elements of $\mathcal{V}$ are the vertices and the weight $w_{ij} = w_{ji}$ is a similarity measure between vertices x_i and x_j. The combinatorial graph Laplacian $\boldsymbol{L} \in \mathbb{R}^{n \times n}$ is defined as $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A}$, where $\boldsymbol{A} = (w_{ij})$ is the weighted adjacency matrix, $\boldsymbol{D} = (d_{ii})$ is the diagonal degree matrix, and $d_{ii} = \sum_j w_{ij}$ is the weighted degree of vertex x_i. Given a sampling $\mathcal{V}$, usually fixed by the application or the available measurements, the freedom in constructing $\mathcal{G}$ is in setting w. §4.3 shows how to set w to minimize the equivariance error.

CONVOLUTION. On Euclidean domains, convolutions are efficiently implemented by sliding a window in the signal domain. On the sphere however, there is no straightforward way to implement a convolution in the signal domain due to non-uniform samplings. Convolutions are most often performed in the spectral domain through a spherical harmonic transform (SHT). That is the approach taken by [34] and [56], which has a computational cost of $\mathcal{O}(n^{3/2})$ on isolatitude samplings (such as the HEALPix and equiangular samplings) and $\mathcal{O}(n^2)$ in general. On the other hand, following [43], graph convolutions can be defined as

$$h(\boldsymbol{L})\boldsymbol{f} = \left(\sum_{i=0}^{P} \alpha_i \boldsymbol{L}^i\right)\boldsymbol{f}, \tag{4.1}$$

where P is the polynomial order (which corresponds to the filter's size) and α_i are the coefficients to be optimized during training.[3] Those convolutions are used by [92] and [132] and cost $\mathcal{O}(n)$ operations through a recursive application of $\boldsymbol{L}$.[4]

[3] In practice, training with Chebyshev polynomials (instead of monomials) is slightly more stable. We believe it to be due to their orthogonality and uniformity.

[4] As long as the graph is sparsified such that the number of edges, i.e., the number of non-zeros in $\boldsymbol{A}$, is proportional to the number of vertices n. This can always be done as most weights are very small.

POOLING. Down- and up-sampling is natural for hierarchical samplings,[5] where each subdivision divides a pixel in (an equal number of) child sub-pixels. To pool (down-sample), the data supported on the sub-pixels is summarized by a permutation invariant function such as the maximum or the average. To unpool (up-sample), the data supported on a pixel is copied to all its sub-pixels.

ARCHITECTURE. All our NNs are fully convolutional, and employ a global average pooling (GAP) for rotation invariant tasks. Graph convolutional layers are always followed by batch normalization and ReLU activation, except in the last layer. Note that batch normalization and activation act on the elements of $\boldsymbol{f}$ independently, and hence don't depend on the domain of f.

4.3 GRAPH CONVOLUTION AND EQUIVARIANCE

While the graph framework offers great flexibility, its ability to faithfully represent the underlying sphere—for graph convolutions to be rotation equivariant—highly depends on the sampling locations and the graph construction.

4.3.1 PROBLEM FORMULATION

A continuous function $f : \mathcal{C}(\mathbb{S}^2) \supset F_{\mathcal{V}} \to \mathbb{R}$ is sampled as $T_{\mathcal{V}}(f) = \boldsymbol{f}$ by the sampling operator $T_{\mathcal{V}} : C(\mathbb{S}^2) \supset F_{\mathcal{V}} \to \mathbb{R}^n$ defined as $\boldsymbol{f} : f_i = f(x_i)$. We require $F_{\mathcal{V}}$ to be a suitable subspace of continuous functions such that $T_{\mathcal{V}}$ is invertible, i.e., the function $f \in F_{\mathcal{V}}$ can be unambiguously reconstructed from its sampled values $\boldsymbol{f}$. The existence of such a subspace depends on the sampling $\mathcal{V}$ and its characterization is a common problem in signal processing [51]. For most samplings, it is not known if $F_{\mathcal{V}}$ exists and hence if $T_{\mathcal{V}}$ is invertible. A special case is the equiangular sampling where a sampling theorem holds, and thus a closed-form of ${T_{\mathcal{V}}}^{-1}$ is known. For samplings where no such sampling formula is available, we leverage the discrete SHT to reconstruct f from $\boldsymbol{f} = T_{\mathcal{V}} f$, thus approximating ${T_{\mathcal{V}}}^{-1}$. For all theoretical considerations, we assume that $F_{\mathcal{V}}$ exists and $f \in F_{\mathcal{V}}$.

By definition, the (spherical) graph convolution is rotation equivariant if and only if it commutes with the rotation operator defined as $R(g), g \in SO(3)$: $R(g)f(x) = f\big(g^{-1}x\big)$. In the context of this work, graph convolution is performed by recursive applications of the graph Laplacian (4.1). Hence, if $R(g)$ commutes with $\boldsymbol{L}$, then, by recursion, it will

[5] The equiangular, HEALPix, and icosahedral samplings are of this kind.

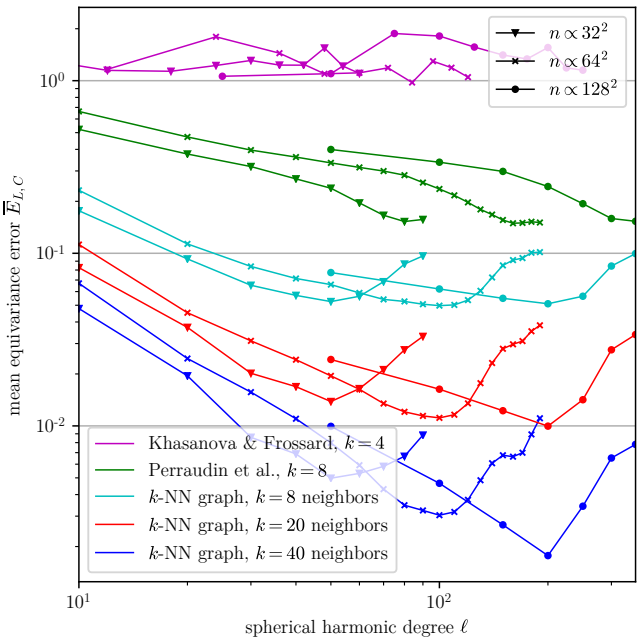


Figure 4.2: Mean equivariance error (4.3). There is a clear tradeoff between equivariance and computational cost, governed by the number of vertices n and edges kn.

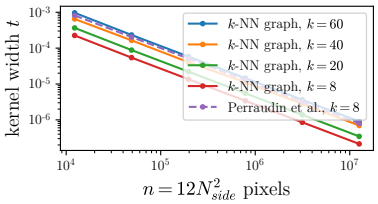


Figure 4.3: Kernel widths.

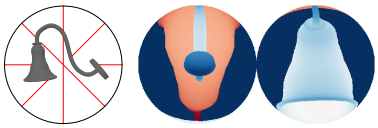


Figure 4.4: 3D object represented as a spherical depth map.

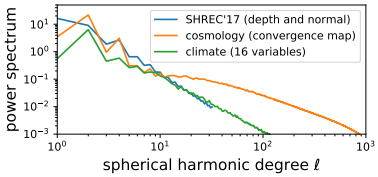


Figure 4.5: Power spectral densities.

also commute with the convolution $h(\boldsymbol{L})$. As a result, $h(\boldsymbol{L})$ is rotation equivariant if and only if

$$\boldsymbol{R}_{\mathcal{V}}(g)\boldsymbol{L}\boldsymbol{f} = \boldsymbol{L}\boldsymbol{R}_{\mathcal{V}}(g)\boldsymbol{f}, \qquad \forall f \in F_{\mathcal{V}} \text{ and } \forall g \in SO(3),$$

where $\boldsymbol{R}_{\mathcal{V}}(g) = T_{\mathcal{V}}R(g){T_{\mathcal{V}}}^{-1}$. For an empirical evaluation of equivariance, we define the *normalized equivariance error* for a signal $\boldsymbol{f}$ and a rotation g as

$$E_{\boldsymbol{L}}(\boldsymbol{f}, g) = \left(\frac{\|\boldsymbol{R}_{\mathcal{V}}(g)\boldsymbol{L}\boldsymbol{f} - \boldsymbol{L}\boldsymbol{R}_{\mathcal{V}}(g)\boldsymbol{f}\|}{\|\boldsymbol{L}\boldsymbol{f}\|}\right)^2. \tag{4.2}$$

More generally for a class of signals $f \in C \subset F_{\mathcal{V}}$, the *mean equivariance error* defined as

$$\overline{E}_{\boldsymbol{L},C} = \mathbb{E}_{\boldsymbol{f}\in C, g\in SO(3)}\, E_{\boldsymbol{L}}(\boldsymbol{f}, g) \tag{4.3}$$

represents the overall equivariance error. The expected value is obtained by averaging over a finite number of random functions and random rotations.

4.3.2 Finding the optimal weighting scheme

Considering the equiangular sampling and graphs where each vertex is connected to 4 neighbors (north, south, east, west), [92] designed a weighting scheme to minimize (4.3) for longitudinal and latitudinal rotations[6]. Their solution gives weights inversely proportional to Euclidean distances:

$$w_{ij} = \frac{1}{\left\|x_i - x_j\right\|}. \tag{4.4}$$

While the resulting convolution is not equivariant to the whole of $SO(3)$ (Figure 4.2), it is enough for omnidirectional imaging because, as gravity consistently orients the sphere, objects only rotate longitudinally or latitudinally.

To achieve equivariance to all rotations, we take inspiration from [18]. They prove that for a *random uniform sampling*, the graph Laplacian $\boldsymbol{L}$ built from weights

$$w_{ij} = e^{-\frac{1}{4t}\left\|x_i - x_j\right\|^2} \tag{4.5}$$

converges to the Laplace-Beltrami operator $\Delta_{\mathbb{S}^2}$ as the number of samples grows to infinity. This result is a good starting point as $\Delta_{\mathbb{S}^2}$ commutes with rotation, i.e., $\Delta_{\mathbb{S}^2} R(g) = R(g)\Delta_{\mathbb{S}^2}$. While the weighting scheme is full (i.e., every vertex is connected to every other vertex), most weights are small due to the exponential. We hence make an approximation to limit the cost of the convolution (4.1) by only considering the k nearest neighbors (k-NN) of each vertex. Given k, the optimal kernel width t is found by searching for the minimizer of (4.3). Figure 4.3 shows the optimal kernel widths found for various resolutions of the HEALPix sampling. As predicted by the theory, $t_n \propto n^\beta, \beta \in \mathbb{R}$. Importantly however, the optimal t also depends on the number of neighbors k.

Considering the HEALPix sampling, [132] connected each vertex to their 8 adjacent vertices in the tiling of the sphere, computed the weights with (4.5), and heuristically set t to half the average squared Euclidean distance between connected vertices. This heuristic however over-estimates t (Figure 4.3) and leads to an increased equivariance error (Figure 4.2).

4.3.3 Analysis of the proposed weighting scheme

We analyze the proposed weighting scheme both theoretically and empirically.

[6] Equivariance to longitudinal rotation is essentially given by the equiangular sampling.

THEORETICAL CONVERGENCE. We extend the work of [18] to a sufficiently regular, deterministic sampling. Following their setting, we work with the *extended graph Laplacian* operator as the linear operator $L_n^t : L^2(\mathbb{S}^2) \to L^2(\mathbb{S}^2)$ such that

$$L_n^t f(y) := \frac{1}{n} \sum_{i=1}^{n} e^{-\frac{\|x_i - y\|^2}{4t}} (f(y) - f(x_i)). \tag{4.6}$$

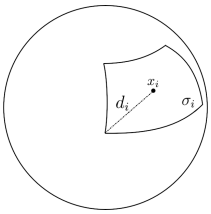


Figure 4.6: Patch.

This operator extends the graph Laplacian with the weighting scheme (4.5) to each point of the sphere (i.e., $\boldsymbol{L}_n^t \boldsymbol{f} = T_{\mathcal{V}} L_n^t f$). As the radius of the kernel t will be adapted to the number of samples, we scale the operator as $\hat{L}_n^t := |\mathbb{S}^2|(4\pi t^2)^{-1} L_n^t$. Given a sampling $\mathcal{V}$, we define σ_i to be the patch of the surface of the sphere corresponding to x_i, A_i its corresponding area, and d_i the largest distance between the center x_i and any point on the surface σ_i. Define $d^{(n)} := \max_{i=1,\dots,n} d_i$ and $A^{(n)} := \max_{i=1,\dots,n} A_i$.

Theorem 4.3.1 *For a sampling $\mathcal{V}$ of the sphere that is equi-area and such that $d^{(n)} \leq \frac{C}{n^\alpha}$, $\alpha \in (0, \frac{1}{2}]$, for all $f : \mathbb{S}^2 \to \mathbb{R}$ Lipschitz with respect to the Euclidean distance in $\mathbb{R}^3$, for all $y \in \mathbb{S}^2$, there exists a sequence $t_n = n^\beta$, $\beta \in \mathbb{R}$ such that*

$$\lim_{n \to \infty} \hat{L}_n^{t_n} f(y) = \Delta_{\mathbb{S}^2} f(y).$$

This is a major step towards equivariance, as the Laplace-Beltrami operator commutes with rotation. Based on this property, we show the equivariance of the scaled extended graph Laplacian.

Theorem 4.3.2 *Under the hypothesis of Theorem 4.3.1, the scaled graph Laplacian commutes with any rotation, in the limit of infinite sampling, i.e.,*

$$\forall y \in \mathbb{S}^2 \quad \left| R(g) \hat{L}_n^{t_n} f(y) - \hat{L}_n^{t_n} R(g) f(y) \right| \xrightarrow{n \to \infty} 0.$$

From this theorem, it follows that the discrete graph Laplacian will be equivariant in the limit of $n \to \infty$ as by construction $\boldsymbol{L}_n^t \boldsymbol{f} = T_{\mathcal{V}} L_n^t f$ and as the scaling does not affect the equivariance property of $\boldsymbol{L}_n^t$.

Importantly, the proof of Theorem 4.3.1 ([72, Appendix A]) inspires our construction of the graph Laplacian. In particular, it tells us that t should scale as n^β, which has been empirically verified (Figure 4.3). Nevertheless, it is important to keep in mind the limits of Theorem 4.3.1 and 4.3.2. Both theorems present asymptotic results, but

in practice we will always work with finite samplings. Furthermore, since this method is based on the capability of the eigenvectors of the graph Laplacian to approximate the spherical harmonics, a stronger type of convergence of the graph Laplacian would be preferable, i.e., spectral convergence (that is proved for a full graph in the case of random sampling for a class of Lipschitz functions in [16]). Finally, while we do not have a formal proof for it, we strongly believe that the HEALPix sampling does satisfy the hypothesis $d^{(n)} \leq \frac{C}{n^\alpha}$, $\alpha \in (0, \frac{1}{2}]$, with α very close or equal to $\frac{1}{2}$. The empirical results discussed in the next paragraph also points in this direction. This is further discussed in [72, Appendix A].

EMPIRICAL CONVERGENCE. Figure 4.2 shows the equivariance error (4.3) for different parameter sets of DeepSphere for the HEALPix sampling as well as for the graph construction of [92] for the equiangular sampling. The error is estimated as a function of the sampling resolution and signal frequency. The resolution is controlled by the number of pixels $n = 12{N_{\text{side}}}^2$ for HEALPix and $n = 4b^2$ for the equiangular sampling. The frequency is controlled by setting the set C to functions f made of spherical harmonics of a single degree ℓ. To allow for an almost perfect implementation (up to numerical errors) of the operator $\boldsymbol{R}_\mathcal{V}$, the degree ℓ was chosen in the range $(0, 3N_{\text{side}} - 1)$ for HEALPix and $(0, b)$ for the equiangular sampling [68]. Using these parameters, the measured error is mostly due to imperfections in the empirical approximation of the Laplace-Beltrami operator and not to the sampling.

Figure 4.2 shows that the weighting scheme (4.4) from [92] does indeed not lead to a convolution that is equivariant to all rotations $g \in SO(3)$.[7] For $k = 8$ neighbors, selecting the optimal kernel width t improves on [132] at no cost, highlighting the importance of this parameter. Increasing the resolution decreases the equivariance error in the high frequencies, an effect most probably due to the sampling. Most importantly, the equivariance error decreases when connecting more neighbors. Hence, the number of neighbors k gives us a precise control of the tradeoff between cost and equivariance.

4.4 EXPERIMENTS

4.4.1 3D OBJECTS RECOGNITION

The recognition of 3D shapes is a rotation invariant task: rotating an object doesn't change its nature. While 3D shapes are usually represented as meshes or point clouds,

[7] We however verified that the convolution is equivariant to longitudinal and latitudinal rotations, as intended.

	performance		size	speed	
	F1	mAP	params	inference	training
[34] ($b = 128$)	-	67.6	1400 k	38.0 ms	50 h
[34] (simplified,[9] $b = 64$)	78.9	66.5	400 k	12.0 ms	32 h
[56] ($b = 64$)	79.4	68.5	500 k	9.8 ms	3 h
DeepSphere (equiangular, $b = 64$)	79.4	66.5	190 k	0.9 ms	50 m
DeepSphere (HEALPix, $N_{\text{side}} = 32$)	80.7	68.6	190 k	0.9 ms	50 m

Table 4.1: Results on SHREC'17 (3D shapes). DeepSphere achieves similar performance at a much lower cost, suggesting that anisotropic filters are an unnecessary price to pay.

representing them as spherical maps (Figure 4.4) naturally allows a rotation invariant treatment.

The SHREC'17 shape retrieval contest [146] contains 51,300 randomly oriented 3D models from ShapeNet [28], to be classified in 55 categories (tables, lamps, airplanes, etc.). As in [34], objects are represented by 6 spherical maps. At each pixel, a ray is traced towards the center of the sphere. The distance from the sphere to the object forms a depth map. The cos and sin of the surface angle forms two normal maps. The same is done for the object's convex hull.[8] The maps are sampled by an equiangular sampling with bandwidth $b = 64$ ($n = 4b^2 = 16,384$ pixels) or an HEALPix sampling with $N_{\text{side}} = 32$ ($n = 12{N_{\text{side}}}^2 = 12,288$ pixels).

The equiangular graph is built with (4.4) and $k = 4$ neighbors (following [92]). The HEALPix graph is built with (4.5), $k = 8$, and a kernel width t set to the average of the distances (following [132]). The NN is made of 5 graph convolutional layers, each followed by a max pooling layer which down-samples by 4. A GAP and a fully connected layer with softmax follow. The polynomials are all of order $P = 3$ and the number of channels per layer is 16, 32, 64, 128, 256, respectively. Following [56], the cross-entropy plus a triplet loss is optimized with Adam for 30 epochs on the dataset augmented by 3 random translations. The learning rate is $5 \cdot 10^{-2}$ and the batch size is 32.

Results are shown in Table 4.1. As the network is trained for shape classification rather than retrieval, we report the classification F1 alongside the mAP used in the retrieval contest.[10] DeepSphere achieves the same performance as [34] and [56] at a much lower cost, suggesting that anisotropic filters are an unnecessary price to pay. As the information in those spherical maps resides in the low frequencies (Figure 4.5), reduc-

[8] Albeit we didn't observe much improvement by using the convex hull.
[9] As implemented in https://github.com/jonas-koehler/s2cnn.
[10] We omit the F1 for [34] as we didn't get the mAP reported in the paper when running it.

	accuracy	time
[132], 2D CNN baseline	54.2	104 ms
[132], CNN variant, $k = 8$	62.1	185 ms
[132], FCN variant, $k = 8$	83.8	185 ms
$k = 8$ neighbors, t from §4.3.2	87.1	185 ms
$k = 20$ neighbors, t from §4.3.2	91.3	250 ms
$k = 40$ neighbors, t from §4.3.2	92.5	363 ms

Table 4.2: Results on the classification of partial convergence maps. Lower equivariance error translates to higher performance.

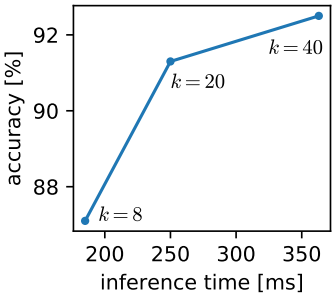


Figure 4.7: Tradeoff between cost and accuracy.

ing the equivariance error didn't translate into improved performance. For the same reason, using the more uniform HEALPix sampling or lowering the resolution down to $N_{\text{side}} = 8$ ($n = 768$ pixels) didn't impact performance either.

4.4.2 Cosmological model classification

Given observations, cosmologists estimate the posterior probability of cosmological parameters, such as the matter density Ω_m and the normalization of the matter power spectrum σ_8. Those parameters are estimated by likelihood-free inference, which requires a method to extract summary statistics to compare simulations and observations. As the sufficient and most concise summary statistics are the parameters themselves, one desires a method to predict them from simulations. As that is complicated to setup, prediction methods are typically benchmarked on the classification of spherical maps instead [149]. We used the same task, data, and setup as [132]: the classification of 720 partial convergence maps made of $n \approx 10^6$ pixels ($1/12 \approx 8\%$ of a sphere at $N_{\text{side}} = 1024$) from two ΛCDM cosmological models, ($\Omega_m = 0.31$, $\sigma_8 = 0.82$) and ($\Omega_m = 0.26$, $\sigma_8 = 0.91$), at a relative noise level of 3.5 (i.e., the signal is hidden in noise of 3.5 times higher standard deviation). Convergence maps represent the distribution of over- and under-densities of mass in the universe (see [10] for a review of gravitational lensing).

Graphs are built with (4.5), $k = 8, 20, 40$ neighbors, and the corresponding optimal kernel widths t given in §4.3.2. Following [132], the NN is made of 5 graph convolutional layers, each followed by a max pooling layer which down-samples by 4. A GAP and a fully connected layer with softmax follow. The polynomials are all of order $P = 4$ and the number of channels per layer is $16, 32, 64, 64, 64$, respectively. The cross-entropy loss is optimized with Adam for 80 epochs. The learning rate is $2 \cdot 10^{-4} \cdot 0.999^{\text{step}}$ and the batch size is 8.

	accuracy	mAP
[88] (rerun)	94.95	38.41
[35] (S2R)	97.5	68.6
[35] (R2R)	97.7	75.9
DeepSphere (weighted loss)	97.8 ± 0.3	77.15 ± 1.94
DeepSphere (non-weighted loss)	87.8 ± 0.5	89.16 ± 1.37

Table 4.3: Results on climate event segmentation: mean accuracy (over TC, AR, BG) and mean average precision (over TC and AR). DeepSphere achieves state-of-the-art performance.

Unlike on SHREC'17, results (Table 4.2) show that a lower equivariance error on the convolutions translates to higher performance. That is probably due to the high frequency content of those maps (Figure 4.5). There is a clear cost-accuracy tradeoff, controlled by the number of neighbors k (Figure 4.7). This experiment moreover demonstrates DeepSphere's flexibility (using partial spherical maps) and scalability (competing spherical CNNs were tested on maps of at most $10,000$ pixels).

4.4.3 Climate event segmentation

We evaluate our method on a task proposed by [125]: the segmentation of extreme climate events, Tropical Cyclones (TC) and Atmospheric Rivers (AR), in global climate simulations (Figure 4.1c). The data was produced by a 20-year run of the Community Atmospheric Model v5 (CAM5) and consists of 16 channels such as temperature, wind, humidity, and pressure at multiple altitudes. We used the pre-processed dataset from [88].[11] There is 1,072,805 spherical maps, down-sampled to a level-5 icosahedral sampling ($n = 10 \cdot 4^l + 2 = 10,242$ pixels). The labels are heavily unbalanced with 0.1% TC, 2.2% AR, and 97.7% background (BG) pixels.

The graph is built with (4.5), $k = 6$ neighbors, and a kernel width t set to the average of the distances. Following [88], the NN is an encoder-decoder with skip connections. Details in [72, Appendix C.3]. The polynomials are all of order $P = 3$. The cross-entropy loss (weighted or non-weighted) is optimized with Adam for 30 epochs. The learning rate is $1 \cdot 10^{-3}$ and the batch size is 64.

Results are shown in Table 4.3 (details in [72, tables 6, 7, and 8]). The mean and standard deviation are computed over 5 runs. Note that while [88] and [35] use a weighted cross-entropy loss, that is a suboptimal proxy for the mAP metric. DeepSphere achieves state-of-the-art performance, suggesting again that anisotropic filters are unnecessary.

[11] Available at `http://island.me.berkeley.edu/ugscnn/data`.

	temp. (from past temp.)			day (from temperature)			day (from precipitations)		
order P	MSE	MAE	R2	MSE	MAE	R2	MSE	MAE	R2
0	10.88	2.42	0.896	0.10	0.10	0.882	0.58	0.42	−0.980
4	8.20	2.11	0.919	0.05	0.05	0.969	0.50	0.18	0.597

Table 4.4: Prediction results on data from weather stations. Structure always improves performance.

Note that results from [125] cannot be directly compared as they don't use the same input channels.

Compared to [35]'s conclusion, it is surprising that S2R does worse than DeepSphere (which is limited to S2S). Potential explanations are (i) that their icosahedral projection introduces harmful distortions, or (ii) that a larger architecture can compensate for the lack of generality. We indeed observed that more feature maps and depth led to higher performance [72, Appendix C.3].

4.4.4 Uneven sampling

To demonstrate the flexibility of modeling the sampled sphere by a graph, we collected historical measurements from $n \approx 10,000$ weather stations scattered across the Earth.[12] The spherical data is heavily non-uniformly sampled, with a much higher density of weather stations over North America than the Pacific (Figure 4.1d). For illustration, we devised two artificial tasks. A dense regression: predict the temperature on a given day knowing the temperature on the previous 5 days. A global regression: predict the day (represented as one period of a sine over the year) from temperature or precipitations. Predicting from temperature is much easier as it has a clear yearly pattern.

The graph is built with (4.5), $k = 5$ neighbors, and a kernel width t set to the average of the distances. The equivariance property of the resulting graph has not been tested, and we don't expect it to be good due to the heavily non-uniform sampling. The NN is made of 3 graph convolutional layers. The polynomials are all of order $P = 0$ or 4 and the number of channels per layer is $50, 100, 100$, respectively. For the global regression, a GAP and a fully connected layer follow. For the dense regression, a graph convolutional layer follows instead. The MSE loss is optimized with RMSprop for 250 epochs. The learning rate is $1 \cdot 10^{-3}$ and the batch size is 64.

Results are shown in Table 4.4. While using a polynomial order $P = 0$ is like modeling each time series independently with an MLP, orders $P > 0$ integrate neighborhood

[12] https://www.ncdc.noaa.gov/ghcn-daily-description

information. Results show that using the structure induced by the spherical geometry always yields better performance.

4.5 Conclusion

This work showed that DeepSphere strikes an interesting, and we think currently optimal, balance between desiderata for a spherical CNN. A single parameter, the number of neighbors k a pixel is connected to in the graph, controls the tradeoff between cost and equivariance (which is linked to performance). As computational cost and memory consumption scales linearly with the number of pixels, DeepSphere scales to spherical maps made of millions of pixels, a required resolution to faithfully represent cosmological and climate data. Also relevant in scientific applications is the flexibility offered by a graph representation (for partial coverage, missing data, and non-uniform samplings). Finally, the implementation of the graph convolution is straightforward, and the ubiquity of graph neural networks—pushing for their first-class support in DL frameworks—will make implementations even easier and more efficient.

A potential drawback of graph Laplacian-based approaches is the isotropy of graph filters, reducing in principle the expressive power of the NN. Experiments from [35] and [22] indeed suggest that more general convolutions achieve better performance. Our experiments on 3D shapes (§4.4.1) and climate (§4.4.3) however show that DeepSphere's isotropic filters do not hinder performance. Possible explanations for this discrepancy are that NNs somehow compensate for the lack of anisotropic filters, or that some tasks can be solved with isotropic filters. The distortions induced by the icosahedral projection in [35] or the leakage of curvature information in [22] might also alter performance.

Developing graph convolutions on irregular samplings that respect the geometry of the sphere is another research direction of importance. Practitioners currently interpolate their measurements (coming from arbitrarily positioned weather stations, satellites or telescopes) to regular samplings. This practice either results in a waste of resolution or computational and storage resources. Our ultimate goal is for practitioners to be able to work directly on their measurements, however distributed.

5 Cosmological parameter inference

Convolutional Neural Networks (CNNs) are a cornerstone of the Deep Learning toolbox and have led to many breakthroughs in Artificial Intelligence. So far, these neural networks (NNs) have mostly been developed for regular Euclidean domains such as those supporting images, audio, or video. Because of their success, CNN-based methods are becoming increasingly popular in Cosmology. Cosmological data often comes as spherical maps, which make the use of the traditional CNNs more complicated. The commonly used pixelization scheme for spherical maps is the Hierarchical Equal Area isoLatitude Pixelisation (HEALPix). We present a spherical CNN for analysis of full and partial HEALPix maps, which we call DeepSphere. The spherical CNN is constructed by representing the sphere as a graph. Graphs are versatile data structures that can represent pairwise relationships between objects or act as a discrete representation of a continuous manifold. Using the graph-based representation, we define many of the standard CNN operations, such as convolution and pooling. With filters restricted to being radial, our convolutions are equivariant to rotation on the sphere, and DeepSphere can be made invariant or equivariant to rotation. This way, DeepSphere is a special case of a graph CNN, tailored to the HEALPix sampling of the sphere. This approach is computationally more efficient than using spherical harmonics to perform convolutions. We demonstrate the method on a classification problem of weak lensing mass maps from two cosmological models and compare its performance with that of three baseline classifiers, two based on the power spectrum and pixel density histogram, and a classical 2D CNN. Our experimental results show that the performance of DeepSphere is always superior or equal to the baselines. For high noise levels and for data covering only a smaller fraction of the sphere, DeepSphere achieves typically 10% better classification accuracy than the baselines. Finally, we show how learned filters can be visualized to introspect the NN. Code and examples are available at https://github.com/deepsphere/deepsphere-cosmo-tf1.

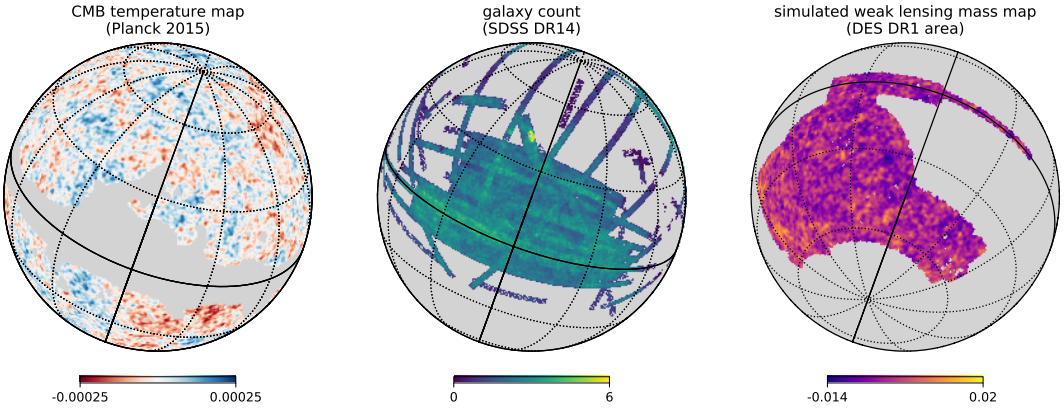


Figure 5.1: Example maps on the sphere: (left) the CMB temperature (K) map from Planck [135], with galactic plane masked, (middle) map of galaxy number counts (number of galaxies per arcmin^2) in SDSS DR14 [2], and (right) simulated weak lensing convergence map (dimensionless) simulated with DES DR1 mask [40]. These maps were pixelised using $N_{\text{side}} = 512$. The CMB and weak lensing mass maps were smoothed with Gaussian kernels with FWHM=1 deg, and the galaxy count map with FWHM=0.5 deg.

5.1 Introduction

Cosmological and astrophysical data often come in the form of spherical sky maps. Observables that cover large parts of the sky, such as the Cosmic Microwave Background (CMB) [96, 136, 158], neutral hydrogen [79, 145], galaxy clustering [4], gravitational lensing [80, 163], and others, have been used to constrain cosmological and astrophysical models. Cosmological information contained in these maps is usually extracted using summary statistics, such as the power spectra or higher order correlation functions. Convolutional Neural Networks (CNNs) have been proposed as an alternative analysis tool in cosmology thanks to their ability to automatically design relevant statistics to maximise the precision[1] of the parameter estimation [6, 31, 63, 71, 76, 113, 139, 149], while maintaining robustness to noise. This is possible as neural networks (NNs) have the capacity to build rich models and capture complicated non-linear patterns often present in the data. CNNs are particularly well suited for the analysis of cosmological data as their trainable weights are shared across the domain, i.e., the network does not have to relearn to detect objects or features at every spatial location.

So far these algorithms have mostly been demonstrated on Euclidean domains, such as images. The main challenge in designing a CNN on the sphere is to define a con-

[1]Here, the word "precision" is to be understood as the final size of the posterior distribution on the measured parameters (including systematic errors).

volution operation that is suitable for this domain, while taking care of the necessary irregular sampling. Moreover, the designed convolution and resulting NN should possess the following three key characteristics. First, the convolution should be equivariant to rotation, meaning that a rotation of the input implies the same rotation of the output. Sky maps are rotation equivariant: rotating a map on the sphere doesn't change its interpretation. Depending on the task, we want the CNN to be either equivariant or invariant to rotation.[2] Second, to be able to train the network in reasonable time, the convolution has to be computationally efficient. Third, a CNN should work well on parts of the sphere, as many cosmological observations cover only a part of the sky. For ground-based observations this can be due to limited visibility of the sky from a particular telescope location, and for space-based instruments due to masking of the galactic plane area (see Figure 5.1 for example maps).

Three ways of generalizing CNNs to spherical data have been pursued. One approach is to apply a standard 2D CNN to a grid discretisation of the sphere [21, 39, 160]. An alternative is to divide the sphere into small chunks and project those on flat 2D surfaces [57, 63, 71, 149]. While these approaches use the well-developed 2D convolution and hierarchical pooling, they are not equivariant to rotation. Another way is to leverage the spherical Fourier transform and to perform the convolution associated to the $SO(3)$ rotation group in the spectral domain, thanks to the convolution theorem [34, 55]. While the resulting convolution is equivariant to rotation, this approach is computationally expensive, even if a fast spherical Fourier transform is used. Moreover, all those methods cannot be much accelerated when maps only span a part of the sky.

Our spherical CNN leverages convolutions on graphs and hierarchical pooling to achieve the following properties: (i) rotation equivariance, (ii) computational efficiency, and (iii) partial sky observations. The main idea is to model the discretised sphere as a graph of connected pixels: the length of the shortest path between two pixels is an approximation of the geodesic distance between them. We use the graph CNN formulation introduced in [43], and a pooling strategy that exploits a hierarchical pixelisation of the sphere to analyse the data at multiple scales. As the Equal Area isoLatitude Pixelisation (HEALPix) [67] is a popular sampling used in cosmology and astrophysics, we tailored the method to that particular sampling. DeepSphere is, however, easily used with other samplings as only two elements depend on it: (i) the choice of neighboring pixels when building the graph, and (ii) the choice of parent pixels when building the hierarchy. The flexibility of modeling the data domain with a graph allows one to easily model data that spans only a part of the sphere, or data that is not uniformly sampled.

[2] When only the statistics of the maps are relevant, they are rotation invariant.

Using a k-nearest neighbours graph, the convolution operation costs $\mathcal{O}(N_{\text{pix}})$ operations, where N_{pix} is the number of pixels. This is the lowest possible complexity for a convolution without approximations. DeepSphere is readily apt to solve four tasks: (i) global classification (i.e., predict a class from a map), (ii) global regression (i.e., predict a set of parameters from a map), (iii) dense classification (i.e., predict a class for each pixel of a map), and (iv) dense regression, (i.e., predict a set of maps from a map). Input data are spherical maps with a single value per pixel, such as the CMB temperature, or multiple values per pixel, such as surveys at multiple radio frequencies.

We give a practical demonstration of DeepSphere on cosmological model discrimination using maps of projected mass distribution on the sky [29]. These kind of maps can be created using the gravitational lensing technique (see [11] for a review). Our maps are similar to the ones used by [149]. In a simplified scenario, we classify partial sky convergence maps into two cosmological models. These models were designed to have very similar angular power spectrum. We compare the performance of DeepSphere to three baselines: a 2D CNN, and an SVM classifier that takes pixel histograms or power spectral densities (PSDs) of these maps as input. The comparison is made as a function of the additive noise level and the area of the sphere used in the analysis. Results show that our model is always better at discriminating the maps, especially in the presence of noise. DeepSphere is implemented with TensorFlow [118] and is intended to be easy to use out-of-the-box for cosmological applications. The Python Graph Signal Processing package (PyGSP) [44] is used to build graphs, compute the Laplacian and Fourier basis, and perform graph convolutions. Code and examples are available online.[3]

5.2 Method

A CNN is composed of the following main building blocks [107]: (i) a convolution, (ii) a non-linearity, and, optionally, (iii) a down-sampling operation, (iv) a pooling operation, and (v) a normalization.[4] Our architecture is depicted in Figure 5.2 and discussed in greater details in §5.2.8. As operations (ii) and (v) are point-wise, they do not depend on the data domain. The pooling operation is simply a permutation invariant aggregation function which does not need to be adapted either. The convolution and down-sampling operations, however, need to be generalized from Euclidean domains to the sphere.

[3] https://github.com/deepsphere/deepsphere-cosmo-tf1

[4] Batch normalization has been shown to help training [86]. We verified this experimentally in our setting as well.

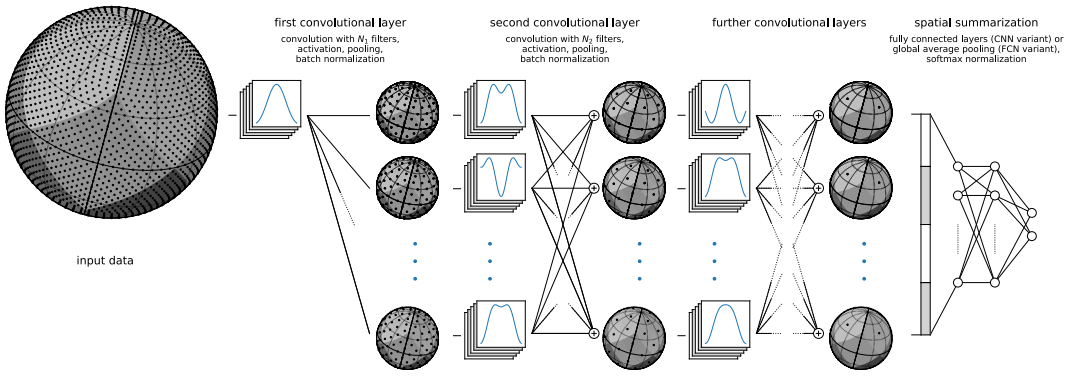


Figure 5.2: Overall NN architecture, showing here three convolutional layers acting as feature extractors followed by three fully connected layers with softmax acting as the classifier. A convolutional layer is based on five operations: convolution, non-linearity, batch normalization, down-sampling, and pooling. While most operations are agnostic to the data domain, the convolution and the down-sampling have to be adapted. In this chapter, we propose first to model the sphere with a graph and to perform the convolution on the graph. Graphs are versatile data structures which can model any sampling, even irregular or partial. Second, we propose to exploit a hierarchical pixelization of the sphere for the down-sampling operation. It allows the NN to analyze the data at multiple scales while preserving the spatial localization of features. This figure shows a network that operates on the whole sphere. The process is the same when working with partial observations, except that the graph is only built for the region of interest.

On regular Euclidean domains, such as 1-dimensional time series or 2-dimensional images, a convolution can be efficiently implemented by sliding a localized convolution kernel (for example a patch of 5×5 pixels) in the signal domain. Because of the irregular sampling, there is no straightforward way to define a convolution on the sphere directly in the pixel domain; convolutions are most often performed using spherical harmonics. In our method we also use the spectral domain to define the convolution. The implementation, however, does not need a direct access to the spectrum, which is computationally more efficient (see §5.2.5).

Down-sampling is achieved on regular Euclidean domains by keeping one pixel every n pixels in every dimension. That is again not a suitable strategy on the sphere because of the irregular sampling.

The gist of our method is to define the convolution operation on a sphere using a graph, and the down-sampling operation using a hierarchical pixelisation of the sphere.

5.2.1 HEALPix sampling

Before doing any numerical analysis on the sphere, one first has to choose a tessellation, i.e., an exhaustive partition of the sphere into finite area elements, where the data under study is quantized. The simplicity of the spherical form belies the intricacy of global analysis on the sphere: there is no known point set that achieves the analogue of uniform sampling in Euclidean space. While our method is applicable to any pixelisation of the sphere, two details depend on the chosen sampling: (i) the choice of neighbours in the construction of the graph, and (ii) the choice of parent vertices when coarsening the graph. As HEALPix [67] is our target application, we tailor the method to that particular sampling in the subsequent exposition. Figure 5.1 shows three examples of HEALPix maps: the Cosmic Microwave Background [135], galaxies found in Sloan Digital Sky Survey Data Release 14 [2], and an example simulated mass map on the footprint of Dark Energy Survey Data Release 1 [40].

HEALPix is a particular case of a more general class of schemes based on a hierarchical subdivision of a base polyhedron. Another example is the geodesic grids which are based on geodesic polyhedrons, i.e., polyhedrons made of triangular faces. A counter-example is the equirectangular projection, which is not constructed from a base polyhedron, although it can be subdivided. In the particular HEALPix case, the base is a rhombic dodecahedron, i.e., a polyhedron made from 12 congruent rhombic faces. See Figure 5.5 for an illustration of the base rhombic dodecahedron and its subdivisions.

The HEALPix pixelisation produces a hierarchical subdivision of a spherical surface where each pixel covers the same surface area as every other pixel. A hierarchy is desired for the data locality in the computer memory. Equal area is advantageous because white noise generated by the signal receiver gets integrated exactly into white noise in the pixel space. Isolatitude is essential for the implementation of a fast spherical transform. HEALPix is the sole pixelisation scheme which satisfies those three properties.

The lowest possible resolution is given by the base partitioning of the surface into $N_{\text{pix}} = 12$ equal-sized pixels (right-most sphere in Figure 5.5). The resolution changes as $N_{\text{pix}} = 12N_{\text{side}}^2$ such that $N_{\text{pix}} = 48$ for $N_{\text{side}} = 2$ and $N_{\text{pix}} = 192$ for $N_{\text{side}} = 3$. High-resolutions maps easily reach millions of pixels.

5.2.2 Graph construction

Our graph is constructed as an approximation of the sphere S^2, a 2D manifold embedded in $\mathbb{R}^3$. Indeed, [16] showed that the graph Laplacian converges to the Laplace-Beltrami when the number of pixels goes to infinity providing uniform sampling of the manifold and a fully connected graph built with exponentially decaying weights. While our construction does not exactly respect their setting (the sampling is deterministic and the graph is not fully connected), we empirically observe a strong correspondence between the eigenmodes of both Laplacians (see [132, Appendix A]).

From the HEALPix pixelization, we build a weighted undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \boldsymbol{W})$, where $\mathcal{V}$ is the set of $N_{\text{pix}} = |\mathcal{V}|$ vertices, $\mathcal{E}$ is the set of edges, and $\boldsymbol{W}$ is the weighted adjacency matrix. In our graph, each pixel i is represented by a vertex $v_i \in \mathcal{V}$. Each vertex v_i is then connected to the 8 (or 7)[5] vertices v_j which represent the neighboring pixels j of pixel i, forming edges $(v_i, v_j) \in \mathcal{E}$. Given those edges, we define the weighted adjacency matrix $\boldsymbol{W} \in \mathbb{R}^{N_{\text{pix}} \times N_{\text{pix}}}$ as

$$\boldsymbol{W}_{ij} = \begin{cases} \exp\left(-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2^2}{\rho^2}\right) & \text{if pixels } i \text{ and } j \text{ are neighbors,} \\ 0 & \text{otherwise,} \end{cases}$$

where $\boldsymbol{x}_i$ is a vector encoding the 3-dimensional coordinates of pixel i, and

$$\rho = \frac{1}{|\mathcal{E}|} \sum_{(v_i, v_j) \in \mathcal{E}} \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2$$

[5] The $12 \times 4 = 48$ pixels at the corner of each rhombus of the base dodecahedron only have 7 neighboring pixels. See Figures 5.6 and 5.5.

is the average Euclidean distance over all connected pixels. This weighting scheme is important as distances between pixels are not equal. Other weighting schemes are possible. For example, [59] uses the inverse of the distance instead. We found out that the one proposed above works well for our purpose, and did not investigate other approaches, leaving it to future work. Figure 5.6 shows a graph constructed from the HEALPix sampling of a sphere.

5.2.3 Graph Fourier basis

Following [156], the normalized graph Laplacian, defined as $\boldsymbol{L} = \boldsymbol{I} - \boldsymbol{D}^{-1/2}\boldsymbol{W}\boldsymbol{D}^{-1/2}$, is a second order differential operator that can be used to define a Fourier basis on the graph. Here $\boldsymbol{D}$ is the diagonal matrix where $\boldsymbol{D}_{ii} = \boldsymbol{d}_i$ and $\boldsymbol{d}_i = \sum_j \boldsymbol{W}_{ij}$ is the weighted degree of vertex v_i. By construction, the Laplacian is symmetric positive semi-definite and hence can be decomposed as $\boldsymbol{L} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top$, where $\boldsymbol{U} = [\boldsymbol{u}_1, \dots, \boldsymbol{u}_{N_\mathrm{pix}}]$ is an orthonormal matrix of eigenvectors and $\boldsymbol{\Lambda}$ is a diagonal matrix of eigenvalues. The graph Fourier basis is defined as the Laplacian eigenvectors, motivated by the fact that a Fourier basis should diagonalize the Laplacian operator. The graph Fourier transform of a signal $\boldsymbol{f} \in \mathbb{R}^{N_\mathrm{pix}}$ is simply its projection on the eigenvectors given by $\hat{\boldsymbol{f}} = \mathcal{F}_\mathcal{G}\{\boldsymbol{f}\} = \boldsymbol{U}^\top\boldsymbol{f}$. It follows that the inverse graph Fourier transform reads $\mathcal{F}_\mathcal{G}^{-1}\{\hat{\boldsymbol{f}}\} = \boldsymbol{U}\hat{\boldsymbol{f}} = \boldsymbol{U}\boldsymbol{U}^\top\boldsymbol{f} = \boldsymbol{f}$. Note that the Fourier modes are ordered in the increasing order of the Laplacian eigenvalues $\boldsymbol{\Lambda}$, which can be interpreted as squared frequencies. Indeed,

$$\boldsymbol{\Lambda}_{ii} = \boldsymbol{u}_i^\top \boldsymbol{L}\boldsymbol{u}_i = \sum_{(v_j,v_k)\in\mathcal{E}} \frac{\boldsymbol{W}_{jk}}{\sqrt{\boldsymbol{d}_j\boldsymbol{d}_k}}(\boldsymbol{U}_{ji} - \boldsymbol{U}_{ki})^2$$

is a measure of the variation of the eigenvector $\boldsymbol{u}_i$ on the graph defined by the Laplacian $\boldsymbol{L}$.

Figure 5.3 shows the Fourier modes of a HEALPix graph, created using the graph construction described above. The graph Fourier modes resemble the spherical harmonics. That is a strong hint that the graph is able to capture the spherical properties of the HEALPix sampling. This topic is further discussed in [132, Appendix A].

5.2.4 Convolution on graphs

As there is no notion of translation on a graph, we cannot convolve two graph signals in a strict sense. We can, however, convolve a signal with a kernel defined in the spectral

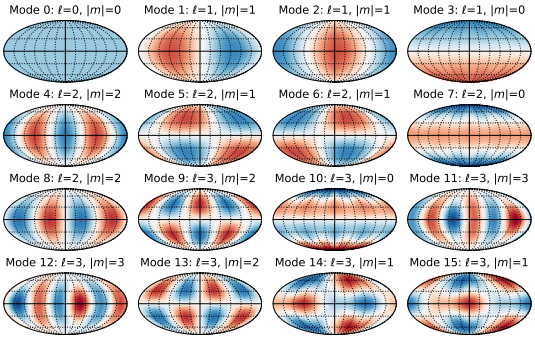


Figure 5.3: The first 16 eigenvectors of the graph Laplacian, an equivalent of Fourier modes, of a graph constructed from the HEALPix sampling of the sphere ($N_{\text{side}} = 16$). Eigenvectors 1–3 could be associated with spherical harmonics of degree $\ell = 1$ and order $|m| = (0, 1)$, eigenvectors 4–8 with degree $\ell = 2$ and order $|m| = (0, 1, 2)$, and eigenvectors 9–15 with degree $\ell = 3$ and order $|m| = (0, 1, 2, 3)$. Nevertheless, graph eigenvectors are only approximating spherical harmonics.

domain. More precisely, we can filter a graph signal by a kernel. Given the convolution kernel $h : \mathbb{R}_+ \to \mathbb{R}$, a signal $\boldsymbol{f} \in \mathbb{R}^{N_{\text{pix}}}$ on the graph is filtered as

$$h(\boldsymbol{L})\boldsymbol{f} = \boldsymbol{U}h(\boldsymbol{\Lambda})\boldsymbol{U}^\top\boldsymbol{f}, \tag{5.1}$$

where $h(\boldsymbol{\Lambda})$ is a diagonal matrix where $\big(h(\boldsymbol{\Lambda})\big)_{ii} = h(\boldsymbol{\Lambda}_{ii})$.

Contrary to classical signal processing on Euclidean domains, the kernel h has no single representation in the vertex domain and cannot be translated on the graph. It can however be *localized* on any vertex v_i by the convolution with a Kronecker delta[6] $\boldsymbol{\delta}^i \in \mathbb{R}^{N_{\text{pix}}}$. The localization operator $\mathcal{T}_i$ reads $\mathcal{T}_i h = h(\boldsymbol{L})\boldsymbol{\delta}^i = \big(h(\boldsymbol{L})\big)_i$, the i^{th} column of $h(\boldsymbol{L})$. This localization of the kernel h can be useful to visualize kernels, as shown in an example of heat diffusion presented in [132, Appendix B]. If the graph is not regular, i.e., all vertices do not have the same number of neighbors, and all distances are not equal, the effect of the kernel will slightly differ from one vertex to another. While there is no perfect sampling of the sphere, these differences are negligible as the structure of the whole graph is very regular. However, when considering only parts of the sphere, one can observe important border effects (see [132, Appendix C]).

[6] A Kronecker delta is the signal $\boldsymbol{\delta}^i \in \mathbb{R}^{N_{\text{pix}}}$ that is zero everywhere except on vertex v_i where it takes the value one.

Finally, the graph convolution can be interpreted in the vertex domain as a scalar product with localizations $\mathcal{T}_i h$ of the kernel h. Indeed, the result of the convolution of the signal $\boldsymbol{f}$ with the kernel h is

$$\big(h(\boldsymbol{L})\boldsymbol{f}\big)_i = \langle \mathcal{T}_i h(\boldsymbol{L}), \boldsymbol{f} \rangle = \langle h(\boldsymbol{L})\boldsymbol{\delta}^i, \boldsymbol{f} \rangle. \tag{5.2}$$

To make the parallel with the classical 1D convolution, let $f, g \in \mathbb{Z} \to \mathbb{R}$ be two 1D discrete signals. Their convolution can be written in the same form as (5.2):

$$(f * g)[i] = \sum_{j=-\infty}^{\infty} f[j]g[i-j] = \langle T_i g, f \rangle,$$

where $T_i g[j] = g[i-j]$ is, up to a flip (i.e., $g[i-j]$ instead of $g[i+j]$), a translation operator. Similarly as (5.2), the convolution of the signal f by a kernel g is the scalar product of f with translated versions $T_i g$ of the kernel g. Additionally, it turns out that the localization operator $\mathcal{T}_i h$ is a generalization of the translation operator on graphs. In the particular case where the Laplacian matrix $\boldsymbol{L}$ is circulant, $\mathcal{T}_i h$ is a translated version of $\mathcal{T}_j h$ for all i, j and both convolutions are equivalent. We refer the reader to [133, Section 2.2] for a detailed discussion of the connection between translation T_i and localization $\mathcal{T}_i$.

To shed some light on the meaning of the convolution on a graph, we show in [132, Appendix B] that the diffusion of heat on a graph can be expressed as the convolution of an initial condition $\boldsymbol{f}$ with a heat kernel h.

5.2.5 Efficient convolutions

While (5.1) is a well justified definition of the convolution, it is computationally cumbersome. As no efficient and general fast Fourier transform (FFT) exists for graphs [104], the execution of the Fourier transform by multiplication of the signal $\boldsymbol{f}$ by the dense matrix $\boldsymbol{U}$ costs $\mathcal{O}(N_{\text{pix}}^2)$ operations. In a NN, this operation has to be performed for each forward and backward pass. As current training procedures require processing of many samples, that would be very slow. Moreover, the eigen-decomposition of the Laplacian $\boldsymbol{L}$ is needed to obtain the Fourier basis $\boldsymbol{U}$ in the first place. That has a unique cost of $\mathcal{O}(N_{\text{pix}}^3)$ operations.

Fortunately, both of these computational issues are overcome by defining the convolution kernel h as a polynomial $h_\theta(\lambda) = \sum_{k=0}^{K} \theta_k \lambda^k$ of degree K parametrised by $K+1$ coefficients θ. The filtering operation (5.1) becomes

$$h_\theta(\boldsymbol{L})\boldsymbol{f} = \boldsymbol{U}\left(\sum_{k=0}^{K} \theta_k \boldsymbol{\Lambda}^k\right)\boldsymbol{U}^\top \boldsymbol{f} = \sum_{k=0}^{K} \theta_k \boldsymbol{L}^k \boldsymbol{f}, \tag{5.3}$$

where $\boldsymbol{L}^k$ captures k-neighborhoods. The entry $(\boldsymbol{L}^k)_{ij}$ is the sum of all weighted paths of length k between vertices v_i and v_j, where the weight of a path is the multiplication of all the edge weights on the path. Hence, it is non-zero if and only if vertices v_i and v_j are connected by at least one path of length k.[7] Filtering with a polynomial convolution kernel can thus be interpreted in the pixel (vertex) domain as a weighted linear combination of neighboring pixel values. In the classical setting, convolutions are also weighted local sums. The weights are however given by the filter coefficients only, and there is one coefficient per pixel in the patch. In the graph setting, the weights are determined by the filter coefficients θ and the Laplacian $\boldsymbol{L}$, and there is one coefficient per neighborhood, not vertex. That defines radial filters, values of which depend only on the distance to the center, and not on the direction. While it may seem odd to restrict filters to be 1D while the sphere is 2D, radial filters are direction-less and result in rotation equivariant convolutions. We note that restricting the graph convolutional kernel to a polynomial is similar to restricting the classical Euclidean convolution to a fixed-size patch. Similarly, each column of the matrix $\sum_{k=0}^{K} \theta_k \boldsymbol{L}^k$ defines an irregular patch of radius K. Hence, filters designed as polynomials of the Laplacian have local support in the vertex domain.

Following [43], we define our filters as Chebyshev polynomials. The filtering operation (5.1) becomes

$$h_\theta(\tilde{\boldsymbol{L}})\boldsymbol{f} = \boldsymbol{U}\left(\sum_{k=0}^{K} \theta_k T_k(\tilde{\boldsymbol{\Lambda}})\right)\boldsymbol{U}^\top \boldsymbol{f} = \sum_{k=0}^{K} \theta_k T_k(\tilde{\boldsymbol{L}})\boldsymbol{f}, \tag{5.4}$$

where

$$\tilde{\boldsymbol{L}} = \frac{2}{\lambda_{\max}}\boldsymbol{L} - \boldsymbol{I} = -\frac{2}{\lambda_{\max}}\boldsymbol{D}^{-1/2}\boldsymbol{W}\boldsymbol{D}^{-1/2}$$

is the rescaled Laplacian with eigenvalues $\tilde{\boldsymbol{\Lambda}}$ in $[-1, 1]$. $T_k(\cdot)$ is the Chebyshev polynomial of degree k defined by the recursive relation $T_k(\tilde{\boldsymbol{L}}) = 2\tilde{\boldsymbol{L}}T_{k-1}(\tilde{\boldsymbol{L}}) - T_{k-2}(\tilde{\boldsymbol{L}})$, $T_1(\tilde{\boldsymbol{L}}) = \tilde{\boldsymbol{L}}$, $T_0(\tilde{\boldsymbol{L}}) = \boldsymbol{I}$. While definitions (5.3) and (5.4) both allow the representation of the same filters, we found in our experiments that optimizing θ in (5.4) is slightly

[7] The length of a path between two vertices defines a distance on the graph.

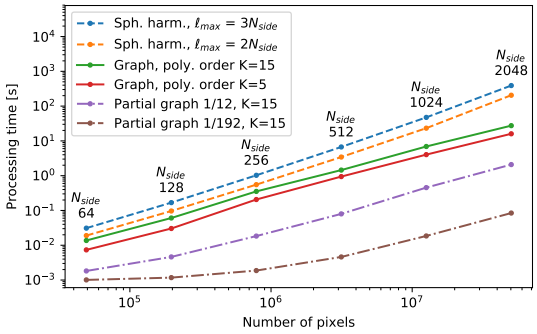


Figure 5.4: Comparison of filtering speed for Gaussian smoothing of maps of various sizes. The fast spherical harmonic transform (SHT) is implemented by the HealPy Python package (via the `healpy.sphtfunc.smoothing` function). The graph filtering is defined by (5.4) and implemented with the NumPy and SciPy Python packages. Both are executed on a single core. The theoretical cost of filtering on the graph is $\mathcal{O}(KN_{\text{pix}})$ and $\mathcal{O}(\ell_{\text{max}}^3) = \mathcal{O}(N_{\text{pix}}^{3/2})$ for the spherical harmonics, where ℓ_{max} is the largest angular frequency. The timings for the partial graphs correspond to a convolution on two fractions (1/12 and 1/192) of the sphere, and illustrates the $\mathcal{O}(N_{\text{pix}})$ scaling of graph convolutions.

more stable than θ in (5.3). We believe that this is due to (i) their almost orthogonality in the spectral and spatial domains, and (ii) their uniformity.[8] Finally, note that while the graph convolution (5.1) is motivated in the spectral domain, definitions (5.3) and (5.4) are implementations in the vertex domain.

Exploiting the recursive formulation of Chebyshev polynomials, evaluating (5.4) requires $\mathcal{O}(K)$ multiplications of the vector $\boldsymbol{f}$ with the sparse matrix $\tilde{\boldsymbol{L}}$. The cost of one such multiplication is $\mathcal{O}(|\mathcal{E}| + |\mathcal{V}|)$. By construction of our graph, $|\mathcal{E}| < 8N_{\text{pix}}$ and the overall computational cost of the convolution reduces to $\mathcal{O}(N_{\text{pix}})$ operations and as such is much more efficient than filtering with spherical harmonics, even though HEALPix was designed as an iso-latitude sampling that has a fast spherical transform. This is especially true for smooth kernels which require a low polynomial degree K. Figure 5.4 compares the speed of low-pass filtering for Gaussian smoothing using the spherical harmonics and the graph-based method presented here. On a single core, a naive implementation of our method is ten to twenty times faster for $N_{\text{side}} = 2048$, with $K = 20$ and $K = 5$, respectively, than using the spherical harmonic transform at $\ell_{\text{max}} = 3N_{\text{side}}$ implemented by the highly optimized Libsharp [140] library used by HEALPix. The further important speed-up of graph convolutions on fractions of the sphere is a direct reflection of the $\mathcal{O}(N_{\text{pix}})$ complexity. On graphs, you only pay for the pixels that you use.

[8] The amplitude of the Chebyshev polynomials $T_k(x)$ is mostly constant over the domain $[-1, 1]$, independently of the order k. On the contrary, the amplitude of the monomials x^k is very different for $|x| \approx 0$ and $|x| \approx 1$.

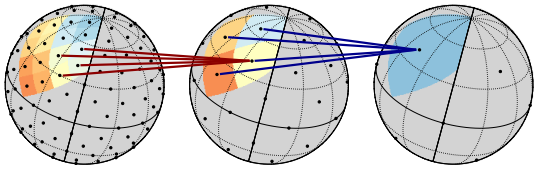


Figure 5.5: Two levels of coarsening and pooling: groups of 4 cells are merged into one, then the data on them is summarized in one. The coarsest cell covers 1/12 of the sphere.

Note that the two might however scale differently, given that Libsharp can be distributed on CPUs through MPI while our method can be distributed on GPUs by TensorFlow.

5.2.6 Coarsening and Pooling

Coarsening can be naturally designed for hierarchical pixelisation schemes, where each subdivision divides a cell in an equal number of child sub-cells. To coarsen, the sub-cells are merged to summarise the data supported on them. Merging cells lead to a coarser graph. Coarsening defines $\mathcal{C}(i)$, the set of children of vertex v_i. For the HEALPix subdivision scheme, the number of children is constant, i.e., $|\mathcal{C}(i)| = 4^p \ \forall i$, for some p.

Pooling refers to the operation that summarizes the data supported on the merged sub-cells in one parent cell. Given a map $\boldsymbol{x} \in \mathbb{R}^{N_{\text{pix}}}$, pooling defines $\boldsymbol{y} \in \mathbb{R}^{N'_{\text{pix}}}$ such that

$$y_i = f\big(\{x_j : j \in \mathcal{C}(i)\}\big), \ \forall i \in [N'_{\text{pix}}], \tag{5.5}$$

where f is a function which operates on sets (possibly of varying sizes) and $N_{\text{pix}}/N'_{\text{pix}}$ is the down-sampling factor, which for HEALPix is

$$|\mathcal{C}(i)| = N_{\text{pix}}/N'_{\text{pix}} = (N_{\text{side}}/N'_{\text{side}})^2 = 4^p,$$

where $p = \log_2(N_{\text{side}}/N'_{\text{side}})$. That operation is often taken to be the maximum value, but it can be any permutation invariant operation, such as a sum or an average. Figure 5.5 illustrates the process.

5.2.7 Layers

Neural networks are constructed as stacks of layers which sequentially transform the data from its raw representation to some predictions. The general DeepSphere architecture, pictured in Figure 5.2, is composed of many layers. The convolutional part, the head of the NN, is composed of graph convolutional layers (*GC*), pooling layers (*P*), and batch normalization layers (*BN*). The tail is composed of multiple fully connected lay-

ers (FC) followed by an optional softmax layer (SM) if the network is used for discrete classification. A non-linear function $\sigma(\cdot)$ is applied after every linear GC and FC layer, except for the last FC layer. That operation is point-wise, i.e., $y_{ij} = \sigma(x_{ij})$ and $y_i = \sigma(x_i)$ for matrices $\boldsymbol{X}, \boldsymbol{Y}$ and vectors $\boldsymbol{x}, \boldsymbol{y}$. The rectified linear unit (ReLU) $\sigma(\cdot) = \max(\cdot, 0)$ is a common choice, and is the one we adopted in this contribution.

Given a matrix $\boldsymbol{X} = [\boldsymbol{x}_1, \dots, \boldsymbol{x}_{F_{\text{in}}}] \in \mathbb{R}^{N_{\text{pix}} \times F_{\text{in}}}$, a GC layer computes $\boldsymbol{Y} = GC(\boldsymbol{X}) = [\boldsymbol{y}_1, \dots, \boldsymbol{y}_{F_{\text{out}}}] \in \mathbb{R}^{N_{\text{pix}} \times F_{\text{out}}}$, where N_{pix} is the number of pixels (and vertices), F_{in} is the number of input features, and F_{out} is the number of output features. Using the efficient graph convolution from (5.4), each output feature map is computed as

$$\boldsymbol{y}_i = \sum_{j=1}^{F_{\text{in}}} h_{\theta_{ij}}(\tilde{\boldsymbol{L}})\boldsymbol{x}_j + b_i \in \mathbb{R}^{N_{\text{pix}}}, \quad \forall i \in [F_{\text{out}}].$$

As such, a GC layer is composed of $F_{\text{in}} \times F_{\text{out}}$ filters, each parameterized by K numbers (see §5.2.5). A bias term $\boldsymbol{b} \in \mathbb{R}^{F_{\text{out}}}$ is jointly optimized.

Given a matrix $\boldsymbol{X} \in \mathbb{R}^{N_{\text{pix}} \times F}$, a pooling layer computes $\boldsymbol{Y} = P(\boldsymbol{X}) \in \mathbb{R}^{N'_{\text{pix}} \times F}$ by reducing its spatial resolution ($N'_{\text{pix}} < N_{\text{pix}}$) according to (5.5). The batch normalization layer [86] computes $\boldsymbol{Y} = BN(\boldsymbol{X})$ such as

$$\boldsymbol{y}_i = \gamma_i \frac{\boldsymbol{x}_i - \mathrm{E}(\boldsymbol{x}_i)}{\sqrt{\mathrm{Var}(\boldsymbol{x}_i) + \epsilon}} + \beta_i, \ \forall i \in [F],$$

where γ_j and β_j are parameters to be learned and ϵ is a constant added for numerical stability. The empirical expectation $\mathrm{E}(\boldsymbol{x}_i) \in \mathbb{R}$ and variance $\mathrm{Var}(\boldsymbol{x}_i) \in \mathbb{R}$ are taken across training examples and pixels.

The layer $FC : \mathbb{R}^{F_{\text{in}}} \to \mathbb{R}^{F_{\text{out}}}$ is defined as

$$\boldsymbol{y} = FC(\boldsymbol{x}) = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}, \tag{5.6}$$

where $\boldsymbol{W} \in \mathbb{R}^{F_{\text{out}} \times F_{\text{in}}}$ and $\boldsymbol{b} \in \mathbb{R}^{F_{\text{out}}}$ are the parameters to be learned. Note that the output $\boldsymbol{Y} \in \mathbb{R}^{N_{\text{pix}} \times F_{\text{out}}}$ of the last GC is vectorized as $\boldsymbol{x} = \mathrm{vec}(\boldsymbol{X}) \in \mathbb{R}^{F_{\text{in}}}$ before being fed to the first FC, where $F_{\text{in}} = N_{\text{pix}} \times F_{\text{out}}$.

The softmax layer is the last layer in a NN engineered for classification. Given the output $\boldsymbol{x} \in \mathbb{R}^{N_{\text{classes}}}$ of the last FC, called the logits in the deep learning literature, the softmax layer outputs $\boldsymbol{y} = SM(\boldsymbol{x})$ such that

$$y_i = \frac{\exp(x_i)}{\sum_{j=1}^{N_{\text{classes}}} \exp(x_j)} \in [0, 1], \ \forall i \in [N_{\text{classes}}],$$

	Classification	Regression
Global	$NN_\theta(\boldsymbol{X}) \in \mathbb{R}^{N_{\text{classes}}}$	$NN_\theta(\boldsymbol{X}) \in \mathbb{R}^{F_{\text{out}}}$
Dense	$NN_\theta(\boldsymbol{X}) \in \mathbb{R}^{N_{\text{pix}} \times N_{\text{classes}}}$	$NN_\theta(\boldsymbol{X}) \in \mathbb{R}^{N_{\text{pix}} \times F_{\text{out}}}$

Table 5.1: The output's size of the neural network $NN_\theta(\boldsymbol{X})$ depends on the task to be solved. N_{pix} is the number of pixels in a HEALPix map, N_{classes} is the number of classes in a classification task, and F_{out} is the number of variables to be predicted in a regression task. The output size of a global task does not depend on the input size, whereas a dense task asks for one prediction per pixel. A classification task asks for discrete (or probabilistic) predictions, whereas a regression task asks for continuous variables.

where N_{classes} is the number of classes to discriminate. Thanks to the softmax, the output $\boldsymbol{y} \in \mathbb{R}^{N_{\text{classes}}}$ of a NN is a discretised conditional distribution for the class given the data and trained parameters. That is, y_i is the confidence of the network that the input sample belongs to class i. This last layer is actually normalizing $\boldsymbol{x}$ into $\boldsymbol{y}$ such that $\|\boldsymbol{y}\|_1 = \sum_i y_i = 1$.

5.2.8 Network architectures

Given a map $\boldsymbol{X} \in \mathbb{R}^{N_{\text{pix}} \times F_{\text{in}}}$, a neural network computes $NN_\theta(\boldsymbol{X})$, where NN is a composition of the above layers and θ is the set of all trainable parameters. The number of input features F_{in} depends on the data. For the CMB radiation temperature, $F_{\text{in}} = 1$. For observations in radio frequencies, F_{in} would be equal to the number of surveyed frequencies. F_{in} might also be the number of slices in the radial direction.

DeepSphere can perform dense or global predictions, for regression or classification. A typical global classification task is to classify maps into cosmological model classes [149]. A typical global regression task is to infer the parameters of a cosmological model from maps [57, 71]. Some dense regression tasks are denoising, interpolation of missing values, or inpainting parts of a map [85]. Segmentation and feature detection [5] are examples of dense classification tasks. The size of the output of the NN depends on the task. See Table 5.1.

Fully convolutional networks (FCNs) have been introduced by [112] and are mostly used for the semantic segmentation of images, a dense classification task. An example FCN for dense regression is

$$\boldsymbol{Y} = NN_\theta(\boldsymbol{X}) = (GC \circ \sigma \circ BN \circ GC)(\boldsymbol{X}) \in \mathbb{R}^{N_{\text{pix}} \times F_{\text{out}}},$$

where $\circ$ denotes composition, i.e., $(f \circ g)(\cdot) = f(g(\cdot))$. If P layers are used, they have to be inverted via up-sampling in an encoder-decoder architecture.

The set of input pixels which influence the value of an output pixel forms a *receptive field*. That field is isotropic and local, i.e., it forms a disk centered around the output pixel. The field's radius is influenced by the polynomial order K of GC layers (setting how far the convolution operation looks around), and the down-sampling factor of P layers, if any. This radius should be large enough to capture statistics of interest. For example, a partial sky observation can provide only limited information of cosmological relevance. On the other hand, looking at the whole sky is often superfluous and waste computations, as most interactions are local. A data and task dependent trade-off is to be found. Note here that while global predictions use, by definition, the whole sky, dense predictions are not necessarily local.

By treating each output pixel independently and in parallel, this architecture is a principled way to perform rotation equivariant operations. Rotation equivariance means that the rotation operation commutes with the NN, i.e., a rotation of the input implies the same rotation of the output.[9]

Global tasks can be solved by averaging dense predictions [111, 157]. Global averaging is computed by the layer $AV : \mathbb{R}^{N_{\text{pix}} \times F} \to \mathbb{R}^F$. Doing so assumes the data is locally independent, and form different observations of an unknown process. Averaging over independent observations is a common way to reduce variance. An example FCN for global regression is

$$NN = AV \circ GC \circ \sigma \circ GC \circ P \circ \sigma \circ BN \circ GC.$$

Here, the FCN predicts parameters for many overlapping parts of the sky in parallel, then average those predictions to get one set of parameters for the whole sky. Global predictions are made invariant to rotation on the $SO(3)$ group by averaging dense equivariant predictions. Rotation invariance means that rotating the input does not impact the prediction.

By replacing the average by a FC layer, one gets the standard convolutional neural network (CNN), which is a generalization of the FCN. Indeed, the AV layer is a FC layer where all the entries of $\boldsymbol{W}$ in (5.6) are equal to $1/N_{\text{pix}}$ and $\boldsymbol{b} = \boldsymbol{0}$. By dropping rotation invariance, CNNs learn where to put attention on the domain. That is useful when pixels are not of the same importance, with respect to the task. For example, on images, the subject of interest is most often around the center of the picture. Hence, those pixels are more predictive than the ones in the periphery when considering image classification. On the sphere, location is important when analyzing weather data on the

[9] Small errors will however appear as GC layers are not exactly equivariant due to the small discrepancy between the graph Fourier modes and the spherical harmonics. See [132, Appendix A].

Earth for example, as oceans and mountains play different roles. An example of such an architecture is

$$NN = FC \circ \sigma \circ F \circ P \circ \sigma \circ BN \circ GC.$$

There are in general two ways to deal with symmetries one wants to be invariant to: (i) build them into the architecture, or (ii) augment the dataset such that a (more general) model learns them. With respect to rotation invariance, the FCN architecture is of the first kind, while the CNN is of the second kind. For tasks that are rotation invariant, FCNs hence need less training data as the rotation symmetry is backed in the architecture and need not be learned. This can be seen as intrinsic data augmentation, as a CNN would need to see many rotated versions of the same data to learn the invariance. Moreover, FCNs can accommodate maps with a varying number of pixels. Such global summarization as the AV layer is commonly used along graph convolutions to classify graphs of varying sizes (as it is invariant to vertex permutation) [52, 110]. As such, CNNs should only be used if rotation invariance is undesired.

All the above architectures can be used for classification (instead of regression) by appending a SM layer. An example FCN for global classification is therefore

$$NN = SM \circ AV \circ GC \circ \sigma \circ GC \circ P \circ \sigma \circ BN \circ GC.$$

Similarly, we emphasize that the use of FC and AV layers is the sole difference between a NN engineered for global or dense prediction.

5.2.9 Training

The cost (or loss) function $C(\boldsymbol{Y}, \bar{\boldsymbol{Y}}) = C(NN_\theta(\boldsymbol{X}), \bar{\boldsymbol{Y}})$ measures how good the prediction $\boldsymbol{Y}$ is for sample $\boldsymbol{X}$, given the ground truth $\bar{\boldsymbol{Y}}$. For a classification task, the cost is usually taken to be the cross-entropy

$$C(\boldsymbol{Y}, \bar{\boldsymbol{Y}}) = -\sum_{i=1}^{N_{\text{pix}}} \sum_{j=1}^{N_{\text{classes}}} \bar{y}_{ij} \log(y_{ij}),$$

where $\bar{\boldsymbol{Y}} \in \mathbb{R}^{N_{\text{pix}} \times N_{\text{classes}}}$ is the ground truth label indicator, i.e., $\bar{y}_{ij} = 1$ if pixel i of sample $\boldsymbol{X}$ belongs to class j and is zero otherwise. For global prediction, we have $N_{\text{pix}} = 1$. For a regression task, a common choice is the mean squared error (MSE)

$$C(\boldsymbol{Y}, \bar{\boldsymbol{Y}}) = \frac{\|\boldsymbol{Y} - \bar{\boldsymbol{Y}}\|_2^2}{N_{\text{pix}} F_{\text{out}}} = \frac{1}{N_{\text{pix}} F_{\text{out}}} \sum_{i=1}^{N_{\text{pix}}} \sum_{j=1}^{F_{\text{out}}} (y_{ij} - \bar{y}_{ij})^2,$$

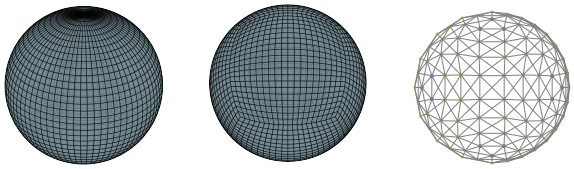


Figure 5.6: Some pixelizations of the sphere. Left: the equirectangular grid, using equiangular spacing in a standard spherical-polar coordinate system. Middle: an equiangular cubed-sphere grid, as described in [144]. Right: graph built from a HEALPix pixelization of half the sphere ($N_{\text{side}} = 4$). By construction, each vertex has eight neighbors, except the highlighted ones which have only seven.[5] Left and middle figures are taken from [21].

where $\bar{\boldsymbol{Y}}$ is the desired output. Again, take $N_{\text{pix}} = 1$ for global regression. We emphasize that the cost function and the *SM* layer are the sole differences between a NN engineered for classification or regression.

The goal of training is to find the parameters θ of the NN that minimize the risk $R(\theta) = \mathrm{E}\big[C\big(NN_\theta(\boldsymbol{X}), \bar{\boldsymbol{Y}}\big)\big]$, where E is the expectation over the joint distribution $(\boldsymbol{X}, \bar{\boldsymbol{Y}})$. In general, that expectation cannot be computed as the data distribution is unknown. We can however minimize an approximation, the empirical risk over the training set $\big\{\big(\boldsymbol{X}_i, \bar{\boldsymbol{Y}}_i\big)\big\}_{i=1}^{N_{\text{samples}}}$:

$$\hat{\theta} = \arg\min_\theta \sum_{i=1}^{N_{\text{samples}}} C\big(NN_\theta(\boldsymbol{X}_i), \bar{\boldsymbol{Y}}_i\big).$$

The optimization is performed by computing an error gradient w.r.t. all the parameters by back-propagation and updating them with a form of stochastic gradient descent (SGD):

$$\theta \leftarrow \theta - \frac{\eta}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \frac{\partial C\big(NN_\theta(\boldsymbol{X}_i), \bar{\boldsymbol{Y}}_i\big)}{\partial \theta},$$

where η is the learning rate, and $\mathcal{B}$ is the set of indices in a mini-batch. Batches are used instead of single samples to gain speed by exploiting the parallelism afforded by modern computing platforms.

5.3 Related work

5.3.1 2D convolutional neural networks

A first approach, explored by [21] for molecular modeling and [39, 160] for omnidirectional imaging, is to use a 2D CNN on a discretisation of the sphere that is a grid, such as the equirectangular projection (Figure 5.6, left panel), or a set of grids, such

as the cubed-sphere defined by [144] (Figure 5.6, middle panel). As this formulation uses the standard 2D convolution, all the optimizations developed for images can be applied, which makes it computationally efficient. This approach is applicable to the many pixelizations that are based on a regular subdivision of a base polyhedron, such as HEALPix. Each base polyhedron then forms a grid. Care has to be taken to handle boundary conditions: for example by padding a grid with the content of the opposite side (equirectangular) or of the neighboring grids (cubed-sphere, HEALPix). That incurs some computational losses. For samplings that are not equal area, such as the equirectangular projection, the convolution operation should be adjusted to take the induced distortion into account [39, 160].

Another approach to leverage 2D CNNs is to project spherical data onto many tangent planes, which are flat 2D surfaces. This approach has been extensively used for cosmological maps [57, 63, 71, 149] and omnidirectional imaging [170, 171]. This idea has been generalized to arbitrary 2D manifolds for shape alignment and retrieval [23, 120, 123].

The main issue with the above two approaches is that they depend on a (local) coordinate system to define anisotropic filters, i.e., filters which are direction dependent. Direction is well defined and matters for some applications, such as the analysis of weather and climate data on the Earth (north, south, east, west), and omnidirectional imaging (up, down). Indeed, rotation invariance has been shown to reduce discriminative power for omnidirectional imaging [39]. Some problems are, however, intrinsically invariant (or equivariant) to rotation. Examples include the analysis of cosmological maps, and the modeling of atoms and molecules. In such cases, directions are arbitrarily defined when setting the origin of the pixelization. Therefore, a convolution operation has to be isotropic to be equivariant to rotation.

5.3.2 Spherical neural networks

Rotation equivariance was addressed by leveraging the convolution associated to the 3D rotation group $SO(3)$, with applications to atomization energy regression and 3D model classification, alignment and retrieval [34, 55]. The resulting convolution is performed by (i) a spherical harmonic transform (SHT), i.e., a projection on the spherical harmonics, (ii) a multiplication in the spectral domain, and (iii) an inverse SHT. Note the similarity with the naive graph convolution defined in (5.1). Likewise, the computational cost of a convolution is dominated by the two SHTs, and a naive implementation of the SHT costs $\mathcal{O}(N_{\text{pix}}^2)$ operations. Accelerated schemes however exist for some sam-

pling sets (see [122, 140, 142] for examples). The convolutions remain nevertheless expensive, limiting the practical use of this approach. For example with HEALPix, which was designed to have a fast SHT by being iso-latitude, the computational cost of the SHT is $\mathcal{O}(N_{\text{pix}}^{3/2}) = \mathcal{O}(N_{\text{side}}^3) = \mathcal{O}(\ell_{\text{max}}^3)$, where ℓ_{max} is the largest angular frequency [67, 140].[10] While the Clebsh-Gordan transform can be leveraged to avoid SHTs between layers (it does a non-linear transformation in the spectral domain), the transform itself costs $\mathcal{O}(N_{\text{pix}}^{3/2})$, for no reduction of overall complexity [97]. This work is another indication that there might be a $\Omega(N_{\text{pix}}^{3/2})$ computational lower bound for the proper treatment of rotation equivariance with the spherical harmonics. In comparison, DeepSphere scales as $\mathcal{O}(N_{\text{pix}})$ (see §5.2.5), at the expense of not being exactly equivariant (see [132, Appendix A]). Being a mathematically well-defined rotation equivariant network is the main advantage of methods based on spherical harmonics, though the fact that convolution kernels are learned probably compensates for inexact equivariance. See [98] for a rigorous treatment of convolution and equivariance in NNs.

While defining the convolution in the spectral domain avoids all sampling issues, filters so defined are not naturally localized in the original domain. Localization is desired for the transformation to be stable to local deformation [116], and most interactions in the data are local anyway. A straightforward way to impose locality in the original domain is to impose smoothness in the spectral domain, by Heisenberg's uncertainty principle.[11] A more elegant approach is to define filters that are provably localized. The filters defined in (5.3) and (5.4), by being polynomials of the Laplacian, are of this kind.

All the presented 2D and spherical CNNs cannot be easily accelerated when the data lies on a part of the sphere only. That is an important use case in cosmology as measurements are often partial, i.e., whole sky maps are rare. One could still fill the unseen part of the sphere with zeros, and avoid computing empty integrals. It is, however, not straightforward to identify empty space, and computations would still be wasted (for example on a ring that mostly contains zeros but a few measurements). With graphs, however, computations are only performed for used pixels. While it results in some

[10] All pixels are placed on $N_{\text{ring}} = 4N_{\text{side}} - 1 = \mathcal{O}\left(\sqrt{N_{\text{pix}}}\right)$ rings of constant latitude. Each ring has $\mathcal{O}\left(\sqrt{N_{\text{pix}}}\right)$ pixels. Thanks to this iso-latitude property, the SHT is computed using recurrence relations for Legendre polynomials on co-latitude and fast Fourier transforms (FFTs) on longitude. The computational cost is thus $\sqrt{N_{\text{pix}}}$ FFTs for a total cost of $\mathcal{O}\left(N_{\text{pix}} \log \sqrt{N_{\text{pix}}}\right)$, plus $\sqrt{N_{\text{pix}}}$ matrix multiplications of size $\sqrt{N_{\text{pix}}}$ for a total cost of $\mathcal{O}\left(N_{\text{pix}}^{3/2}\right)$ operations.

[11] Heisenberg's uncertainty principle states that a filter cannot be arbitrarily concentrated in one domain without being de-concentrated in the other.

distortions due to border effects (see [132, Figure C.15 and Appendix C]), these can be mitigated by padding with zeros a small area around the measurements.

5.3.3 Graph neural networks

The use of a graph to model the discretised sphere was also considered for omnidirectional imaging [59]. This work is the closest to our method, with three differences. First, they parametrize their convolution kernel with (5.3) instead of (5.4) (see §5.2.5 for a discussion and comparison). Second, they did not take advantage of a hierarchical pixelization of the sphere and resorted to dynamic pooling [90]. While that operation has proved its worth to pool sequences of varying length, such as sentences in language models, it is undesired in our context as it is not local and destroys spatial coherence. Third, they introduced a statistical layer — an operation that computes a set of statistics from the last feature maps — to provide invariance to rotation. We propose to use the idea of fully convolutional networks (FCNs) instead (see §5.2.8). While statistics have to be hand-chosen to capture relevant information for the task, the filters in a FCN are trained end-to-end to capture it.

Many formulations of graph neural networks, reviewed by [24] and [73], have been proposed. For this contribution, we chose the formulation of [43] as its root on strong graph signal processing theory makes the concept of convolutions and filters explicit [156]. As the convolution is motivated by a multiplication in the graph Fourier spectrum, it is close in spirit, and empirically, to the formulation based on the spherical harmonics, which is the ideal rotation equivariant formulation.

Thanks to their versatility, graph neural networks have been used in a variety of tasks, such as identifying diseases from brain connectivity networks [102] or population graphs [127], designing drugs using molecular graphs [81], segmenting 3D point clouds [137], optimizing shapes to be aerodynamic [9], and many more. By combining graph convolutional layers and recurrent layers [150], they can, for example, model structured time series such as traffic on road networks [108], or recursively complete matrices for recommendation [124]. Another trend, parallel to the modeling of structured data, is the use of graph neural networks for relational reasoning [12].

5.4 Experiments

The performance of DeepSphere is demonstrated on a discrimination problem: the classification of convergence maps into two cosmological model classes. The experiment

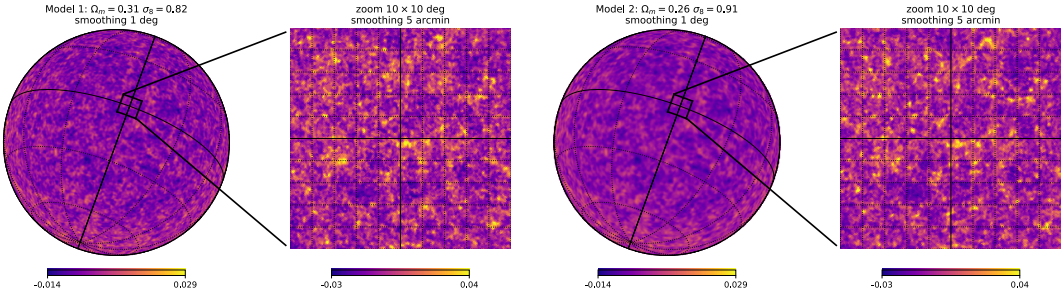


Figure 5.7: Example maps from two classes to be discriminated. Left: model 1 with $\Omega_m = 0.31$ and $\sigma_8 = 0.82$. Right: model 2 with $\Omega_m = 0.26$ and $\sigma_8 = 0.91$. The initial conditions for both simulations are the same, so differences only arise due to different cosmological parameters.

presented here is similar to the one by [149]. These maps are similar to those created with gravitational lensing techniques [29]. The two sets of maps were created using the standard cosmological model with two sets of cosmological parameters (see §5.4.1 for details). The classification methods were trained to predict a label from a HEALPix map. Our Python implementation to reproduce those experiments is openly available online.[12] The data is available upon request.[13]

While we only demonstrate a classification task here, other tasks, such as regression or segmentation, are also possible (see §5.2.8). The regression task will most likely be the most practical cosmological application, as described in [71] and [57]. Implementation of full cosmological inference typically requires, however, many more simulations, building the likelihood function, and several other auxiliary tasks. The classification problem is much more straightforward to implement and execute, and can be used to fairly compare the accuracy of the algorithm against benchmarks [149]. For these reasons we decided to use the classification problem in this work, and we expect the relative performance of the methods to generalise to the regression scenario.

5.4.1 Data

Convergence maps represent the dimensionless distribution of over- and under-densities of mass in the universe, projected on the sky plane. The 3D structures are projected using a geometric kernel, the value of which depends on the radial distance. In gravitational lensing, this kernel is dependent on the radial distances between the observer,

[12] https://github.com/SwissDataScienceCenter/DeepSphere
[13] https://doi.org/10.5281/zenodo.1303272

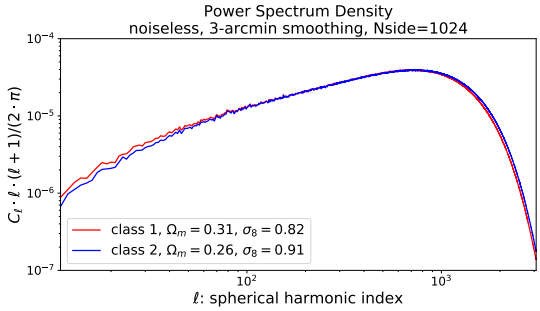


Figure 5.8: Power spectral densities of the noiseless maps. To prevent the cosmological models from being distinguished by their power spectra alone, the maps have been smoothed with a Gaussian kernel of radius 3 arcmins to remove high frequencies ($\ell > 1000$).

the mass plane, and the plane of source galaxies (see [10] for a review of gravitational lensing).

We make whole sky N-body simulations for two parameter sets of the ΛCDM cosmological model: model 1 ($\Omega_m = 0.31, \sigma_8 = 0.82$) and model 2 ($\Omega_m = 0.26, \sigma_8 = 0.91$), where Ω_m is the matter density in the universe and σ_8 is the normalisation of the matter power spectrum. Other parameters are set to: Hubble constant $H_0 = 70$ km/s/Mpc, spectral index $n_s = 0.96$, and Baryon density today $\Omega_b = 0.05$. The parameters Ω_m and σ_8 were chosen for the maps to have the same spherical harmonic power spectrum. That means that it is difficult to distinguish between these cosmological models. We found that the differences in power spectrum is $\lesssim 5\%$ for $\ell > 1000$. To remove this information, we additionally smooth the spherical maps with a Gaussian kernel of radius 3 arcmin. The resulting power spectral density (PSD), computed using the `anafast` function of the HEALPix package, are displayed in Figure 5.8. We also subtract the mean of each map and down-sample them to a resolution of $N_{\text{side}} = 1024$, which corresponds to maps of $12 \times 1024^2 \approx 12 \times 10^6$ pixels. As shown by the occupied spectrum (Figure 5.8), a larger resolution would waste memory and computation, while a lower resolution would destroy information.

The simulations are created using the fast lightcone method UFalcon described in [151]. A brief overview about the map making procedure used in UFalcon as well as the simulation parameters are given in [132, Appendix F]. We however use a single simulation box, as opposed to two used in that work, as we use source galaxies at a lower redshift of $z = 0.8$, instead of $z = 1.5$. L-PICOLA [83] is used for fast and approximate N-body simulations. We generate 30 simulations for each of the two classes. Out of the 60 simulations, 20 are kept as the test set, and 20% of the remaining training data is used as a validation set, to monitor the training process and select the hyper-parameters. Fig-

ure 5.7 shows the whole sky simulations and a zoom region for both models. Initial conditions for these simulations are the same, so the differences in structures can only be attributed to different cosmological parameters used to evolve the particle distribution.

5.4.2 Problem formulation

As the distribution of matter in the universe is homogeneous and isotropic, no pixel is more informative than any other.[14] As such, we can control the difficulty of the classification problem by limiting the number of pixels available to the algorithms, i.e., extracting partial maps from whole sky maps. Using the properties of HEALPix, we split maps into $12 \times o^2$ independent samples of $(N_{\text{pix}}/o)^2$ pixels (for $o = 1, 2, 4, \dots$). The resulting partial maps are large enough to suffer from the effects of the spherical geometry, and cover 8.3% ($\approx 1{\times}10^6$ pixels), 2.1% ($\approx 260{\times}10^3$ pixels), and 0.5% ($\approx 65{\times}10^3$ pixels) of the sphere for $o = 1, 2, 4$, respectively. Corresponding areas can be seen in Figure 5.5: the surface of pixels in the left, middle, and right spheres correspond to samples at order $o = 4, 2, 1$, respectively. We report results for $o = 1, 2, 4$ only, as full sphere classification is easy at such resolution (perfect classification accuracy is already obtained at $o = 1$, see Figure 5.9). The published code nonetheless includes an example demoing classification on full spheres of lower resolution ($N_{\text{side}} = 64$).

To make the discrimination harder and the problem more realistic, centered Gaussian noise is added to the simulated maps. The standard deviation of the noise varies from zero (i.e., no noise) to 2× the standard deviation of pixel's values in the noiseless maps. While the noise model of real maps often has a slightly different distribution, Gaussian noise should be a sufficient model to demonstrate the performance of our method. To avoid over-fitting, random noise is generated during training, so that no two samples are exactly the same (i.e., two samples might be from the same simulation, but the added noise will be different). That is a data augmentation scheme, as it creates more training examples than we have simulations.

5.4.3 Baselines

DeepSphere is first compared against two simple yet powerful benchmarks. The two baselines are based on features that are (i) the power spectral densities (PSDs) of maps, and (ii) the histogram of pixels in the maps [130]. After normalization, those features are used by a linear support vector machine (SVM) trained for classification. A linear kernel

[14] Contrast that with images, where the subject of interest is most often around the center of the picture.

is used as other kernels did not provide better results, while having significantly worse scaling properties. For fairness, the training set was augmented in a similar way as for DeepSphere: we created samples by adding new random realizations of the noise. We stopped adding new samples to the training data once the validation error converged. This process is detailed in [132, Appendix E]. As for DeepSphere, the SVM regularization hyper-parameter was tuned based on the performance over the validation set. The classification accuracy of DeepSphere and both baselines was checked as a function of the used area (the order o), and the relative level of additive noise.

DeepSphere is further compared to a classical CNN for 2D Euclidean grids, later referred to as 2D ConvNet. To be fed into the 2D ConvNet, samples have to be transformed into flat images. As described in [132, Appendix D], we use the property that HEALPix is defined as the subdivision of 12 square surfaces to efficiently project the spherical maps. This property has been independently exploited (while this work was under review) to define another spherical CNN [100]. In our experiments, we used the same architecture as DeepSphere (described in §5.4.4), with convolution kernels of size 5×5 and an $\ell 2$ weight decay with a weight of 3 as regularization. The number of filters was reduced in order to match the number of trainable parameters between DeepSphere and the 2D ConvNet. (A larger architecture resulted in unstable training for a marginal performance gain.)

Ideally, we would compare DeepSphere to alternative spherical CNNs like [34] and [55]. There are however two issues with that comparison. First, these CNNs were developed for the equirectangular sampling. An option is to modify the available implementations to use HEALPix, which is a major undertaking due to their complexity. Another option is to transform the cosmological maps to the equirectangular sampling, which is of disputable usefulness as the field is unlikely to use that sampling. Second, those methods, based on SHTs, only work on the full sphere (at least in their current state). As shown in Figure 5.4, using the full sphere at a resolution of $N_{\text{side}} = 1024$ would result in a slowdown of multiple orders of magnitude compared to using a partial graph. The authors of [77] could not use [34] for CMB analysis because of that.[15] For those reasons and because we think that a proper comparison is better carried out on diverse datasets and tasks, we leave it as future work.

[15] Note that reported results in [34] and [55] are for maps of at most $(2 \times 128)^2 = 65,536$ pixels. That is the size of our smallest partial map, while the largest is of approximately one million pixels.

5.4.4 Network architecture and hyper-parameters

As discussed in §5.2.8, the choice of a network architecture depends on the data and task at hand. For our problem, where we assume that each pixel carries the same quantity of information about the cosmological model, a rotation invariant FCN architecture should be best. The selected network is defined as

$$FCN = \underbrace{SM \circ AV \circ GC_2}_{\text{classifier}} \circ \underbrace{L_5 \circ L_4 \circ L_3 \circ L_2 \circ L_1}_{\text{feature extractor}},$$

$$\text{with} \quad \begin{aligned} L_1 &= P_4 \circ \sigma \circ BN \circ GC_{16}, \\ L_2 &= P_4 \circ \sigma \circ BN \circ GC_{32}, \\ L_3, L_4, L_5 &= P_4 \circ \sigma \circ BN \circ GC_{64}, \end{aligned}$$

where GC_F indicates a graph convolutional layer with F output feature maps, P_4 is a pooling layer that divides the number of pixels by 4, σ is a ReLU, and the Chebyshev polynomials in (5.4) are of degree $K = 5$. The layers L_1 to L_5 build the statistical evidence to classify each pixel of a down-sampled sphere. The last GC layer acts as a linear classifier and outputs two predictions. The AV layer averages these predictions and the SM layer normalizes them.

For comparison, we tried the more conventional CNN architecture, where the classification is performed by a fully connected layer from the last feature maps instead of the average of local classifications. This architecture is not invariant to rotation. The selected network is defined as

$$CNN = \underbrace{SM \circ FC_2}_{\text{classifier}} \circ \underbrace{L_5 \circ L_4 \circ L_3 \circ L_2 \circ L_1}_{\text{feature extractor}},$$

where FC_2 denotes a fully connected layer with two outputs. Architectures were selected over their performance on the validation set.

The Adam optimizer [94] with $\beta_1 = 0.9$, $\beta_2 = 0.999$ was used with an initial learning rate of 2×10^{-4} that is exponentially decreased by a multiplication with 0.999 at each step. All models were trained for 80 epochs, which corresponds to 1920 steps. The batch size is set to $16 \times o^2$ to keep the amount of supervision used to estimate the gradient identical, irrespective of the chosen order o (the learning rate would otherwise need to be adapted). By this choice, DeepSphere is trained with the same amount of information, i.e., the number of pixels, across all variants of the problem. Training took approximately 1.5 hour using a single Nvidia GeForce GTX 1080 Ti and 5 hours with a Tesla K20.

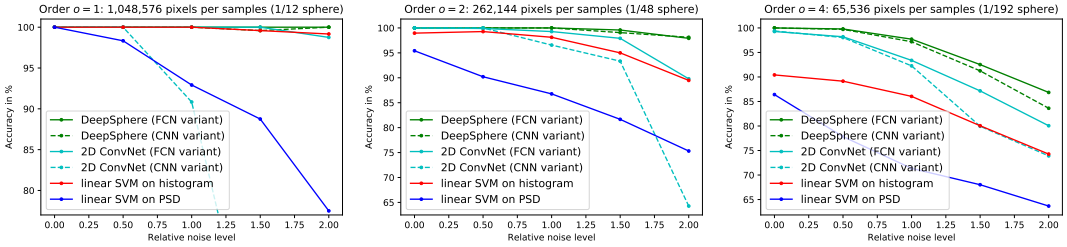


Figure 5.9: Classification accuracy of the fully convolutional variant of DeepSphere (DeepSphere FCN), the standard convolutional variant of DeepSphere (DeepSphere CNN), the fully convolutional 2D ConvNet (2D ConvNet FCN), the standard 2D ConvNet (2D ConvNet CNN), the support vector machine (SVM) with the power spectral density (PSD) as features, and the SVM with the histogram as features. The difficulty of the task depends on the level of noise and the size of a sample (that is, the number of pixels that constitute the sample to classify). Order $o = 1$ corresponds to samples which area is $\frac{1}{12} = 8.1\%$ of the sphere, order $o = 2$ to $\frac{1}{12 \times 2^2} = 2.1\%$, and order $o = 4$ to $\frac{1}{12 \times 4^2} = 0.5\%$. The standard deviation of the added Gaussian noise varies from zero to 2× the standard deviation of pixel's values in the noiseless maps. It is clear from those results that the noise makes the problem harder, and that having more pixels available to classify a sample makes the problem easier (the classifier having more evidence to make a decision). The FCN variant of DeepSphere beats the CNN variant by being invariant to rotation. Both variants largely beat the 2D ConvNet and the two SVM baselines.

5.4.5 Results

Figure 5.9 compares the classification accuracy of DeepSphere (FCN and CNN variants), and the three baselines across five noise levels and three sample sizes. We indeed observe that the problem is made more difficult as the sample size decreases, and the noise increases. As fewer pixels are available to make a decision about a sample, the algorithms have access to less information and thus cannot classify as well. As the noise increases, the useful information gets diluted in irrelevant data, i.e., the signal-to-noise ratio (SNR) diminishes.

As the cosmological parameters were chosen for the maps to have similar PSDs (see Figure 5.8), it is reassuring to observe that an SVM with those as features has difficulties discriminating the models. Other statistics are therefore needed to solve the problem. Using histograms as features gives a significant improvement. Performance is very good for larger maps and deteriorates for the smaller cases with increased noise level, reaching 80% at the highest noise level for $o = 4$. The histogram features contain information about the distribution of pixels, which clearly varies for the two classes. They do not, however, include any spatial information. DeepSphere (both the FCN and CNN variants) shows superior performance over all configurations. The gap widens as the

problem becomes more difficult. This is likely due to DeepSphere learning features that are adapted to the problem instead of relying on hand-designed features. The accuracy of DeepSphere is $> 97\%$ for orders $o = 1$ and $o = 2$, for all noise levels, and starts to deteriorate for $o = 4$, reaching 90% for the highest noise level.

It may seem unfair to compare DeepSphere, who has access to raw pixels, with SVMs who only see limited features (histograms and PSDs). We however trained an SVM on the raw pixels and were unable to obtain over 60% accuracy in any of the three noiseless cases.

The 2D ConvNet fairs in-between the SVMs and DeepSphere. Its relatively low performance probably come from the following drawbacks. First, it does not exploit the rotational invariance of the problem. We observed that the learned convolution kernels were not radial. Second, the 2D projection distorts the geometry, and the NN has to learn to compensate for it, which comparatively requires more training data.

As expected, the FCN variants outperform the CNN variants (both for DeepSphere and the 2D ConvNet). This may seem counterintuitive as the CNN is a generalization of the FCN and hence should be able to learn the same function and provide at least equivalently good results. Nevertheless, as discussed in §5.2.8, the larger number of parameters incurred by the increased flexibility requires more training data to learn the rotation symmetry. The superior performance of the FCN is an empirical validation that these maps are stationary with a small-support radial autocorrelation function, stemming from the fact that the mass distribution is homogeneous and isotropic. It also implies that this classification problem is invariant to rotation. Hence, a pixel can be statistically classified using only its surrounding, and those local predictions can be averaged to vote for a global consensus. The CNN variant, however, may be better for data that does not have this property, as the architecture should always be adapted to the data and task (see §5.2.8).

While testing the FCN and CNN variants of DeepSphere, we made the following empirical observations. First, training was more stable (in the sense that the loss decreased more smoothly) when using Chebyshev polynomials, as in (5.4), rather than monomials, as in (5.3). Nevertheless, we could not observe a significant difference in accuracy after convergence. Second, using ℓ_2 regularization does not help either with performance or training of the models, mostly because the models are not over-fitting. Third, we recommend initializing the Chebyshev coefficients with a centered Gaussian distribution with standard deviation $\sqrt{2/(F_{\text{in}} \times (K + 0.5))}$, where K is the degree of the Chebyshev polynomials and F_{in} the number of input channels. This standard deviation has the property of keeping the energy of the signal more or less constant across

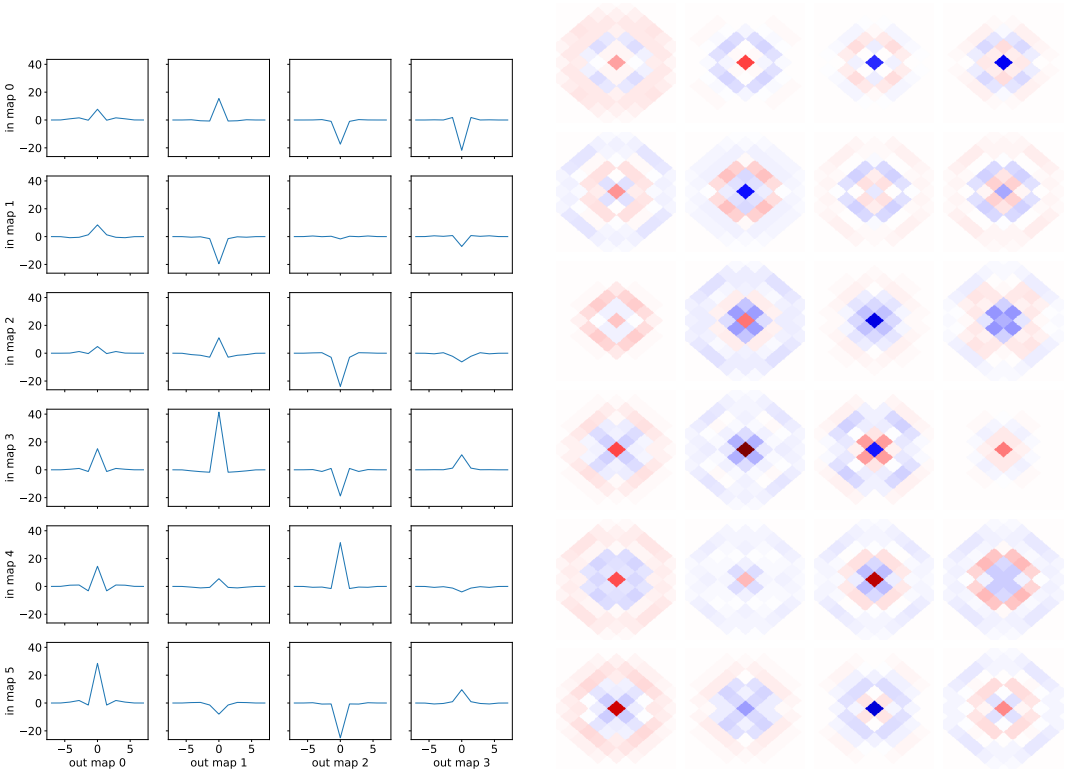


Figure 5.10: Random selection of 24 learned filters from the fifth spherical convolutional layer L_5. Top: section. Bottom: gnomonic projection. The structure in the filters resembles peaks, which is not unexpected, given that the convergence maps largely consist of concentrated clumps.

layers.[16] Finally, we observe that scaling the Laplacian's eigenvalues between $[-a, a]$, where $0 < a \leq 1$, significantly helps in stabilizing the optimization. We use $a = 0.75$ in our experiments.

5.4.6 Filter visualization

A common visualization to introspect and try to understand how a CNN function is to look at the learned filters. Since our construction leads to almost (up to sampling irregularities) spherical filters, we plot in Figure 5.10 both the radial profile and a gnomonic projection on a plane of a random selection of learned filters from the last layer of the network. While those particular filters were obtained from the experiment with order $o = 2$ and a relative noise level of 2, all trained networks presented visually similar

[16] We derived this rule from the principles presented in [65], the Chebyshev polynomial equations, and some empirical experiments.

patterns. Details of how the convolution kernels are plotted are described in [132, Appendix B]. While it is usually difficult to interpret the shape of the filters, especially given the type of data, we can notice that they often have a "peak"-like structure. An example of filter interpretation was demonstrated in [141].

5.5 Conclusion

We present DeepSphere, a convolutional neural network defined on the sphere with HEALPix sampling, designed for the analysis of cosmological data. The main contributions of this chapter are (i) to show that spherical CNNs are a great NN architecture for cosmological applications, and (ii) that a graph-based spherical CNN has certain undeniable advantages. The core of our method is the use of a graph to represent the discretised sphere. This allows us to leverage the advantages of the graph convolution. It is both efficient, with a complexity of $\mathcal{O}(N_{\text{pix}})$, and flexible, which allows DeepSphere to efficiently work on a partial sphere. The spherical properties of the domain are well captured: the graph Fourier modes are close to the spherical harmonics. Filters are restricted to be radial for the convolution operation to be equivariant to rotation. DeepSphere can then be made either equivariant or invariant to rotation. Equivariance is not perfect as small imprecisions due to the sampling cause the action of a graph filter to slightly depend on the location. However, we do not expect that to cause problems for practical applications.

We demonstrate that DeepSphere systematically and significantly outperforms three benchmark methods on an example problem of cosmological model discrimination with weak lensing mass maps, designed similarly to [149]. The maps were produced from two cosmological models with varying σ_8 and Ω_m, chosen to follow the typical weak lensing degeneracy in these parameters, so that the power spectra for these models are similar. We compared the performance of DeepSphere versus that of a classical 2D CNN and two SVM classifiers (one trained on the spherical harmonics power spectrum, and the other on the pixel density histogram). DeepSphere performs better than the three baselines for all considered cases. The advantage is small for large, noise-free maps, and grows up to 10% for smaller, noisier data.

Spherical CNNs are so far mostly used for omnidirectional imaging. Many scientific fields, such as weather or climate modelling, would however benefit from an efficient and versatile spherical CNN. With this work, we hope to demonstrate and help democratize the use of those tools for spherical data analysis. We publish the code as a small and easy-to-use python package. The code was designed so that DeepSphere can easily

be used for typical machine learning tasks, such as classification and regression, both dense and global.

As future work, it would be interesting to further investigate the relation between the graph Fourier basis and the spherical harmonics. The goal would be to find edge weights such that the graph Laplacian converges (or is equivalent) to the Laplace-Beltrami (up to a certain bandwidth). Ideally, that should work for any sampling of the sphere. That would enable a truly rotation equivariant graph convolution and high-precision filtering using the graph rather than the SHT (for speed). Finally, a comparison of DeepSphere to other spherical CNN formulations, with different sampling schemes, would be worthwhile.

The fact that a graph representation of the discretised sphere enables an efficient convolution relaxes the iso-latitude constraint on the design of sampling schemes which aim to enable fast convolutions. In the long term, graphs might enable researchers to consider sampling schemes with different trade-offs, or remove the need for schemes and interpolation altogether and simply consider the positions at which measures were taken.

6 Conclusion

When learning from data, leveraging the symmetries of the domain the data lies on is a principled way to combat the curse of dimensionality: it constrains the set of functions to learn from. It is more data efficient than augmentation and gives a generalization guarantee, a guarantee that tropical cyclones will be recognized regardless of their location and orientation.

From the building blocks of space (vertices, edges, simplices), an incidence structure B, and a metric M—the domain's topology and geometry—we have a discrete calculus and the Laplacian $L = B^\mathsf{T} M B$. As domain symmetries leave L unchanged, they must act as rotations within its eigenspaces. The spectral basis that diagonalizes the Laplacian as $L = U\Lambda U^{-1}$ hence jointly block-diagonalizes these symmetry group actions. The *generalized convolution operator* $g(L) = Ug(\Lambda)U^{-1}$, where $g(\Lambda) = \operatorname{diag}\big(g(\lambda_1), \dots, g(\lambda_n)\big)$ is a kernel evaluated at the Laplacian's eigenvalues, emerges as the linear operator that commutes with any potential symmetry action—without knowing them. It is an equivariant operator. As in classical signal processing and harmonic analysis, the convolution operator $g(L)$ in the spatial basis becomes the multiplication operator $g(\Lambda)$ in the spectral basis, where the change-of-basis is performed by the Fourier transforms U and U^{-1}. In our generalized setting involving non-transitive and unknown symmetry groups, the convolution is however an inner-product with a kernel that is localized instead of moved around by group actions like translations.

While the kernel g is domain independent—hence transfer across domains—we can use g to probe them. The number of distinct eigenvalues measures domain complexity: an 8-vertices path graph can express kernels with 8 degrees of freedom while the additional symmetries of a cycle reduces that to 5. By adding a single edge! Information about the domain is obtained by convolving indicator functions, yielding invariant positional (global) and structural (local) features as well as centrality and distance measures. The kernel g can be designed for $g(L)$ to propagate waves or compute resistance distances. Or it can be expanded in a truncated spectral or polynomial basis (for the kernel to have global or local spatial support) whose weights are learned from data.

With $g(L)$, an interpolation operator between finer and coarser discrete domains (perhaps with non-linear pooling), and a non-linear activation function, we build a convolutional neural network. Its implementation boils down to the multiplications of data tensors by sparse matrices and pointwise operations. These have linear compute and memory requirements, along with sparse communication requirements allowing distributed implementations. Unlike group convolutions, one does not need to explicitly know the (potentially expensive to find) symmetries. Neither does one need an homogeneous domain to share weights around. There is of course no free lunch and our kernels have an additional constraint: they are isotropic. This bias–variance tradeoff paves the way to efficient learning on arbitrary discrete domains.

To avoid the standard but inflexible and wasteful practice of interpolating to discretizations with nice properties, we developed DeepSphere to enable practitioners to efficiently represent spherical data by placing vertices where it matters. In comparisons with alternatives, including group convolutions, we found DeepSphere to reach state-of-the-art performance while being more memory and compute efficient. These results suggest that anisotropic kernels might be an unnecessary price to pay. DeepSphere has been used for studies in cosmology and shall be used for operational weather forecasting—advancing our understanding of the world and impacting billions of individuals.

Bibliography

1. M. Abadi et al. “TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems”, 14, 2016. URL: http://arxiv.org/abs/1603.04467.

2. B. Abolfathi, D. S. Aguado, G. Aguilar, C. Allende Prieto, A. Almeida, T. T. Ananna, F. Anders, S. F. Anderson, B. H. Andrews, B. Anguiano, and et al. “The Fourteenth Data Release of the Sloan Digital Sky Survey: First Spectroscopic Data from the Extended Baryon Oscillation Spectroscopic Survey and from the Second Phase of the Apache Point Observatory Galactic Evolution Experiment”. *The Astrophysical Journal Supplement Series* 235, 42, 2018, p. 42.

3. H. Aguettaz, E. J. Bekkers, and M. Defferrard. “ChebLieNet: Invariant spectral graph NNs turned equivariant by Riemannian geometry on Lie groups”. In: 2021. arXiv: 2111.12139. URL: https://arxiv.org/abs/2111.12139.

4. S. Alam, M. Ata, S. Bailey, F. Beutler, D. Bizyaev, J. A. Blazek, A. S. Bolton, J. R. Brownstein, A. Burden, C.-H. Chuang, J. Comparat, A. J. Cuesta, K. S. Dawson, D. J. Eisenstein, S. Escoffier, H. Gil-Marín, J. N. Grieb, N. Hand, S. Ho, K. Kinemuchi, D. Kirkby, F. Kitaura, E. Malanushenko, V. Malanushenko, C. Maraston, C. K. McBride, R. C. Nichol, M. D. Olmstead, D. Oravetz, N. Padmanabhan, N. Palanque-Delabrouille, K. Pan, M. Pellejero-Ibanez, W. J. Percival, P. Petitjean, F. Prada, A. M. Price-Whelan, B. A. Reid, S. A. Rodríguez-Torres, N. A. Roe, A. J. Ross, N. P. Ross, G. Rossi, J. A. Rubiño-Martín, S. Saito, S. Salazar-Albornoz, L. Samushia, A. G. Sánchez, S. Satpathy, D. J. Schlegel, D. P. Schneider, C. G. Scóccola, H.-J. Seo, E. S. Sheldon, A. Simmons, A. Slosar, M. A. Strauss, M. E. C. Swanson, D. Thomas, J. L. Tinker, R. Tojeiro, M. V. Magaña, J. A. Vazquez, L. Verde, D. A. Wake, Y. Wang, D. H. Weinberg, M. White, W. M. Wood-Vasey, C. Yèche, I. Zehavi, Z. Zhai, and G.-B. Zhao. “The clustering of galaxies in the completed SDSS-III Baryon Oscillation Spectroscopic Survey: cosmological analysis of the DR12 galaxy sample”. *Monthly Notices of the Royal Astronomical Society* 470, 2017, pp. 2617–2652.

5. S. Amsel, J. Berger, and R. H. Brandenberger. “Detecting cosmic strings in the CMB with the Canny algorithm”. *Journal of Cosmology and Astroparticle Physics* 4, 015, 2008, p. 015.
6. M. A. Aragon-Calvo. “Classifying the Large Scale Structure of the Universe with Deep Neural Networks”. *arXiv:1804.00816*, 2018.
7. M. Aubry, U. Schlickewei, and D. Cremers. “The wave kernel signature: A quantum mechanical approach to shape analysis”. In: *2011 IEEE international conference on computer vision workshops (ICCV workshops)*. IEEE. 2011, pp. 1626–1633.
8. J. Banjac, L. Donati, and M. Defferrard. “Learning to recover orientations from projections in single-particle cryo-EM”. In: 2021. arXiv: 2104.06237. URL: https://arxiv.org/abs/2104.06237.
9. P. Baqué, E. Remelli, F. Fleuret, and P. Fua. “Geodesic Convolutional Shape Optimization”. In: *International Conference on Machine Learning*. 2018.
10. M. Bartelmann. “Gravitational lensing”. *Classical and Quantum Gravity*, 2010.
11. M. Bartelmann and P. Schneider. “Weak gravitational lensing”. *Physics Reports* 340, 2001, pp. 291–472. DOI: 10.1016/S0370-1573(00)00082-X. eprint: astro-ph/9912508.
12. P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, et al. “Relational inductive biases, deep learning, and graph networks”. *arXiv:1806.01261*, 2018.
13. J. R. Baumgardner and P. O. Frederickson. “Icosahedral discretization of the two-sphere”. *SIAM Journal on Numerical Analysis*, 1985.
14. M. Bawa, T. Condie, and P. Ganesan. “LSH Forest: Self-Tuning Indexes for Similarity Search”. In: *International Conference on World Wide Web*. 2005, pp. 651–660.
15. M. Belkin and P. Niyogi. “Towards a Theoretical Foundation for Laplacian-based Manifold Methods”. *Journal of Computer and System Sciences* 74(8), 2008, pp. 1289–1308.
16. M. Belkin and P. Niyogi. “Convergence of Laplacian eigenmaps”. In: *Advances in Neural Information Processing Systems*. 2007.
17. M. Belkin and P. Niyogi. “Laplacian eigenmaps for dimensionality reduction and data representation”. *Neural computation* 15:6, 2003, pp. 1373–1396.

18. M. Belkin and P. Niyogi. “Towards a theoretical foundation for Laplacian-based manifold methods”. *Journal of Computer and System Sciences*, 2008.

19. A. I. Bobenko, J. M. Sullivan, P. Schröder, and G. Ziegler. *Discrete differential geometry*. Vol. 38. Springer, 2008.

20. W. Boomsma and J. Frellsen. “Spherical convolutions and their application in molecular modelling”. In: *Advances in Neural Information Processing Systems*. 2017.

21. W. Boomsma and J. Frellsen. “Spherical convolutions and their application in molecular modelling”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 3436–3446.

22. D. Boscaini, J. Masci, E. Rodolà, and M. Bronstein. “Learning shape correspondence with anisotropic convolutional neural networks”. In: *Advances in Neural Information Processing Systems*. 2016.

23. D. Boscaini, J. Masci, E. Rodolà, and M. Bronstein. “Learning shape correspondence with anisotropic convolutional neural networks”. In: *Advances in Neural Information Processing Systems*. 2016, pp. 3189–3197.

24. M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. “Geometric deep learning: going beyond euclidean data”. *IEEE Signal Processing Magazine*, 2017.

25. J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. “Spectral Networks and Deep Locally Connected Networks on Graphs”. *arXiv:1312.6203*, 2013.

26. J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun. “Spectral Networks and Locally Connected Networks on Graphs”. *arXiv:1312.6203*, 2013. URL: https://arxiv.org/abs/1312.6203.

27. T. Bui and C. Jones. “Finding Good Approximate Vertex and Edge Partitions is NP-hard”. *Information Processing Letters* 42(3), 1992, pp. 153–159.

28. A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, et al. “Shapenet: An information-rich 3d model repository”. *arXiv:1512.03012*, 2015.

29. C. Chang et al. “Dark Energy Survey Year 1 results: curved-sky weak lensing mass map”. *Monthly Notices of the Royal Astronomical Society* 475, 2018, pp. 3165–3190.

30. F. R. K. Chung. *Spectral Graph Theory*. Vol. 92. American Mathematical Society, 1997.

31. R. Ciuca, O. F. Hernández, and M. Wolman. “A Convolutional Neural Network For Cosmic String Detection in CMB Temperature Maps”. *arxiv:1708.08878*, 2017.

32. A. Coates and A. Ng. “Selecting Receptive Fields in Deep Networks”. In: *Neural Information Processing Systems (NIPS)*. 2011, pp. 2528–2536.

33. T. Cohen and M. Welling. “Group equivariant convolutional networks”. In: *International conference on machine learning*. PMLR. 2016, pp. 2990–2999.

34. T. S. Cohen, M. Geiger, J. Koehler, and M. Welling. “Spherical CNNs”. In: *International Conference on Learning Representations (ICLR)*. 2018. URL: https://arxiv.org/abs/1801.10130.

35. T. S. Cohen, M. Weiler, B. Kicanaoglu, and M. Welling. “Gauge Equivariant Convolutional Networks and the Icosahedral CNN”. In: *International Conference on Machine Learning (ICML)*. 2019. URL: http://arxiv.org/abs/1902.04615.

36. R. Coifman and S. Lafon. “Diffusion Maps”. *Applied and Computational Harmonic Analysis* 21(1), 2006, pp. 5–30.

37. R. R. Coifman and S. Lafon. “Diffusion maps”. *Applied and computational harmonic analysis* 21:1, 2006, pp. 5–30.

38. B. Coors, A. P. Condurache, and A. Geiger. “SphereNet: Learning Spherical Representations for Detection and Classification in Omnidirectional Images”. In: *European Conference on Computer Vision*. 2018.

39. B. Coors, A. P. Condurache, and A. Geiger. “SphereNet: Learning Spherical Representations for Detection and Classification in Omnidirectional Images”. In: *European Conference on Computer Vision*. 2018.

40. Dark Energy Survey Collaboration. “The Dark Energy Survey Data Release 1”. *arxiv: 1801.03181*, 2018.

41. M. Defferrard. “Generalized convolutions”. In: *In preparation*. 2022.

42. M. Defferrard, K. Benzi, P. Vandergheynst, and X. Bresson. “FMA: A Dataset for Music Analysis”. In: *18th International Society for Music Information Retrieval Conference (ISMIR)*. 2017. arXiv: 1612.01840. URL: https://arxiv.org/abs/1612.01840.

43. M. Defferrard, X. Bresson, and P. Vandergheynst. "Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering". In: *Advances in Neural Information Processing Systems (NIPS)*. 2016. arXiv: 1606.09375. URL: https://arxiv.org/abs/1606.09375.

44. M. Defferrard, L. Martin, R. Pena, and N. Perraudin. *PyGSP: Graph Signal Processing in Python*. DOI: 10.5281/zenodo.1003157. URL: https://github.com/epfl-lts2/pygsp/.

45. M. Defferrard, M. Milani, F. Gusset, and N. Perraudin. "DeepSphere: a graph-based spherical CNN". In: *International Conference on Learning Representations (ICLR)*. 2020. arXiv: 2012.15000. URL: https://arxiv.org/abs/2012.15000.

46. M. Defferrard, S. P. Mohanty, S. F. Carroll, and M. Salathé. "Learning to Recognize Musical Genre from Audio. Challenge Overview". In: *The 2018 Web Conference Companion*. ACM Press, 2018. ISBN: 9781450356404. DOI: 10.1145/3184558.3192310. arXiv: 1803.05337. URL: https://arxiv.org/abs/1803.05337.

47. M. Defferrard, R. Pena, and N. Perraudin. *PyUNLocBoX: Optimization by Proximal Splitting*. DOI: 10.5281/zenodo.1199081. URL: https://github.com/epfl-lts2/pyunlocbox/.

48. M. Defferrard, N. Perraudin, T. Kacprzak, and R. Sgier. "DeepSphere: towards an equivariant graph-based spherical CNN". In: *ICLR Workshop on Representation Learning on Graphs and Manifolds*. 2019. arXiv: 1904.05146. URL: https://arxiv.org/abs/1904.05146.

49. M. Desbrun, A. N. Hirani, M. Leok, and J. E. Marsden. "Discrete exterior calculus", 2005. arXiv: math/0508341. URL: https://arxiv.org/abs/math/0508341.

50. I. Dhillon, Y. Guan, and B. Kulis. "Weighted Graph Cuts Without Eigenvectors: A Multilevel Approach". *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 29(11), 2007, pp. 1944–1957.

51. J. R. Driscoll and D. M. Healy. "Computing Fourier Transforms and Convolutions on the 2-Sphere". *Adv. Appl. Math.*, 1994. URL: http://dx.doi.org/10.1006/aama.1994.1008.

52. D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams. "Convolutional networks on graphs for learning molecular fingerprints". In: *Advances in neural information processing systems*. 2015, pp. 2224–2232.

53. S. Ebli, M. Defferrard, and G. Spreemann. "Simplicial Neural Networks". In: *Topological Data Analysis and Beyond workshop at NeurIPS*. 2020. arXiv: 2010.03633. URL: https://arxiv.org/abs/2010.03633.

54. B. Eckmann. "Harmonische Funktionen und Randwertaufgaben in einem Komplex". *Commentarii Mathematici Helvetici* 17:1, 1944, pp. 240–255.

55. C. Esteves, C. Allen-Blanchette, A. Makadia, and K. Daniilidis. "Learning SO(3) Equivariant Representations with Spherical CNNs". *arXiv:1711.06721*, 2017.

56. C. Esteves, C. Allen-Blanchette, A. Makadia, and K. Daniilidis. "Learning SO(3) Equivariant Representations with Spherical CNNs". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018. URL: https://arxiv.org/abs/1711.06721.

57. J. Fluri, T. Kacprzak, A. Lucchi, A. Refregier, A. Amara, and T. Hofmann. "Cosmological constraints from noisy convergence maps through deep learning". *arxiv:1807.08732*, 2018.

58. F. Fouss, A. Pirotte, J.-M. Renders, and M. Saerens. "Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation". *IEEE Transactions on knowledge and data engineering* 19:3, 2007, pp. 355–369.

59. P. Frossard and R. Khasanova. "Graph-Based Classification of Omnidirectional Images". In: *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*. 2017, pp. 860–869.

60. R. Frucht. "Herstellung von Graphen mit vorgegebener abstrakter Gruppe". German. *Compositio Mathematica* 6, 1938, pp. 239–250. ISSN: 0010-437X.

61. M. Gavish, B. Nadler, and R. Coifman. "Multiscale Wavelets on Trees, Graphs and High Dimensional Data: Theory and Applications to Semi Supervised Learning". In: *International Conference on Machine Learning (ICML)*. 2010, pp. 367–374.

62. G. Ghiggi, M. Defferrard, W. Feng, Y. Y. Haddad, N. Bolón Brun, I. Lloréns Jover, and P. Dueben. "DeepSphere-Weather: Deep Learning on spherical unstructured grids for weather/climate applications". *In preparation*, 2022. URL: https://github.com/deepsphere/deepsphere-weather.

63. N. Gillet, A. Mesinger, B. Greig, A. Liu, and G. Ucci. “Deep learning from 21-cm images of the Cosmic Dawn”. *arxiv:1805.02699*, 2018.

64. K. Glomb, J. Rué Queralt, D. Pascucci, M. Defferrard, S. Tourbier, M. Carboni, M. Rubega, S. Vulliémoz, G. Plomp, and P. Hagmann. “Connectome spectral analysis to track EEG task dynamics on a subsecond scale”. *NeuroImage* 221, 2020. bioRxiv: 2020.06.22.164111, pp. 117–137. ISSN: 1053-8119. DOI: 10.1016/j.neuroimage.2020.117137.

65. X. Glorot and Y. Bengio. “Understanding the difficulty of training deep feedforward neural networks”. In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010, pp. 249–256.

66. F. Göbel and A. Jagers. “Random walks on graphs”. *Stochastic processes and their applications* 2:4, 1974, pp. 311–336.

67. K. M. Gorski, E. Hivon, A. Banday, B. D. Wandelt, F. K. Hansen, M. Reinecke, and M. Bartelmann. “HEALPix: a framework for high-resolution discretization and fast analysis of data distributed on the sphere”. *The Astrophysical Journal*, 2005.

68. K. M. Gorski, B. D. Wandelt, F. K. Hansen, E. Hivon, and A. J. Banday. “The healpix primer”. *arXiv preprint astro-ph/9905275*, 1999.

69. L. J. Grady and J. R. Polimeni. *Discrete calculus: Applied analysis on graphs for computational science*. Vol. 3. Springer, 2010.

70. K. Gregor and Y. LeCun. “Emergence of Complex-like Cells in a Temporal Product Network with Local Receptive Fields”. In: *arXiv:1006.0448*. 2010.

71. A. Gupta, J. M. Z. Matilla, D. Hsu, and Z. Haiman. “Non-Gaussian information from weak lensing data via deep learning”. *Physical Review D* 97:10, 103515, 2018, p. 103515.

72. F. Gusset, L. Vancauwenberghe, M. Allemann, J. Fluri, N. Perraudin, and M. Defferrard. *DeepSphere: learning on the sphere*. URL: https://github.com/deepsphere/.

73. W. L. Hamilton, R. Ying, and J. Leskovec. “Representation learning on graphs: Methods and applications”. *IEEE Data Engineering Bulletin, arXiv:1709.05584*, 2017.

74. D. Hammond, P. Vandergheynst, and R. Gribonval. “Wavelets on Graphs via Spectral Graph Theory”. *Applied and Computational Harmonic Analysis* 30(2), 2011, pp. 129–150.

75. Z. Harteveld, J. Southern, M. Defferrard, A. Loukas, P. Vandergheynst, M. M. Bronstein, and B. E. Correia. “Deep sharpening of topological features for de novo protein design”. In: *Machine Learning for Drug Discovery Workshop at ICLR*. 2022. URL: https://openreview.net/forum?id=H5g7W-6Oec.

76. S. Hassan, A. Liu, S. Kohn, J. E. Aguirre, P. La Plante, and A. Lidz. “Reionization Models Classifier using 21cm Map Deep Learning”. In: *IAU Symposium*. Ed. by V. Jelić and T. van der Hulst. Vol. 333. IAU Symposium. 2018, pp. 47–51.

77. S. He, S. Ravanbakhsh, and S. Ho. “Analysis of Cosmic Microwave Background with Deep Learning”, 2018.

78. M. Henaff, J. Bruna, and Y. LeCun. “Deep Convolutional Networks on Graph-Structured Data”. *arXiv:1506.05163*, 2015.

79. HI4PI Collaboration, N. Ben Bekhti, L. Flöer, R. Keller, J. Kerp, D. Lenz, B. Winkel, J. Bailin, M. R. Calabretta, L. Dedes, H. A. Ford, B. K. Gibson, U. Haud, S. Janowiecki, P. M. W. Kalberla, F. J. Lockman, N. M. McClure-Griffiths, T. Murphy, H. Nakanishi, D. J. Pisano, and L. Staveley-Smith. “HI4PI: A full-sky H I survey based on EBHIS and GASS”. *Astronomy & Astrophysics* 594, A116, 2016, A116.

80. H. Hildebrandt, M. Viola, C. Heymans, S. Joudaki, K. Kuijken, C. Blake, T. Erben, B. Joachimi, D. Klaes, L. Miller, C. B. Morrison, R. Nakajima, G. Verdoes Kleijn, A. Amon, A. Choi, G. Covone, J. T. A. de Jong, A. Dvornik, I. Fenech Conti, A. Grado, J. Harnois-Déraps, R. Herbonnet, H. Hoekstra, F. Köhlinger, J. McFarland, A. Mead, J. Merten, N. Napolitano, J. A. Peacock, M. Radovich, P. Schneider, P. Simon, E. A. Valentijn, J. L. van den Busch, E. van Uitert, and L. Van Waerbeke. “KiDS-450: cosmological parameter constraints from tomographic weak gravitational lensing”. *Monthly Notices of the Royal Astronomical Society* 465, 2017, pp. 1454–1498.

81. P. Hop, B. Allgood, and J. Yu. “Geometric Deep Learning Autonomously Learns Chemical Features That Outperform Those Engineered by Domain Experts”. *Molecular pharmaceutics*, 2018.

82. K. Hornik, M. Stinchcombe, and H. White. “Multilayer feedforward networks are universal approximators”. *Neural networks* 2:5, 1989, pp. 359–366.

83. C. Howlett, M. Manera, and W. J. Percival. “L-PICOLA: A parallel code for fast dark matter simulation”. *Astronomy and Computing* 12, 2015, pp. 109–126.

84. J. D. Hunter. “Matplotlib: A 2D graphics environment”. *Computing in Science & Engineering* 9:3, 2007, pp. 90–95. DOI: 10.1109/MCSE.2007.55.

85. K. T. Inoue, P. Cabella, and E. Komatsu. “Harmonic inpainting of the cosmic microwave background sky: Formulation and error estimate”. *Physical Review D* 77:12, 123539, 2008, p. 123539.

86. S. Ioffe and C. Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *International Conference on Machine Learning*. 2015.

87. M. Jaderberg, K. Simonyan, A. Zisserman, et al. “Spatial transformer networks”. In: *Advances in Neural Information Processing Systems*. 2015, pp. 2017–2025.

88. C. ” Jiang, J. Huang, K. Kashinath, Prabhat, P. Marcus, and M. Niessner. “Spherical CNNs on Unstructured Grids”. In: *International Conference on Learning Representations (ICLR)*. 2019. URL: https://arxiv.org/abs/1901.02039.

89. T. Joachims. “A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization”. *Carnegie Mellon University, Computer Science Technical Report* CMU-CS-96-118, 1996.

90. N. Kalchbrenner, E. Grefenstette, and P. Blunsom. “A Convolutional Neural Network for Modelling Sentences”. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vol. 1. 2014, pp. 655–665.

91. G. Karypis and V. Kumar. “A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs”. *SIAM Journal on Scientific Computing (SISC)* 20(1), 1998, pp. 359–392.

92. R. Khasanova and P. Frossard. “Graph-based classification of omnidirectional images”. In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017. URL: https://arxiv.org/abs/1707.08301.

93. D. Kingma and J. Ba. “Adam: A Method for Stochastic Optimization”. *arXiv:1412.6980*, 2014.

94. D. P. Kingma and J. Ba. “Adam: A method for stochastic optimization”. *arXiv:1412.6980*, 2014.

95. D. J. Klein and M. Randić. “Resistance distance”. *Journal of mathematical chemistry* 12:1, 1993, pp. 81–95.

96. E. Komatsu, K. M. Smith, J. Dunkley, C. L. Bennett, B. Gold, G. Hinshaw, N. Jarosik, D. Larson, M. R. Nolta, L. Page, D. N. Spergel, M. Halpern, R. S. Hill, A. Kogut, M. Limon, S. S. Meyer, N. Odegard, G. S. Tucker, J. L. Weiland, E. Wollack, and E. L. Wright. "Seven-year Wilkinson Microwave Anisotropy Probe (WMAP) Observations: Cosmological Interpretation". *The Astrophysical Journal Supplement Series* 192, 18, 2011, p. 18.

97. R. Kondor, Z. Lin, and S. Trivedi. "Clebsch-Gordan Nets: a Fully Fourier Space Spherical Convolutional Neural Network". *arXiv:1806.09231*, 2018.

98. R. Kondor and S. Trivedi. "On the generalization of equivariance and convolution in neural networks to the action of compact groups". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 2747–2755.

99. R. I. Kondor and J. Lafferty. "Diffusion kernels on graphs and other discrete structures". In: *Proceedings of the 19th international conference on machine learning*. Vol. 2002. 2002, pp. 315–322.

100. N. Krachmalnicoff and M. Tomasi. "Convolutional Neural Networks on the HEALPix sphere: a pixel-based algorithm and its application to CMB data analysis". *arXiv preprint arXiv:1902.04083*, 2019.

101. D. Krackhardt. "Assessing the Political Landscape: Structure, Cognition, and Power in Organizations". *Administrative Science Quarterly* 35:2, 1990, pp. 342–369. ISSN: 00018392. URL: http://www.jstor.org/stable/2393394.

102. S. I. Ktena, S. Parisot, E. Ferrante, M. Rajchl, M. Lee, B. Glocker, and D. Rueckert. "Metric learning with spectral graph convolutions on brain connectivity networks". *NeuroImage* 169, 2018, pp. 431–442.

103. T. Kuhn. *The structure of scientific revolutions*. University of Chicago Press, 1962.

104. L. Le Magoarou, R. Gribonval, and N. Tremblay. "approximate fast graph Fourier transforms via multi-layer sparse approximations". *IEEE transactions on Signal and Information Processing over Networks* 4:2, 2018, pp. 407–420.

105. Y. LeCun, Y. Bengio, and G. Hinton. "Deep Learning". *Nature* 521(7553), 2015, pp. 436–444.

106. Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-Based Learning Applied to Document Recognition". In: *Proceedings of the IEEE, 86(11)*. 1998, pp. 2278–2324.

107. Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition". *Proceedings of the IEEE* 86:11, 1998, pp. 2278–2324.

108. Y. Li, R. Yu, C. Shahabi, and Y. Liu. "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting", 2018.

109. Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. "Gated Graph Sequence Neural Networks", 17, 2015. URL: http://arxiv.org/abs/1511.05493.

110. Y. Li, D. Tarlow, M. Brockschmidt, and R. Zemel. "Gated graph sequence neural networks". In: *International Conference on Learning Representation*. 2016.

111. M. Lin, Q. Chen, and S. Yan. "Network in network". *arXiv:1312.4400*, 2013.

112. J. Long, E. Shelhamer, and T. Darrell. "Fully convolutional networks for semantic segmentation". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 3431–3440.

113. L. Lucie-Smith, H. V. Peiris, A. Pontzen, and M. Lochner. "Machine learning cosmological structure formation". *Monthly Notices of the Royal Astronomical Society* 479, 2018, pp. 3405–3414.

114. U. V. Luxburg. "A Tutorial on Spectral Clustering". *Statistics and Computing* 17(4), 2007, pp. 395–416.

115. S. Mallat. *A Wavelet Tour of Signal Processing*. Academic press, 1999.

116. S. Mallat. "Group invariant scattering". *Communications on Pure and Applied Mathematics* 65:10, 2012, pp. 1331–1398.

117. Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

118. Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever,

Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

119. J. Masci, D. Boscaini, M. Bronstein, and P. Vandergheynst. "Geodesic convolutional neural networks on riemannian manifolds". In: *Proceedings of the IEEE International Conference on Computer Vision Workshops*. 2015, pp. 37–45.

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

121. T. Mikolov, K. Chen, G. Corrado, and J. Dean. "Estimation of Word Representations in Vector Space". In: *International Conference on Learning Representations*. 2013.

122. M. J. Mohlenkamp. "A fast transform for spherical harmonics". *Journal of Fourier analysis and applications* 5:2-3, 1999, pp. 159–184.

123. F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein. "Geometric deep learning on graphs and manifolds using mixture model CNNs". In: *Proc. CVPR*. Vol. 1. 2. 2017, p. 3.

124. F. Monti, M. Bronstein, and X. Bresson. "Geometric matrix completion with recurrent multi-graph neural networks". In: *Advances in Neural Information Processing Systems*. 2017, pp. 3697–3707.

125. M. Mudigonda, S. Kim, A. Mahesh, S. Kahou, K. Kashinath, D. Williams, V. Michalski, T. O'Brien, and M. Prabhat. "Segmenting and tracking extreme climate events using neural networks". In: *Deep Learning for Physical Sciences (DLPS) Workshop, held with NIPS Conference*. 2017. URL: https://dl4physicalsciences.github.io/files/nips_dlps_2017_20.pdf.

126. L. Page, S. Brin, R. Motwani, and T. Winograd. *The PageRank citation ranking: Bringing order to the web*. Technical report. Stanford InfoLab, 1999.

127. S. Parisot, S. I. Ktena, E. Ferrante, M. Lee, R. G. Moreno, B. Glocker, and D. Rueckert. "Spectral graph convolutions for population-based disease prediction". In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2017, pp. 177–185.

128. B. Pasdeloup, R. Alami, V. Gripon, and M. Rabbat. “Toward an Uncertainty Principle for Weighted Graphs”. In: *Signal Processing Conference (EUSIPCO)*. 2015, pp. 1496–1500.

129. A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

130. K. Patton, J. Blazek, K. Honscheid, E. Huff, P. Melchior, A. J. Ross, and E. Suchyta. “Cosmological constraints from the convergence 1-point probability distribution”. *Monthly Notices of the Royal Astronomical Society* 472, 2017, pp. 439–446.

131. N. Perraudin, B. Ricaud, D. Shuman, and P. Vandergheynst. “Global and Local Uncertainty Principles for Signals on Graphs”. *arXiv:1603.03030*, 2016.

132. N. Perraudin, M. Defferrard, T. Kacprzak, and R. Sgier. “DeepSphere: Efficient spherical Convolutional Neural Network with HEALPix sampling for cosmological applications”. *Astronomy and Computing* 27, 2019, pp. 130–146. ISSN: 2213-1337. DOI: 10.1016/j.ascom.2019.03.004. arXiv: 1810.12186. URL: https://arxiv.org/abs/1810.12186.

133. N. Perraudin and P. Vandergheynst. “Stationary Signal Processing on Graphs.” *IEEE Trans. Signal Processing* 65:13, 2017, pp. 3462–3477.

134. F. Peter and H. Weyl. “Die Vollständigkeit der primitiven Darstellungen einer geschlossenen kontinuierlichen Gruppe”. *Mathematische Annalen* 97:1, 1927, pp. 737–755.

135. Planck Collaboration. “Planck 2015 results. I. Overview of products and scientific results”. *Astronomy & Astrophysics*, 2016.

136. Planck Collaboration. “Planck 2015 results. XIII. Cosmological parameters”. *Astronomy & Astrophysics* 594, A13, 2016, A13.

137. X. Qi, R. Liao, J. Jia, S. Fidler, and R. Urtasun. “3D Graph Neural Networks for RGBD Semantic Segmentation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 5199–5208.

138. I. Ram, M. Elad, and I. Cohen. “Generalized Tree-based Wavelet Transform”. *IEEE Transactions on Signal Processing,* 59(9), 2011, pp. 4199–4209.

139. S. Ravanbakhsh, J. Oliva, S. Fromenteau, L. C. Price, S. Ho, J. Schneider, and B. Póczos. “Estimating Cosmological Parameters from the Dark Matter Distribution”. In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. ICML’16. JMLR.org, New York, NY, USA, 2016, pp. 2407–2416.

140. M. Reinecke and D. S. Seljebotn. “Libsharp–spherical harmonic transforms revisited”. *Astronomy & Astrophysics* 554, 2013, A112.

141. D. Ribli, B. Ármin Pataki, and I. Csabai. “Learning from deep learning: better cosmological parameter inference from weak lensing maps”. *arxiv:1806.05995*, 2018.

142. V. Rokhlin and M. Tygert. “Fast algorithms for spherical harmonic expansions”. *SIAM Journal on Scientific Computing* 27:6, 2006, pp. 1903–1928.

143. D. Ron, I. Safro, and A. Brandt. “Relaxation-based Coarsening and Multiscale Graph Organization”. *SIAM Iournal on Multiscale Modeling and Simulation* 9, 2011, pp. 407–423.

144. C. Ronchi, R. Iacono, and P. S. Paolucci. “The “cubed sphere”: a new method for the solution of partial differential equations in spherical geometry”. *Journal of Computational Physics* 124:1, 1996, pp. 93–114.

145. M. Santos, P. Bull, D. Alonso, S. Camera, P. Ferreira, G. Bernardi, R. Maartens, M. Viel, F. Villaescusa-Navarro, F. B. Abdalla, M. Jarvis, R. B. Metcalf, A. Pourtsidou, and L. Wolz. “Cosmology from a SKA HI intensity mapping survey”. *Advancing Astrophysics with the Square Kilometre Array (AASKA14)*, 19, 2015, p. 19.

146. M. Savva, F. Yu, H. Su, A. Kanezaki, T. Furuya, R. Ohbuchi, Z. Zhou, R. Yu, S. Bai, X. Bai, et al. “Large-scale 3D shape retrieval from ShapeNet Core55: SHREC’17 track”. In: *Eurographics Workshop on 3D Object Retrieval*. 2017.

147. F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. “The Graph Neural Network Model”. *IEEE Transactions on Neural Networks* 20:1, 2009, pp. 61–80. ISSN: 1045-9227. DOI: 10.1109/TNN.2008.2005605.

148. A. Scheck, S. Rosset, M. Defferrard, A. Loukas, J. Bonet, P. Vandergheynst, and B. E. Correia. “RosettaSurf—A surface-centric computational design approach”. *PLOS Computational Biology* 18:3, 2022. bioRxiv: 2021.06.16.448645, pp. 1–23. DOI: 10.1371/journal.pcbi.1009178.

149. J. Schmelzle, A. Lucchi, T. Kacprzak, A. Amara, R. Sgier, A. Réfrégier, and T. Hofmann. "Cosmological model discrimination with Deep Learning". *arxiv:1707.05167*, 2017.

150. Y. Seo, M. Defferrard, P. Vandergheynst, and X. Bresson. "Structured Sequence Modeling with Graph Convolutional Recurrent Networks". In: *International Conference on Neural Information Processing (ICONIP)*. 2017. arXiv: 1612.07659. URL: https://arxiv.org/abs/1612.07659.

151. R. Sgier, A. Réfrégier, A. Amara, and A. Nicola. "Fast Generation of Covariance Matrices for Weak Lensing". *arxiv:1801.05745*, 2018.

152. J. Shi and J. Malik. "Normalized Cuts and Image Segmentation". *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* 22(8), 2000, pp. 888–905.

153. D. Shuman, S. Narang, P. Frossard, A. Ortega, and P. Vandergheynst. "The Emerging Field of Signal Processing on Graphs: Extending High-Dimensional Data Analysis to Networks and other Irregular Domains". *IEEE Signal Processing Magazine* 30(3), 2013, pp. 83–98.

154. D. Shuman, M. Faraji, and P. Vandergheynst. "A Multiscale Pyramid Transform for Graph Signals". *IEEE Transactions on Signal Processing* 64(8), 2016, pp. 2119–2134.

155. D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst. "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains". *IEEE signal processing magazine* 30:3, 2013, pp. 83–98. arXiv: 1211.0053.

156. D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst. "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains". *IEEE Signal Processing Magazine* 30:3, 2013, pp. 83–98.

157. J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. "Striving for simplicity: The all convolutional net". *arXiv preprint arXiv:1412.6806*, 2014.

158. S. Staggs, J. Dunkley, and L. Page. "Recent discoveries from the cosmic microwave background: a review of recent progress". *Reports on Progress in Physics* 81:4, 044901, 2018, p. 044901.

159. Y.-C. Su and K. Grauman. "Learning spherical convolution for fast features from 360 imagery". In: *Advances in Neural Information Processing Systems*. 2017.

160. Y.-C. Su and K. Grauman. “Learning spherical convolution for fast features from 360 imagery”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 529–539.

161. J. Sun, M. Ovsjanikov, and L. Guibas. “A concise and provably informative multi-scale signature based on heat diffusion”. In: *Computer graphics forum*. Vol. 28. 5. Wiley Online Library. 2009, pp. 1383–1392.

162. A. Susnjara, N. Perraudin, D. Kressner, and P. Vandergheynst. “Accelerated Filtering on Graphs using Lanczos Method”. *preprint arXiv:1509.04537*, 2015.

163. M. A. Troxel et al. “Dark Energy Survey Year 1 Results: Cosmological Constraints from Cosmic Shear”. *arxiv:1708.01538*, 2017.

164. M. Tsitsvero and S. Barbarossa. “On the Degrees of Freedom of Signals on Graphs”. In: *Signal Processing Conference (EUSIPCO)*. 2015, pp. 1506–1510.

165. P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. Jarrod Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. Carey, İ. Polat, Y. Feng, E. W. Moore, J. Vand erPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and S. 1. 0. Contributors. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. *Nature Methods*, 2020. DOI: https://doi.org/10.1038/s41592-019-0686-2.

166. S. v. d. Walt, S. C. Colbert, and G. Varoquaux. “The NumPy array: a structure for efficient numerical computation”. *Computing in Science & Engineering* 13:2, 2011, pp. 22–30.

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

168. M. Welling. “Do we still need models or just more data and compute?”, 2019. URL: https://staff.fnwi.uva.nl/m.welling/wp-content/uploads/Model-versus-Data-AI-1.pdf.

169. A. G. Wilson and P. Izmailov. “Bayesian deep learning and a probabilistic perspective of generalization”. *arXiv preprint arXiv:2002.08791*, 2020.

170. J. Xiao, K. A. Ehinger, A. Oliva, and A. Torralba. “Recognizing scene viewpoint using panoramic place representation”. In: *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE. 2012, pp. 2695–2702.

171. Y. Zhang, S. Song, P. Tan, and J. Xiao. “Panocontext: A whole-room 3d context model for panoramic scene understanding”. In: *European Conference on Computer Vision*. Springer. 2014, pp. 668–686.

172. A. Zonca, L. Singer, D. Lenz, M. Reinecke, C. Rosset, E. Hivon, and K. Gorski. “healpy: equal area pixelization and spherical harmonics transforms for data on the sphere in Python”. *Journal of Open Source Software* 4:35, 2019, p. 1298. DOI: 10.21105/joss.01298. URL: https://doi.org/10.21105/joss.01298.

Biography

I am a Machine Learning researcher, currently pursuing a PhD at the École Polytechnique Fédérale de Lausanne (EPFL) with Prof. Pierre Vandergheynst. My main research interest is the modeling, analysis, and understanding of structured data. Data structured by networks, such as brain activity supported by neural connections, or manifolds, such as the temperature and wind fields on the Earth. To this end I am developing Deep Learning to leverage that structure, often modeled as a graph. In my research, I draw from theoretical insights and practical needs to develop principled methods—and strive for impactful applications by co-leading interdisciplinary research efforts. Besides research, I teach Data Science and Machine Learning (with graphs).

Michaël Defferrard

"I make machines learn; better by leveraging structure."

Strengths

- Experienced with Machine/Deep Learning on graphs/networks/manifolds/meshes/complexes.
- Skilled in software development (ML frameworks, scientific Python, package maintenance).
- Published at top ML and domain-specific venues and contributed to open-source software.

Education

2015–2021 **PhD Machine Learning**, *École Polytechnique Fédérale de Lausanne (EPFL)*
- Thesis: Leveraging topology, geometry, and symmetries for efficient Machine Learning.
- Adviser: Prof. Pierre Vandergheynst.
- Examiners: Martin Jaggi (EPFL), Max Welling (UvA, MSR), Yann LeCun (NYU, FAIR).

2012–2015 **MSc Electrical and Electronic Engineering**, *EPFL*, *GPA 96%*
- Thesis: Structured auto-encoder with application to music genre recognition.
- Minor in Computational Neuroscience.
- Courses and projects on signal processing, data analysis, machine learning.

2009–2012 **BASc Electrical Engineering**, *École d'Ingénieurs de Fribourg (EIA-FR)*, *GPA 98%*
- Courses and projects on electronic design, analog and digital circuits, embedded systems.
- Exchange year at the Hochschule München.

2005–2009 **Federal VET Diploma in Electronics**, *EPAI*, Fribourg CH, *GPA 96%*

Experience

present **Research Assistant**, *École Polytechnique Fédérale de Lausanne (EPFL)*
2014-02 I researched on Machine Learning and data structured by graphs and manifolds. I published papers in top-tier venues, co-led interdisciplinary research teams, supervised students, gave talks, taught courses, developed software. My work pioneered graph ML research and proved useful in tackling important real-world problems.

2015-08 **Software Engineer**, *Infoteam*, Givisiez CH
2011-08 Part-time job in the Energy R&D team. I ported a core product of the company, a control-command tool for energy distribution and production, to embedded systems. My work enabled the company to close its largest contract to date.

2012-08 **Research Intern**, *Lawrence Berkeley National Laboratory (LBNL)*
2012-05 I characterized the performance of a new particle detector for the ATLAS experiment at the CERN's Large Hadron Collider (LHC).

2011-03 **Electronics Specialist**, *Meggitt*, Fribourg CH
2005-08 Apprenticeship and part-time job. Production, test, quality assurance, repair, certification and development of sensing systems for the aerospace and energy markets.

michael.defferrard@epfl.ch • deff.ch •

Awards

2021 Nominated for the EPFL Doctorate award. Given to the best 2 of ~ 420 theses.

2020 Spotlight talk for [7] at ICLR.

2016, 2017 Nominated by EPFL for a Google PhD Fellowship.

2014 Selected and funded for the Silicon Valley Startup Camp.

2012 Award from Phonak Communications for an excellent BASc thesis.

2009 Awards from the UPCF and the SFP for the highest GPA.

Scientific output

Publications **10+ papers, 5000+ citations, h-index of 10.** Published at top ML conferences (NeurIPS, ICLR) as well as domain-specific journals (NeuroImage, Astronomy & Computing) and conferences (The Web Conference, ISMIR). List below, on Google Scholar, and at `deff.ch` with code, data, reviews.

Software Maintainer of 3 Python packages. Contributor to the Python scientific stack (NumPy, SciPy, Matplotlib, Jupyter, etc.). List below, with more open-source contributions (paper implementations, teaching materials) at `github.com/mdeff`.

Talks I gave 20+ talks. List with slides and some videos at `deff.ch`.

Teaching I co-taught 8 courses around Machine Learning, Networks, and Data Science, in various roles (TAing, lecturing, teaching team & student management, curriculum design) and forms (university class, workshop, summer school). List with roles and resources at `deff.ch`.

Leadership

Supervision I supervised 20+ students (MSc theses, semester projects, internships). List of students with project title, co-supervisors, and outputs at `deff.ch`.

Collaborations I tackle real-world problems by assembling and leading interdisciplinary collaborations. So far in Neuroscience, Cosmology, Biology, Geoscience.

Organization
- Machine Learning for Earth, Seminar, 2019–now.
- Musical Genre Recognition Challenge, The Web Conference (WWW), 2018.
- Open Science in Practice, Summer School, EPFL, 2017.
- Deep Learning on Irregular Domains, Workshop, BMVC, 2017.

Extra I serve as the president of a band of 40 musicians and a leadership member at a firefighting brigade of 80.

Miscellaneous

Open Open science, open source, open data, and reproducibility are values I advocate for and adhere to in my research.

Extra Brass band musician, militia firefighting officer, runner, computing enthusiast.

Publications

scholar.google.com/citations?user=Ztj2-gUAAAAJ

[1] M. Defferrard. "Generalized convolutions". *In preparation*. 2022.

[2] G. Ghiggi, M. Defferrard, W. Feng, Y. Y. Haddad, N. Bolón Brun, I. Lloréns Jover, P. Dueben. "DeepSphere-Weather: Deep Learning on spherical unstructured grids for weather/climate applications". *In preparation* (2022).

[3] Z. Harteveld, J. Southern, M. Defferrard, A. Loukas, P. Vandergheynst, M. M. Bronstein, B. E. Correia. "Deep sharpening of topological features for de novo protein design". *Machine Learning for Drug Discovery Workshop at ICLR*. 2022.

[4] A. Scheck, S. Rosset, M. Defferrard, A. Loukas, J. Bonet, P. Vandergheynst, B. E. Correia. "RosettaSurf—A surface-centric computational design approach". *PLOS Computational Biology* 18.3 (Mar. 2022). bioRxiv: 2021.06.16.448645, pp. 1–23.

[5] H. Aguettaz, E. J. Bekkers, M. Defferrard. "ChebLieNet: Invariant spectral graph NNs turned equivariant by Riemannian geometry on Lie groups". 2021. arXiv: 2111.12139.

[6] J. Banjac, L. Donati, M. Defferrard. "Learning to recover orientations from projections in single-particle cryo-EM". 2021. arXiv: 2104.06237.

[7] M. Defferrard, M. Milani, F. Gusset, N. Perraudin. "DeepSphere: a graph-based spherical CNN". *International Conference on Learning Representations (ICLR)*. 2020. arXiv: 2012.15000.

[8] S. Ebli, M. Defferrard, G. Spreemann. "Simplicial Neural Networks". *Topological Data Analysis and Beyond workshop at NeurIPS*. 2020. arXiv: 2010.03633.

[9] K. Glomb, J. Rué Queralt, D. Pascucci, M. Defferrard, S. Tourbier, M. Carboni, M. Rubega, S. Vulliémoz, G. Plomp, P. Hagmann. "Connectome spectral analysis to track EEG task dynamics on a subsecond scale". *NeuroImage* 221 (2020). bioRxiv: 2020.06.22.164111, pp. 117–137.

[10] M. Defferrard, N. Perraudin, T. Kacprzak, R. Sgier. "DeepSphere: towards an equivariant graph-based spherical CNN". *ICLR Workshop on Representation Learning on Graphs and Manifolds*. 2019. arXiv: 1904.05146.

[11] N. Perraudin, M. Defferrard, T. Kacprzak, R. Sgier. "DeepSphere: Efficient spherical Convolutional Neural Network with HEALPix sampling for cosmological applications". *Astronomy and Computing* 27 (Apr. 2019), pp. 130–146. arXiv: 1810.12186.

[12] M. Defferrard, S. P. Mohanty, S. F. Carroll, M. Salathé. "Learning to Recognize Musical Genre from Audio. Challenge Overview". *The 2018 Web Conference Companion*. ACM Press, 2018. arXiv: 1803.05337.

[13] M. Defferrard, K. Benzi, P. Vandergheynst, X. Bresson. "FMA: A Dataset for Music Analysis". *18th International Society for Music Information Retrieval Conference (ISMIR)*. 2017. arXiv: 1612.01840.

[14] Y. Seo, M. Defferrard, P. Vandergheynst, X. Bresson. "Structured Sequence Modeling with Graph Convolutional Recurrent Networks". *International Conference on Neural Information Processing (ICONIP)*. 2017. arXiv: 1612.07659.

[15] M. Defferrard, X. Bresson, P. Vandergheynst. "Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering". *Advances in Neural Information Processing Systems (NIPS)*. 2016. arXiv: 1606.09375.

Software

github.com/mdeff

[1] M. Defferrard, L. Martin, R. Pena, N. Perraudin. *PyGSP: Graph Signal Processing in Python*. URL: https://github.com/epfl-lts2/pygsp/.

[2] M. Defferrard, R. Pena, N. Perraudin. *PyUNLocBoX: Optimization by Proximal Splitting*. URL: https://github.com/epfl-lts2/pyunlocbox/.

[3] F. Gusset, L. Vancauwenberghe, M. Allemann, J. Fluri, N. Perraudin, M. Defferrard. *DeepSphere: learning on the sphere*. URL: https://github.com/deepsphere/.