LEVERAGING TOPOLOGICAL STRUCTURE OF DATA FOR APPLICATIONS OF DEEP LEARNING

By

Sarah McGuire

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

Computational Mathematics, Science & Engineering—Doctor of Philosophy

2024

ABSTRACT

Topological data analysis and deep learning are fields which have widely growing interest and rapid research developments, each in their own right. However, at the intersection of these fields, there is an opportunity to leverage the topological structure of data, incorporating the additional information into deep learning algorithms and methods applied to deep learning tasks. The design of deep learning architectures for various tasks on the domain of topological objects has seen quick progress, fueled by the desire to model higher-order interactions that are often naturally occurring in data.

The first focus area of this dissertation is an extension of pooling layers to simplicial complex input data. For deep learning problems on graph-structured data, pooling layers are important for down sampling, reducing computational cost, and to minimize overfitting in the model. We define a pooling layer, NervePool, for data structured as simplicial complexes, which are generalizations of graphs that include higher-dimensional simplices beyond vertices and edges; this structure allows for greater flexibility in modeling higher-order relationships. The proposed simplicial coarsening scheme is built upon partitions of vertices, which allow us to generate hierarchical representations of simplicial complexes, collapsing information in a learned fashion. NervePool builds on the learned vertex cluster assignments and extends to coarsening of higher dimensional simplices in a deterministic fashion. While in practice, the pooling operations are computed via a series of matrix operations, the topological motivation is a set-theoretic construction based on unions of stars of simplices and the nerve complex.

The second focus of this dissertation is another input data type with topological structure, the Euler Characteristic Transform (ECT). The ECT provides a summary of the topological shape of data which is both simple to define and simple to compute for many different input data types, including images, graphs, and embedded simplicial complexes. In contrast to alternative directional transform methods in topological data analysis, the ECT is easier to compute and represent in a format well-suited for machine learning tasks. To leverage the inherent structure of ECT data on a cylinder for our input data types, we employ a particular choice of convolutional neural

network (CNN) architecture for the classification of ECT data. We prove that our ECT-CNN pipeline produces equivariant representations of input data, which allows for the use of un-aligned input data. We apply the ECT-CNN to two different leaf shape datasets and compare the model performance against traditionally used methods which require data to be prealigned, and in doing so we exhibit its efficacy in shape classification tasks.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my family for their support and patience while I have been in graduate school. Mom and dad, thank you for always supporting me. Knowing that I have your full confidence in everything I do brings me so much reassurance. Thank you to my husband, Sean. You have supported me, encouraged me, and cheered for me, believing in me even on the days when I did not believe in myself. Thank you for always encouraging me to focus on work when I needed a push, and for reminding me to enjoy life and take breaks too. To my siblings Elizabeth and Ian, thank you for always being a phone call away and for your willingness to make in-person visits happen, despite us living in three different states. To my grandparents, Tom and Sally, thank you for your support and for always being excited to hear about what I'm working on. To my in-laws, Mike, Liesl, Lindsey, Eric, and Nicholas, thank you for always including me and making me feel like family (long before I was). You made Michigan feel like home and have been some of my biggest cheerleaders through my years of graduate school.

Thank you to all of my current and former committee members: Dr. Matthew Hirn (my former co-advisor), Dr. Teena Gerhardt, Dr. Dan Chitwood, and Dr. Vishnu Boddeti. I greatly appreciate the privilege of learning from each of you. I would like to thank my first mathematics mentor, Dr. Dave Damiano, for introducing me to research as an undergraduate student and for encouraging me to pursue a PhD; I would not have even considered this path without your guidance. Thank you also to Dr. Tegan Emerson and Dr. Henry Kvinge for your ongoing mentorship, beginning during my time as an intern at PNNL.

I am thankful to all of the MunchLab members, former and current, for their feedback on talks, helpful discussions, and overall camaraderie. You have been such a great group of people to work with and learn from the past five years. Finally, I would especially like to thank my advisor, Dr. Liz Munch, for being such a strong role model to me, both professionally and personally. Your integrity, work ethic, and compassion for all those you encounter has inspired me as a researcher and human being in general.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Topological Data Analysis (TDA) describes a suite of tools for quantifying shape in data, encompassing a wide range of methods derived from algebraic topology and computational geometry and their use within the context of machine learning [30, 100]. The primary value of such tools is that they can achieve robust, quantitative measures of shape that summarize the topological features of data. Meanwhile, deep learning encompasses a vast class of machine learning algorithms in which typically large neural networks are used for representation learning tasks. These algorithms combine many simple affine (i.e. linear) functions with pointwise nonlinear functions (i.e. activation functions) and pooling operations to approximate complex functions representing a specific task. As deep learning algorithms are widely used in contexts ranging from healthcare and computer vision to insurance and autonomous driving vehicles, it is increasingly important to understand the underlying mathematics of these algorithms. While the empirical results of these algorithms often show great success, there is much left unexplained in terms of mathematical justification. Additionally, unintentional bias in models can have very real and harmful effects when these algorithms are used in more widespread societal contexts. Understanding and interpreting how deep learning models function in relation to- and within- the context of other mathematical fields is important to the overall comprehension of mathematical underpinnings of deep learning algorithms. Despite the opportunity for generalizations of methods in deep learning (particularly those of graph representation learning and geometric deep learning) to adapt them for use in TDA, these fields have remained largely independent for much of their existence. More recently, we have seen the emergence of research at the intersection of these two fields, an effort commonly referred to as topological deep learning (TDL) [39, 99, 16].

In [42], the authors combine deep learning with TDA by proposing a technique to learn task-optimal representations of input topological signatures to ensure that the decision of machine-learning-compatible representation is not agnostic to the task. There have been rapid advancements within this scope, including the emergence of many varying proposed architectures for different

types of topological input (we refer to [72] for a survey of message passing-style topological neural networks). Additionally, the position paper [71] effectively summarizes existing work in this space along with open problems and unresolved lines of inquiry. For data supported on different topological domains, there are necessary adjustments and generalizations from traditional deep learning methods defined on structures such as grids and graphs. In this dissertation, we address necessary considerations for two different types of topological input data for deep learning models: simplicial complexes and directional transform data. Each of these topological data types have associated structures, which we exploit in different ways to support their use within deep learning models. Specifically, we contribute to the rapidly expanding body of topological deep learning literature by

1. developing and implementing a pooling method for neural networks on the domain of simplicial complexes, and

2. proposing a Convolutional Neural Network (CNN) framework for analysis of directional transform data.

In the first part of this work, we introduce a pooling layer for simplicial complexes, NervePool, which extends existing methods for graph pooling to be defined on the full simplicial complex using standard tools from combinatorial topology. While in practice, the pooling operations are computed via a series of matrix operations, NervePool has a compatible topological formulation.

Given an input simplicial complex and learned partition of the vertices (i.e. a specified cover on the vertex set), the NervePool method is defined by both by its topologically motivated framework and a compatible matrix representation. For exposition purposes, we assume that the initial clusters can form a soft partition of the vertex set, meaning every vertex is assigned to at least one cluster but vertices can have membership in more than one cluster. However, theoretical proof of some properties requires restriction to the setting of a hard partition of the vertex set. In particular, under the assumption of a hard partition of vertices, we prove that NervePool maintains invariance properties of pooling layers necessary for simplicial complex pooling in neural networks, as well

as additional properties to maintain simplicial complex structure after pooling. The initial vertex clusters give us a natural way to coarsen the underlying graph (1-skeleton) of the input simplicial complex, where clusters of vertices in the original complex are represented by meta-vertices in the pooled complex. We use the nerve of the extended cover of the complex to construct a new, pooled simplicial complex, which in practice is achieved though matrix multiplication of cluster assignment matrices with boundary matrices of each dimension.

In this work, we show that there is a choice of input cover on the vertices such that NervePool returns the same simplicial complex (up to re-weighting) and that when used in the context of a simplicial neural network with hard vertex clusters, it is a simplex-permutation invariant layer. Additionally, we prove the equivalence of the nerve/cover topological interpretation and matrix implementation using boundary matrices for the setting restricted to hard vertex partitions. This pooling layer has potential applications in a range of deep learning tasks such as classification and link prediction, in settings where the input data can be naturally modeled as a simplicial complex, helping to mitigate the additional computation cost of including higher dimensional simplices.

The second part of this work leverages topological structure of the Euler Characteristic Transform (ECT) in the context of deep learning. The ECT is a simple to define and simple to compute topological representation which descriptively represents the topological shape of data. In contrast to alternative options defined in the literature, the ECT is easier to compute, as well as being amenable to machine learning input requirements in a format well-suited for machine learning tasks. In this work, we propose to apply a particular choice of CNN architecture for classification of directional transform data, leveraging the inherent structure of the data on a cylinder. We prove that this ECT-CNN pipeline is equivariant to rotations of the input simplicial complex, which is a necessary property for its use on un-aligned data.

Using our proposed ECT-CNN pipeline, we apply the method for classification tasks of multiple leaf shape datasets. Measuring leaf shape is paramount to discovering and understanding phylogenetic and evolutionary relationships of plants. Traditional methods, however, often rely on direct measurements and limited statistical methods to quantify differences in leaf shape. In this example

application, we harness the effectiveness of ECT representations and the power of convolutional neural network models to quantify the naturally occurring widespread variation in leaf morphology.

This dissertation is structured in the following way: Chapter 2 describes relevant background in topological data analysis and deep learning. Chapter 3 proposes a pooling layer for simplicial complex neural networks (note that this chapter is largely duplicated from our preprint [64]). Chapter 4 proposes the use of convolutional neural networks for classification of directional transform data (specifically the ECT). Chapter 5 describes applications of this method to biologically relevant leaf-shape datasets. Chapter 6 outlines overall conclusions and directions for future work.

CHAPTER 2

BACKGROUND

This chapter gives background context for some important concepts in topological data analysis (TDA) and deep learning methods for data modeled as grids, graphs, and simplicial complexes.

2.1 Topological Data Analysis

We first introduce relevant background on simplicial complexes and their homology, chain complexes, boundary maps between linear spaces generated by p-simplices, and simplicial maps. Additionally, we outline different notions of local neighborhoods on simplicial complexes through adjacency and coadjacency relations. Finally, we describe directional transforms used in TDA, specifically focusing on the Euler Characteristic Transform (ECT), and some of its applications, variations, and properties.

2.1.1 Simplicial Complexes

A **simplicial complex** is a generalization of a graph or network. While graphs can model relational information between pairs of objects (via vertices and edges connecting them), simplicial complexes are able to model higher-order interactions. With this construction, we can still model pair-wise interactions as edges, but also triple interactions as triangles, four-way interactions as tetrahedra, and so on.

The building blocks of simplicial complexes are **p-dimensional simplices** (or p-simplices for short), formally defined as a set of $p + 1$ vertices

$$\sigma_p = (v_0, v_1, \ldots, v_p)\,,$$

$v_i \in V(K)$, where $V(K)$ is a non-empty vertex set and we denote this set by V when K is understood. Note that we often use a subscript on the simplex to denote the dimension of the simplex, i.e. $\dim(\sigma_p) = p$. The cyclic ordering of vertex sets that constitute a simplex induce an orientation of that simplex, which can be fixed if necessary for the given task. For each simplex, there are two possible orientations, each of which are equivalence classes representing all possible cyclic permutations of the vertices which define the simplex. We say that $\sigma_{p-1} = [v_0, v_1, \ldots, v_{n_{p-1}}]$ and

$\sigma_p = [w_0, w_1, \ldots, w_{n_p}]$ have the same orientation if the ordered set of vertices $[v_0, v_1, \ldots, v_{n_{p-1}}]$ is contained in any cyclic permutation of the vertices w_i forming σ_p. Conversely, if the ordered set of vertices for σ_{p-1} are contained in any cyclic permutations for the other orientation of σ_p, then we say that σ_{p-1} and σ_p have opposite orientation.

Simplicial complexes are collections of these p-simplices, glued together with the constraint that the collection is closed under taking subsets. In other words, an **abstract simplicial complex**, K, is a finite collection of non-empty subsets of V such that for a simplex $\alpha \in K$, $\beta \subseteq \alpha$ implies $\beta \in K$. In this case we call β a face of α and write $\beta \leq \alpha$. Abstract simplicial complexes are combinatorial objects, defined in terms of collections of vertices (p-simplices), however they can have geometric realizations corresponding to vertices, edges, triangles, tetrahedra, etc. The dimension of a simplicial complex is defined as the maximum dimension of all of its simplices

$$\dim(K) = \max_{\sigma \in K} \dim(\sigma)\,,$$

where the dimension of a simplex $\dim(\sigma)$ is one less than the cardinality of the vertex set which defines it. For the use of simplicial complexes as input to neural networks in Ch. 3, we need additional notation to keep track of the neural network layer index within the network. We denote the simplicial complex at layer ℓ of a neural network by $K^{(\ell)}$ and its dimension $\mathcal{P}^{(\ell)} = \dim(K^{(\ell)})$. Note that for the purposes of this dissertation, the superscript indexes the neural network layer and does not represent the ℓ-skeleton of the complex as is common in the topology literature.

2.1.2 Chain groups and boundary maps

For $K_{\mathcal{P}}$ a finite abstract simplicial complex with $\dim(K_{\mathcal{P}}) = \mathcal{P}$, let $C_p(K_{\mathcal{P}})$ denote the **chain group** (denoted by C_p when $K_{\mathcal{P}}$ is understood). For dimension $p \geq 0$, the chain group is a finite dimensional vector space over a field $\mathbb{F}$ (e.g., $\mathbb{F} = \mathbb{R}, \mathbb{C}$, or $\mathbb{Z}_2$). The chain group has a basis $\mathcal{B}_p = \{\sigma_i\}_{i=1}^{n_p}$ given by the p-dimensional oriented simplices of K, where n_p is the number of p-simplices of K. Elements of the chain group C_p are linear combinations of p-simplices of the form:

$$\alpha = \sum_{i=1}^{n_p} \alpha_i \sigma_i$$

where the summation is over all oriented p-simplices $\sigma_i \in \mathcal{B}_p$ and $\alpha_i \in \mathbb{F}$. An element $\alpha \in C_p(K)$ is called a p-chain. Addition of p-chains $\alpha, \beta \in C_p$ is defined by,

$$\alpha + \beta = \sum_{i=1}^{n_p} (\alpha_i + \beta_i)\sigma_i$$

The vector space C_p is also equipped with an inner product, which is defined for two p-chains $\alpha, \beta \in C_p$ as,

$$\langle \alpha, \beta \rangle_{C_p} := \sum_{i=1}^{n_p} \alpha_i \overline{\beta}_i \,.$$

Here, $\overline{\beta}_i$ is the complex conjugate of β_i, which is required when $\mathbb{F} = \mathbb{C}$ and can be ignored when $\mathbb{F} = \mathbb{R}$. We will denote this inner product by $\langle \cdot, \cdot \rangle$ when C_p is understood. This inner product defines a norm:

$$\|\alpha\|_{C_p} := \sqrt{\langle \alpha, \alpha \rangle_{C_p}}\,, \quad \alpha \in C_p \,.$$

With this choice of inner product for C_p, the basis $\mathcal{B}_p$ is an orthonormal basis and every $\alpha \in C_p$ can be uniquely written as

$$\alpha = \sum_{i=1}^{n_p} \underbrace{\langle \alpha, \sigma_i \rangle}_{\alpha_i} \sigma_i \,.$$

From the mapping $C_p \to \mathbb{F}^{n_p}$ defined as $\alpha \mapsto (\alpha_1, \ldots, \alpha_{n_p})$, we have that $C_p \cong \mathbb{F}^{n_p}$.

Now we consider a collection of operators that map between these vector spaces generated by p-simplices. The **boundary map** ∂_p, which operates on an oriented simplex $\sigma = [v_0, \cdots, v_p]$ is defined as:

$$\partial_p(\sigma) := \sum_{i=0}^{p} (-1)^i [v_0, \cdots, \widehat{v}_i, \cdots, v_p] \in C_{p-1}(K)\,,$$

where $\widehat{v}_i$ denotes the removal of vertex v_i from the vertex subset. Since we are working in a linear space, this definition can be extended linearly to operate on the p-chains (i.e. the entire space $C_p(K)$) so that we have a map $\partial_p : C_p(K) \to C_{p-1}(K)$ with

$$\partial_p(\alpha) := \sum_{i=1}^{n_p} \alpha_i \partial_p(\sigma_i) \,.$$

The boundary information of each simplex in a simplicial complex is crucial to homology computations. Combinations of these operators also form the Hodge Laplacian operator, which is important

to facilitate message passing neural networks on simplicial complexes. The Hodge Laplacian operator is further discussed in Sec. 2.1.5, and relevant to neural networks discussed in Ch. 3.

Combining chain groups $C_p(K)$ and the boundary maps between them ∂_p, we define a **chain complex** between C_{p+1}, C_p, and C_{p-1} as follows:

$$\cdots \to C_{p+1} \xrightarrow{\partial_{p+1}} C_p \xrightarrow{\partial_p} C_{p-1} \to \cdots ,$$

where ∂_{p+1} and ∂_p are boundary maps on C_{p+1} and C_p respectively. An important standard property of these maps is that the composition of two boundary maps is the zero map.

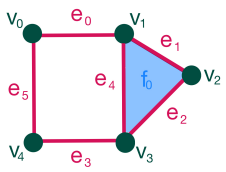
Proposition 2.1.1. $(\partial_p \circ \partial_{p+1})(\alpha) = 0 \quad$ for all $\alpha \in C_{p+1}$

Proof. We show that $(\partial_{p-1} \circ \partial_p)(\sigma) = 0$ for an oriented simplex $\sigma = [v_0, \ldots, v_p]$, which in turn implies that $(\partial_{p-1} \circ \partial_p)(\alpha) = 0$ for all $\alpha \in C_p$,

$$\begin{aligned}
(\partial_{p-1} \circ \partial_p)(\sigma) &= \partial_{p-1}\partial_p[v_0, v_1, \ldots, v_p] \\
&= \sum_{i=0}^{p} (-1)^i \partial_{p-1}[v_0, , \ldots, \hat{v}_i, \ldots, v_p] \\
&= \sum_{i=0}^{p} \left(\sum_{j<i}^{p} (-1)^i (-1)^j [\ldots, \hat{v}_j, \ldots, \hat{v}_i, \ldots] + \sum_{j>i}^{p} (-1)^i (-1)^j [\ldots, \hat{v}_i, \ldots, \hat{v}_j, \ldots] \right) \\
&= 0 \, .
\end{aligned}$$

□

Boundary matrices Boundary maps are an operator (represented as a matrix given a fixed basis) which map between vector spaces C_p and C_{p-1} generated by p-simplices and $(p-1)$-simplices respectively. Each boundary matrix captures incidence relations for a given dimension, keeping track of which simplices of dimension $p-1$ are faces of a simplex of dimension p. For example, in Fig. 2.1, vertices v_0 and v_1 are on the boundary of edge e_0 and edge e_1 is on the boundary of face f_0. For each dimension p, this incidence-relation information is encoded in the matrices corresponding to the boundary operators ∂_p. For clarity in subsequent sections, we equivalently use notation $\partial_p := \mathbf{B}_p \in \mathbb{R}^{n_{p-1} \times n_p}$ to represent the matrix boundary operator, with subscripts indicating



$$|\mathbf{B}_1| = \begin{array}{c} \\ v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{cccccc} e_0 & e_1 & e_2 & e_3 & e_4 & e_5 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{pmatrix} \end{array} \quad |\mathbf{B}_2| = \begin{array}{c} \\ e_0 \\ e_1 \\ e_2 \\ e_3 \\ e_4 \\ e_5 \end{array} \begin{array}{c} \begin{array}{c} f_0 \end{array} \\ \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} \end{array}$$

Figure 2.1 Non-oriented boundary matrices $|\mathbf{B}_1|$ and $|\mathbf{B}_2|$ for an example simplicial complex (left).

the dimension. When necessary to associate a boundary matrix to a specific simplicial complex K, we denote this matrix by $\mathbf{B}_{K,p}$. Note that $\mathbf{B}_0 = \mathbf{0}$, since there are no simplices of negative dimension to map down to.

Orientation Simplex orientation is encoded in boundary matrices using ± 1, with different orientations of the same simplex indicated by different placement of negative values in the boundary matrix. Thus, we can fix an orientation and define entries of oriented p-boundary matrices as follows:

$$\mathbf{B}_p(\sigma_{p-1}, \sigma_p) = \begin{cases} 1 & \text{if } \sigma_{p-1} \text{ and } \sigma_p \text{ have the same orientation} \\ -1 & \text{if } \sigma_{p-1} \text{ and } \sigma_p \text{ have opposite orientation} \\ 0 & \text{if } \sigma_{p-1} \text{ is not a face of } \sigma_p \end{cases}$$

where σ_p and σ_{p-1} are simplices in K. If simplex orientation is not necessary, we use the non-oriented boundary matrix $\left|\mathbf{B}_p\right|$.

2.1.3 Homology and Persistent Homology

Recall from Sec. 2.1.2 that elements of the vector spaces generated by p-simplices (chain groups, $C_p(K)$) are called p-chains, $\alpha \in C_p(K)$. Consider the set of p-chains which have boundary 0, $\partial_p(\alpha) = 0$. These p-chains form a subspace $Z_p \subseteq C_p$,

$$Z_p = \{\alpha \in C_p | \partial_p(\alpha) = 0\} = \ker \partial_p \ ,$$

which we call the **p-cycles**. From the boundary map definition, the p-chains that are in the boundary of $(p+1)$-chains form a subspace $B_p \subseteq C_p$,

$$B_p = \{\partial_{p+1}(\alpha) | \alpha \in C_{p+1}\} = \text{Im } \partial_{p+1} \ .$$

We call these $p-$chains the **p-boundaries**. From the property that the composition of two consecutive boundary maps is the zero map (See Eq. 2.1.1 and proof), we know that the boundaries are contained in the cycles, $B_p \subseteq Z_p$. Then, the **homology group** in dimension p is given by the cycles modulo the boundaries, the quotient group,

$$H_p(K) = Z_p/B_p \ .$$

Elements of $H_p(K)$ are homology classes and we say that a specific p-chain $\alpha \in Z_p$ is a representative of the homology class if the equivalence class $[\alpha]$ is the homology class. Two p-chains α and α' are cycle representatives of the same homology class if $[\alpha] = [\alpha']$. The **p-th Betti number** is the rank of this group, $\beta_p = \text{rank}(H_p(K))$.

Persistent Homology Persistent homology is an illustrious tool in the field of topological data analysis, widely used for study the topological shape of data in applications including, but certainly not limited to, proteins [92, 50], collaboration networks [17, 70], dynamical systems and machining dynamics [60, 86, 46], periodicity in time series and satellite imagery [73, 85], and neuroscience [34, 83, 58, 97].

A **simplicial map** defines a function from one simplicial complex K_a to another K_b, induced by a map from the vertices of K_a to the vertices of K_b. This map $f : V(K_a) \to V(K_b)$ must maintain the property that all images of the vertices of a simplex span a simplex, i.e. $\forall \sigma \in K_a, f(\sigma) \in K_b$ [67]. An example simplicial map is an induced map $\iota : K_i \hookrightarrow K_j$ from the inclusion of a subcomplex K_i into another larger simplicial complex K_j.

A filtration of a simplicial complex K is a nested sequence of subcomplexes,

$$\emptyset = K_0 \subseteq K_1 \subseteq \ldots \subseteq K_n = K \ .$$

We compute persistent homology of a simplicial complex for a fixed filtration as follows. For each step i in the filtration, we compute the homology of the subcomplex $H_p(K_i)$, keeping track of the steps in the filtration in which homology classes are born and when they die. We have induced

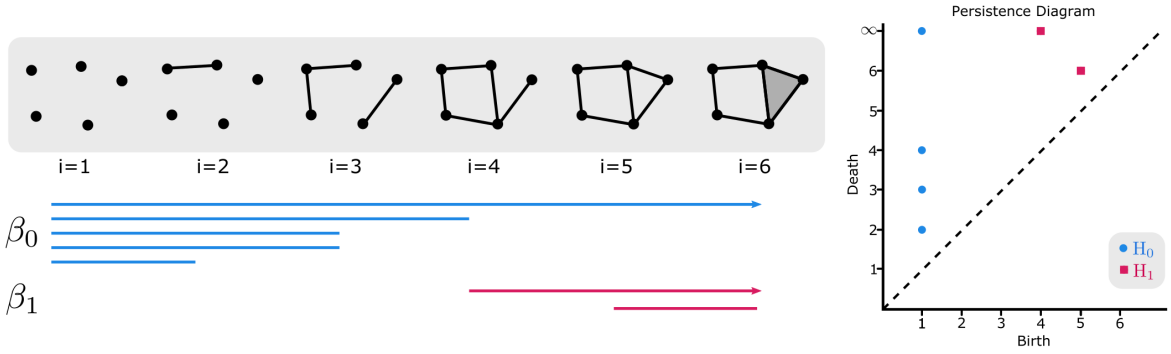


Figure 2.2 A simplex-wise filtration on an example simplicial complex. The barcode showing the persistence of features through the filtration for β_0 and β_1 is shown below. On the right, this information is plotted in a persistence diagram with coordinates given by (birth, death) pairs.

maps $\iota : K_i \hookrightarrow K_j$ by the inclusions $K_i \subseteq K_j$, for all $i < j$ for the sequence of nested subcomplexes.

$$0 \subseteq \ldots \subseteq K_{i-1} \subseteq K_i \subseteq \ldots \subseteq K_{j-1} \subseteq K_j \subseteq \ldots \subseteq K$$

$$0 \hookrightarrow \ldots \hookrightarrow K_{i-1} \hookrightarrow K_i \hookrightarrow \ldots \hookrightarrow K_{j-1} \hookrightarrow K_j \hookrightarrow \ldots \hookrightarrow K \ .$$

We say that a homology class is **born** at K_i if it is not in the image of the induced map $\iota_i : K_{i-1} \hookrightarrow K_i$. We say the same class **dies** at K_j if the image of the induced map $\iota_a : K_{i-1} \to K_{j-1}$ does not contain the image of the homology class but the image of $\iota_b : K_{i-1} \to K_j$ does. By this definition, we have that the homology class persists from index i to index j in the filtration. The paired birth and death values of homology classes give us information about how the topological shape of the complex changes through the lens of the specific filtration chosen. We keep track of this information using multisets of the birth, death pairs in what we call a **persistence diagram**. In these scatter plots, each homology class is given coordinates (i, j), where i is the filtration index in which the class is born, and j is the index in the filtration at which the class dies. Figure 2.2 shows an example of a filtration on a simplicial complex and its accompanying barcode and persistence diagram.

2.1.4 Adjacency

It is often useful to have a concise way to represent the structure of a simplicial complex. For a 1-dimensional simplicial complex, consisting of just vertices and edges, the simplicial complex is a graph and we represent it by the corresponding **adjacency matrix**: a square matrix that

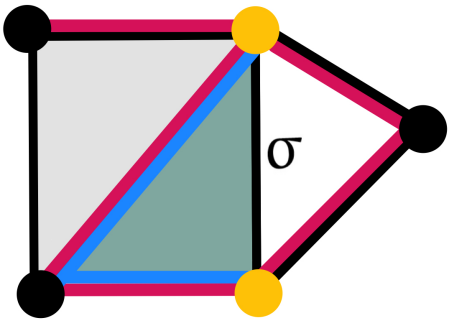


Figure 2.3 For simplex σ, geometrically realized as an edge, its four different types of adjacent simplices are: boundary adjacent (yellow), coboundary adjacent (green), lower adjacent (pink), and upper adjacent (blue).

stores information about which vertices are connected by edges. In general, however, simplicial complexes have higher dimensional structure that must be accounted for. While simplicial complex structure is also captured by adjacency relations, there is more than one way to generalize this idea to higher dimensional simplices. Specifically, p-simplices can relate to each other by either their common upper or lower dimensional neighbors. In order to fully capture the face and coface relations in a simplicial complex, we consider four different adjacency types: boundary adjacent, coboundary adjacent, upper adjacent, and lower adjacent [11], which we define next and are shown for an example simplex in Fig. 2.3.

Fix a p-simplex σ_p. The **boundary adjacent** simplices are the set of $(p-1)$-dimensional faces of the p-simplex, $\{\sigma_{p-1} \mid \sigma_{p-1} \subseteq \sigma_p\}$. This boundary adjacent relation directly corresponds to the standard boundary map for simplicial complexes: $\mathbf{B}_p$. For the same simplex, the set of $(p+1)$-dimensional simplices which have σ_p as a face are its **coboundary adjacent** simplices, $\{\sigma_{p+1} \mid \sigma_p \subseteq \sigma_{p+1}\}$. The coboundary adjacency relation corresponds to the transpose of the standard boundary map: $\mathbf{B}_{p+1}^T$.

The usual notion of adjacency on a graph corresponds to the set of vertices which share an edge; i.e. a higher dimensional simplex with each as a face. More generally, for simplicial complexes, p-simplices are considered **upper adjacent** if there exists a $(p+1)$-simplex that they are both faces of. We can capture upper adjacent neighbor relations in terms of the complex's boundary maps

(the up-down combinatorial Laplacian operator):

$$\mathbf{A}_{up,p} = \mathbf{B}_{p+1}\mathbf{B}_{p+1}^T . \tag{2.1.1}$$

The **lower adjacent** neighbors are p-simplices such that there exists a $(p-1)$-simplex that is a face of both (i.e. both p-simplices are cofaces of a common $(p-1)$-simplex). The lower adjacent neighbor relations can also be written in terms of the complex's boundary maps (the down-up combinatorial Laplacian operator):

$$\mathbf{A}_{low,p} = \mathbf{B}_p^T\mathbf{B}_p .$$

Note that the sum of these two (upper and lower) adjacency operators gives the p-dimensional Hodge Laplacian, which is leveraged in different simplicial neural network architectures to facilitate local information sharing on complexes. Using the Hodge Laplacian as a diffusion operator, there are various existing neural network architectures (e.g. [11, 29]) that operate on simplicial complexes. Simplicial neural networks of this type are the context in which our pooling layer for simplicial complexes, NervePool, described in Ch, 3, can be leveraged.

2.1.5 Boundary map connection to Hodge Laplacian Operator

Boundary matrices are operators in their own regard, however we can also combine boundary matrices of different dimensions and their transposes to form different operators. Combinations of these boundary matrices are closely related to the Hodge Laplacian on a graph, which is a higher-order generalization of the graph Laplacian [28]. In [57], they reconstruct the definition of this operator using algebraic properties of cohomology, in particular using the properties of coboundary maps (the dual of the boundary map ∂_p).

A linear functional is a linear map that takes a vector space (in this case C_p) into the scalar field $\mathbb{F}$; so in particular, $T(\alpha+\beta) = T(\alpha) + T(\beta)$ and $T(c\alpha) = cT(\alpha)$ for $\alpha, \beta \in C_p$ and $c \in \mathbb{F}$. **Coboundary operators** map between cochain groups, which are defined by the homomorphism group

$$C^p = \mathrm{Hom}(C_p, \mathbb{F}) := \{T : C_p \to \mathbb{F} \mid T \text{ is a linear functional}\} ,$$

that is, the set of linear functionals that map from the p-th chain group to the specified field. Another name for C^p is the dual space of C_p and elements of C^p are called cochains. The dual of the boundary map ∂_p maps between cochain groups,

$$\partial_p^* : \mathrm{Hom}(C_{p-1}, \mathbb{F}) \to \mathrm{Hom}(C_p, \mathbb{F}).$$

This operator ∂_p^* takes in a linear functional $T \in \mathrm{Hom}(C_{p-1}, \mathbb{F})$ and outputs a new linear functional $\partial_p^*(T) \in \mathrm{Hom}(C_p, \mathbb{F})$. We have vector spaces C_p and C_{p-1} over a field $\mathbb{F}$ and a linear map between them given by the boundary map, $\partial_p : C_p \to C_{p-1}$. By definition, the dual of the map ∂_p is the linear map between cochain groups, $\partial_p^* : C^{p-1} \to C^p$, given by,

$$\partial_p^*(T) = T \circ \partial_p \text{ so that } \partial_p^*(T)(\alpha) := T(\partial_p(\alpha)),$$

for all $\alpha \in C_p$ and $T \in C^{p-1}$. In the same way as for the chain complex, we get the associated cochain complex by mapping between subsequent cochain groups using coboundary maps:

$$\cdots \leftarrow \mathrm{Hom}(C_{p+1}, \mathbb{F}) \xleftarrow{\partial_{p+1}^*} \mathrm{Hom}(C_p, \mathbb{F}) \xleftarrow{\partial_p^*} \mathrm{Hom}(C_{p-1}, \mathbb{F}) \leftarrow \cdots . \tag{2.1.2}$$

Dual to the analogous property for composing boundary maps (see Prop. 2.1.1), we have the following property for applying consecutive coboundary maps,

Proposition 2.1.2.

$$(\partial_{p+1}^* \circ \partial_p^*)(T) = \mathbf{0} \quad \text{for all } T \in \mathrm{Hom}(C_{p-1}, \mathbb{F}), \tag{2.1.3}$$

where $\mathbf{0} \in \mathrm{Hom}(C_{p+1}, \mathbb{F})$ is understood as the linear functional defined as $\mathbf{0}(\alpha) = 0$ for all $\alpha \in C_{p+1}$.

Proof. We use the property of ∂_p and ∂_p^* as dual maps to verify that $(\partial_{p+1}^* \circ \partial_p^*)(T)(\alpha) = 0$ for coboundary maps. In particular, since ∂_p^* is the dual map of ∂_p, we have that

$$\partial_p^*(T)(\alpha) = T(\partial_p(\alpha)) \text{ for all } T \in C^{p-1} \text{ and } \alpha \in C_p.$$

Using this property of dual maps, we can rewrite $(\partial_{p+1}^* \circ \partial_p^*)(T)(\alpha)$ in terms of boundary maps,

$$\begin{aligned}(\partial_{p+1}^* \circ \partial_p^*)(T)(\alpha) &= \partial_{p+1}^*(\partial_p^* T)(\alpha) \\ &= \partial_p^*(T)(\partial_{p+1}(\alpha)) \\ &= T(\partial_p(\partial_{p+1}(\alpha))).\end{aligned}$$

We use the property of applying consecutive boundary maps: $(\partial_p \circ \partial_{p+1})(\alpha) = 0$ (Prop. 2.1.1) to verify,

$$T(\partial_p(\partial_{p+1}(\alpha))) = T(0) = 0.$$

□

Using this setting for coboundary maps, the Hodge Laplacian is the operator which can be represented by a matrix $\mathbf{L}_p : \mathrm{Hom}(C_p, \mathbb{F}) \to \mathrm{Hom}(C_p, \mathbb{F})$ defined by,

$$\mathbf{L}_p(T) := (((\partial^*_{p+1})^\dagger \circ \partial^*_{p+1}) + (\partial^*_p \circ (\partial^*_p)^\dagger))(T), \quad p \geq 0\,, \tag{2.1.4}$$

where † denotes the adjoint and ∂^*_0 is the zero map (in other words it becomes the zero map for $\mathbf{L}_0$). For our setting of finite abstract simplicial complexes, we can restrict to the choice of field $\mathbb{F} = \mathbb{R}$, and thus rewrite the p-dim Hodge Laplacian [28, 57] in terms of boundary maps and coboundary maps (the transpose):

$$\mathbf{L}_p = \mathbf{B}_p^T\mathbf{B}_p + \mathbf{B}_{p+1}\mathbf{B}_{p+1}^T \tag{2.1.5}$$

Note that for $p = 0$, this is equivalent to the well-studied graph Laplacian $\mathbf{B}_1\mathbf{B}_1^T$ since $\mathbf{B}_0$ is the zero map.

2.1.6 Euler Characteristic

The Euler Characteristic is a simple yet powerful description of the topology of a shape. For a simplicial complex $K \in \mathbb{R}^d$, we can compute this topological invariant by an alternating sum of the number of simplices in each dimension,

$$\chi(K) = \sum_{p=0}^{d} (-1)^i n_p\ ,$$

where n_p is the number of p-dimensional simplices in K [5]. For a general triangulated polygon, the Euler Characteristic is simply $\chi(K) = |V| - |E| + |F|$, the alternating sum of the number of vertices, edges, and faces of the polygon.

Equivalently, the **Euler Characteristic** can be computed by an alternating sum of Betti numbers across dimensions

$$\chi(K) = \sum_{p=0}^{d} (-1)^i \beta_p\ .$$

Recall Betti numbers are defined by the rank of homology groups $\beta = \text{rank}(H_p)$, see Section 2.1.3.

The Euler Characteristic is not, however, discriminatory enough to distinguish between shapes which are homeomorphic. In fact Leonhard Euler proved that all platonic solids (e.g. a hollow sphere, hollow tetrahedron, hollow cube, etc) have an Euler Characteristic of 2.

Instead of one integer value which represents the topology of an entire shape M, we can instead define a filtration on the shape, computing the Euler Characteristic of sublevel sets of the complex for various steps in the filtration. This results in a collection of Euler Characteristic values, called the **Euler Characteristic Curve (ECC)**, a piece-wise constant integer valued function, which is in practice a vector of integers. For the Specifically, we define a filtration on M to be the set of sublevel sets $\{M_a\}_{a\in[0,t]}$ defined by

$$M_a = \{\sigma \text{ in } M \mid f(a) \leq a\} \text{ for } a \in [0, t] \ ,$$

where $f : M \to \mathbb{R}$ is any filtration function on the shape M. The ECC of M with is given by the function

$$\begin{aligned} ECC : \mathbb{R} &\to \mathbb{Z} \\ t &\mapsto \chi(f^{-1}(\infty, t]) \ , \end{aligned}$$

and the ECC maps each stopping threshold t of the filtration function f to the Euler Characteristic of the sub-level set of M at threshold t.

2.1.7 Directional Transforms for Topological Data Analysis

Directional transforms provide a flexible framework to model high dimensional surfaces in a more computationally feasible manner, using a choice of topological representation. Instead of considering topological summaries of input data from the vantage point of a single choice of direction and associated filtration, we can instead consider many possible directions and call this collection of topological summaries from various perspectives a directional transform. In their 2014 paper, Turner, Mukherjee, and Boyer define a Persistent Homology Transform (PHT) and Euler Characteristic Transform (ECT), which are statistics used to represent surfaces in $\mathbb{R}^d$

[84]. For the PHT, a collection of persistence diagrams (multiscale topological summaries) are computed by applying filtrations from varying directions on the surface. Similarly for the ECT, Euler Characteristics of filtrations are computed from different fixed directions on the surface.

For a shape $M \subset \mathbb{R}^d$, we associate to each direction $\omega \in \mathbb{S}^{d-1}$ a shape summary by scanning M in direction ω. We consider the general class of shapes which are a subset of $\mathbb{R}^d$ and as such can be written as a simplicial complex. The chosen shape summary measures the topology of sublevel sets of a height function for each direction $\omega \in \mathbb{S}^{d-1}$. The **Euler Characteristic Transform (ECT)**, is a directional transform of this type, where the choice of topological summary is the Euler Characteristic curve. Define f_ω to be the filtration function for direction ω such that all points of the shape $x \in M$ can be written as the standard dot product of vectors,

$$\begin{aligned} f_\omega : M &\to \mathbb{R} \\ x &\mapsto \langle x, \omega \rangle = \sum x_i y_i \ . \end{aligned}$$

Then, the ECT defines a map from directions on the sphere to a cylinder,

$$\begin{aligned} ECT(M) : \mathbb{S}^{d-1} &\to \mathbb{R} \times S^{d-1} \\ \omega &\mapsto ECC_\omega \ . \end{aligned}$$

Computing the ECT of binary images In addition to data represented as a simplicial complex, we can also compute the ECT of different data types such as images. In order to compute the ECT by counting vertices, edges, and faces of sublevels sets of defined filtration of a complex, we first must represent the image as a cubical complex. Cubical complexes are a natural way to represent image data [51]. As opposed to simplicial complexes, which are built up through triangulations (i.e. faces are groups of 3 vertices), with this representation the building blocks are vertices, line segments, squares, cubes (and higher-dimensional counterparts) such that faces are groups of 4 vertices. To represent a 2D grayscale image data as a cubical complex, we represent each non-zero valued pixel as a vertex. To construct the higher dimensional cells, edges are added between pairs of pixels which are directly adjacent (in the direct 4 adjacent neighborhood sense) and square faces

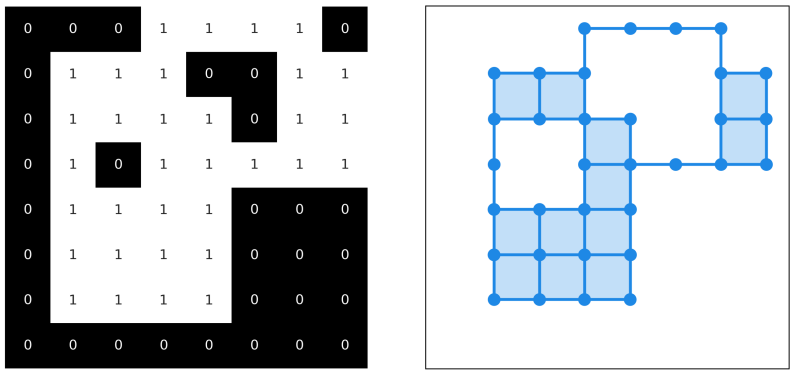


Figure 2.4 An example of a binary image (left) and the corresponding cubical complex (right).

are added for groups of 4 pixels which are all pair-wise adjacent. This construction is extended for cubes in higher dimensions for images in 3D, for example. Figure 2.4 shows an example of this process to represent a binary image as a cubical complex. We employ this method to compute the ECT of binary images, represented as cubical complexes, in an application in Ch. 4.

Properties of the ECT By computing a chosen summary statistic from finitely many directions on the surface, the directional transform result is a lower dimensional representation of the original surface, which retains important structural information (according to the choice of topological summary). In [84], they prove stability results that ensure the PHT metric can be approximated using finitely many persistence diagrams, meaning that the number of directions necessary to compute PH is indeed finite. There has been additional work done to determine more specific upper bounds on this number [26]. The PHT is a sufficient statistic and the map that the PHT defines is, in fact, injective for surfaces and shapes (in $\mathbb{R}^2$ and $\mathbb{R}^3$) [84]. This work was extended to show injectivity holds for a finite number of necessary directions with M of $\mathbb{R}^d$ [84, 26]. Injectivity guarantees that these transforms will not represent different surfaces by the same topological summary. Additionally, both the finite number of directions and injective properties of the PHT also apply for the ECT, using Euler characteristics instead of persistence diagrams as the choice of topological representative, which significantly reduces computational complexity. Indeed, the proof that the PHT is injective is constructive and built upon Betti numbers, indicating that the

same injectiveness is automatically true when you swap Euler curves for persistent homology.

This provides motivation to use the ECT for applications of classification of shapes by their topology, since it is a simple computation (especially in comparison to persistent homology) and given enough directions sampled, we have theoretical guarantees of an injective function. Injectivity of the ECT is also important for applications to actual data because it allows for the comparison of shapes through comparison of their respective collections of Euler Curves.

The directional transform is a very flexible framework; the choice of topological summary statistic can be application dependent. In addition to persistence diagrams and Euler Characteristics, we can choose to represent the surface (or discrete dataset modeling a surface) using mapper graphs [82] or Reeb graphs [75], which are more simplified, discrete representations. Independent of the choice of topological signature, however, the directional transform method always encodes additional structure from the relationship between directions in the form of circle-valued data.

There has been work towards shape reconstruction of 2D images and plane graphs from their ECT or PHT alone [6, 31]. In Fasy et al 2024, they provide insight into sufficient conditions for discretizations of the PHT to provide faithful shape reconstruction, in the sense that the shape can be unambiguously reconstructed. Additionally, they provide stability results on the discretizations for arbitrary dimensions [32].

Variations and applications of the ECT There are a few notable variations of the ECT that have been adapted for various applications including the Smooth Euler Characteristic Transform (SECT)[24], Weighted Euler Characteristic Transform [44] and the Differentiable Euler Characteristic Transform (DECT) [78]. Statistical inference properties and applications of the SECT are further investigated in [62] and [65], while [66] presents a survey description and application of the SECT to a $2D$ fern leaf scan, represented as an embedded simplicial complex.

Here, we review how the SECT adapts the original ECT by replacing the step-functions of standard ECC functions with smooth, continuous functions in the form of Smooth Euler Characteristic Curves (SCC). These SECC functions have a Hilbert space structure, which circumvents the stabil-

ity issues and challenges applying statistical methods when using the step-function Euler curves. The SECT can be computed directly from the ECT by using the average Euler Characteristic value of each direction to mean center each ECC vector and then integrating the ECC across all filtration thresholds. Assume the input is an embedded shape M, contained in a bounding ball of radius r. For a fixed choice of direction $\omega \in \mathbb{S}^{d-1}$, we first compute the average Euler Characteristic value,

$$\overline{ECC}_\omega = \frac{1}{2r} \int_{-r}^{r} ECC_\omega(a)\ da\ .$$

Using this average, we then zero-center the Euler Characteristic values and integrate over all of the thresholds to define the Smooth Euler Curve, $SECC_\omega : [-r, r] \to \mathbb{R}$ by the map,

$$t \mapsto \int_{-r}^{t} (ECC_\omega(a) - \overline{ECC_\omega})\ da\ .$$

Applying this smoothing to the entire ECT matrix (i.e. each column corresponding to a direction) extends the SECC to a Smooth Euler Characteristic Transform, $SECT(M) : \mathbb{S}^{d-1} \to \text{Fun}([-r, r], \mathbb{R})$, which maps $\omega \mapsto SECC_\omega$.

Overall, this defines a function from the space of directions on the $d-1$ sphere to the space of functions from $[-r, r]$ to $\mathbb{R}$.

In [78], the authors present the Differentiable Euler Characteristic Transform (DECT), an ECT neural network layer (or loss term) which is end-to-end trainable in the sense that the layer is differentiable with respect to the ECT directions and coordinates of the shape. While most previous work use the ECT as static feature descriptors, they suggest using the ECT as either a computational layer, or as a loss term in a deep learning model. The primary contribution of this method is a differentiable version of the ECT computation, swapping the alternating sum of indicator functions

$$\begin{aligned} ECT : \mathbb{S}^{d-1} \times \mathbb{R}^d &\to \mathbb{Z} \\ \omega, t &\mapsto \sum_{p}^{\mathcal{P}} (-1)^p \sum_{\sigma_p} \mathbb{1}[f_\omega(x_{\sigma_p}), \infty)(t) \end{aligned}$$

with a sigmoid function to make the function differentiable,

$$DECT : \mathbb{S}^{d-1} \times \mathbb{R}^d \to \mathbb{Z}$$
$$\omega, t \mapsto \sum_{p}^{\mathcal{P}} (-1)^p \sum_{\sigma_p} S(\lambda(t - f_\omega(x_{\sigma_p}))) \ ,$$

where x_{σ_p} is the feature vector of a p-dimensional simplex and S a sigmoid function with tuning parameter λ.

We note also that the SECT could potentially be a useful alternative method in this setting due to its differentiability. However, this method is in contrast to the method we suggest in Ch. 4 because we use the ECT as a static shape descriptor for input to a convolutional neural netowrk model.

2.2 Deep Learning

In this section, we discuss various neural network architectures designed to leverage the inherent structure of different data types- in particular grids, graphs, and simplicial complexes. We think of these different data structures and their relation to each other to define different versions of neural networks. First, we consider the case of image data, which we can think of as signals on a Euclidean 2D grid. Traditional CNNs are designed relying on the regular pixel structure of the image to perform convolutions with filters at different locations on the grid. We describe general CNN architectures in Sec 2.2.1. We then discuss the case of using graphs as the input data structure; graphs are a generalization of grids, where instead of necessarily having a regular structure, there are vertices with different degrees and no euclidean structure. For the input space of graphs, we can use GNNs for machine learning tasks, which generalizes convolution on 2D images to the space of graphs. This type of neural network architecture, and its properties are discussed in Sec 2.2.2. Finally, in Sec 2.2.3 we consider a further generalization to the space of simplicial complexes and generalizations of the neural network architecture that allow us to use simplicial complexes as input to the model.

2.2.1 Convolutional Neural Networks

Convolutional neural networks are a widely used tool for image classification and more generally as a basis for development of deep learning architectures. First introduced in 1990 by Le Cun et

al., as the first convolution neural network (or ConvNet) trained with backpropagation for the classification of low resolution images, this network architecture has been foundational in the deep learning literature [25].

Due to their reliance on the convolution operation, CNNs have foundations relating to spectral theory and graph signal processing. Definitions of convolution and application to convolution layers for neural networks from the signal processing perspective are described in this background section [12, 48, 27]. Convolution is an operation that can be applied to extract information from a signal $f : \mathbb{R} \to \mathbb{R}$ using a filter $h : \mathbb{R} \to \mathbb{C}$ in the following way,

$$f * h(t) := \int_{\mathbb{R}} f(u)h(t-u)du \ ,$$

where $h(t-u)$ is translation of filter h. In the case of CNNs, we are working with the discrete case of fixed grids and a signal on the grid (pixel values). Let x represent the signal (an $N \times M$ image) and let h be a small filter to be used for convolution. These small filters, 3×3 for example, are also referred to as kernels in the deep learning literature and correspond to the learnable parameters of the model. Let u and v be pixels, with row and column coordinates (u_1, u_2) and (v_1, v_2), respectively. Convolution of image x with filter h is defined as

$$(x * h)(u) = \sum_v x(v)h(u-v),$$

where $(u-v) = (u_1 - v_1, u_2 - v_2)$ and the convolution operator takes a sum over the support of h in local neighborhood of u, meaning the pixels v surrounding u. If h is a 3×3 filter for example, we think of $h(u-v)$ as being supported on $u \cup N(u)$, where the neighborhood $N(u)$ is all of the pixels directly adjacent to pixel u, including those diagonally adjacent. We use the convolution operator to convolve small learned filters with the input image at each layer of the CNN, defining a mapping from each image at layer ℓ to its representation at layer $\ell + 1$, for $\ell \geq 0$. Suppose at layer ℓ, we have an image $x_i^{(\ell)} \in \mathbb{R}^{N \times M}$ with c_ℓ channels,

$$(x_i^{(\ell)})_{i=1}^{c_\ell}.$$

In the case of grayscale input images, the initial layer is a single channel so $c_0 = 1$, however for RGB color images we have $c_0 = 3$ for each of the red, green, and blue color channels. Then, we

convolve the filters with the image to get the image at layer $\ell + 1$ with $c_{\ell+1}$ channels, defined as

$$x_j^{(\ell+1)} := \psi\Big(\sum_{i=1}^{c_\ell} x_i^{(\ell)} * h_{ij}^{(\ell)}\Big)$$

for $1 \leq j \leq c_{\ell+1}$. Here, $h_{ij}^{(\ell)}$ is the collection of learned filters, for $1 \leq i \leq c_\ell$ and $1 \leq j \leq c_{\ell+1}$ and ψ is any choice of pointwise nonlinearity (e.g. ReLu) satisfying $\psi(x)(u) := \psi(x(u))$.

Convolutional neural networks combine these convolution layers with pooling layers to build models for used for image and video recognition, image classification and segmentation, recommender systems, and other computer vision tasks. As a type of feed-forward neural network, CNNs are prone to overfitting and often require the use of regularization techniques to encourage better generalization of the model to data it has not been trained on. Commonly used regularization techniques include penalizing parameters (e.g. through weight decay on the optimizer) and reducing the number of connections in the network (e.g. through random dropout or skipping connections in the network).

2.2.2 Graph Neural Networks

Graphs are useful devices to model objects (vertices) and the relations between them (edges). As such, graphs provide a flexible framework to model a wide variety of data: social networks, chemical compounds, protein interaction networks, knowledge graphs, etc. Graph Neural Networks (GNNs) are a class of deep learning methods which operate on the input space of graphs. We can think of GNNs as a generalization of CNNs, where instead of a fixed, regular graph (i.e. pixel grid) with the task to classify different signals on the grid (i.e. images), we instead work with a more general class of graphs and the signals defined on them. In this setting, however, there are also various learning tasks well-suited for GNNs including signal classification, node classification, link prediction and clustering, all of which capture the graph structure using message passing between vertices. For node classification, from a fixed (possibly weighted) graph $G = (V, E, w)$ with some vertices labeled $V_L \subseteq V$ and others not $V - V_L \subseteq V$, the task is to assign labels to the unlabeled vertices. Similarly, link prediction involves using information about existing edges in a fixed graph $G = (V, E)$ to predict if additional edges should exist between two vertices $v_1, v_2 \in V$ for which

$(v_1, v_2) \notin E$. As opposed to standard CNNs, which operate on input spaces with grid-like structure, GNNs must incorporate the inherently non-Euclidean structure encoded in graphs.

Architecture In the graph domain, GNNs are able to model dependencies between nodes. They are widely used for both vertex-level and graph-level tasks. The overall method is such that each vertex aggregates information from their neighbors to generate an embedding. These embeddings can then be separately used for vertex-level tasks, or aggregated to perform graph-level tasks. For each of these vertex embeddings, the learned vector representation results from aggregating features on the vertex itself, as well as features on its neighboring vertices. For a standard graph-level GNN layer, features are updated using a function of the general form,

$$\mathbf{X}^{(\ell+1)} = \psi(\mathbf{A}\mathbf{X}^{(\ell)}\Theta^{(\ell)}_{neigh} + \mathbf{X}^{(\ell)}\Theta^{(\ell)}_{self}),$$

where $\mathbf{A}$ is an adjacency style matrix, $\mathbf{X}$ are features on vertices, ψ a non-linearity, and Θ are learnable weight parameters. The learnable weights control the message passing between the features on a vertex itself, and separately the passing of information from a vertex's neighbors. The embedding function must have a notion locality (i.e. which vertices are in the local neighborhood of each vertex), and must be able to aggregate information, both of which are satisfied by using the adjacency matrix of the graph. Vertices should be embedded into the lower-dimensional space such that "similar" vertices are close to each other, where the notion of similarity is dependent on both the application and the choice of loss function.

One important property of message passing GNNs is the correspondence between the number of layers in the network and the distance on the graph from which each vertex aggregates information. In particular, a vertex embedding for vertex v at layer ℓ contains information up to ℓ-hops away from v. Figure 2.5 shows an example of 1-hop and 2-hop neighborhoods for a given vertex. Depending on the application, a graph level task may require prohibitively many message passing GNN layers for two given nodes in the graph to share information. While additional layers expand the distance on the graph by which information is aggregated, they also increase the computational cost of the model.

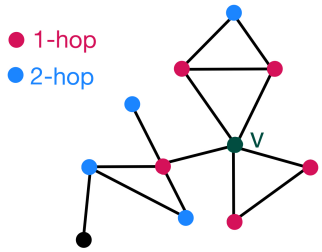


Figure 2.5 An example of 1-hop (pink) and 2-hop (blue) neighborhoods for a given vertex, v (green).

In addition to this message passing-style (spatial) perspective of GNNs, we can also consider the convolution-style (spectral) perspective, which relies on convolution operations of a signal with a graph filter. Convolution graph neural networks (GCNs) extend directly from the definition of CNNs, with convolution defined from the signal processing perspective, more generally on the graph instead of the pixel grid [12, 48, 27]. We must adjust the definition of convolution defined on graphs from the CNN grid version, $x * h(u) = \sum_v x(v)h(u-v)$, because graphs have no inherent ordering of vertices which prohibits translation $h(t-u)$ of the filter h in a coherent way. In this setting, we use the concept of the graph Fourier transform to define convolution. For $\mathbf{x} : V \to \mathbb{R}$, the graph signal, and $\mathbf{h} : V \to \mathbb{R}$, the graph filter, we have the graph Fourier transform of $\mathbf{x}$ given by the inner product,

$$\hat{\mathbf{x}}(r) := \langle \mathbf{x}, \boldsymbol{\varphi}_r \rangle \ ,$$

where $\boldsymbol{\varphi}_1, \ldots, \boldsymbol{\varphi}_R$ are eigenvectors of the graph Laplacian (recall the definition in Sec. 2.1.5) of the graph. Then, we can define graph convolution using the graph Fourier transform by,

$$\mathbf{x} * \mathbf{h}(u) := \sum_{r=1}^{R} \hat{\mathbf{x}}(r)\hat{\mathbf{h}}(r)\boldsymbol{\varphi}_r(u) \ .$$

Equivalently, we can write the graph convolution in terms of matrices by letting $\mathbf{H}$ be the $R \times R$ diagonal matrix with $\hat{\mathbf{h}}$ on the diagonal and $\boldsymbol{\Phi}$ be the matrix of eigenvectors such that column r corresponds to $\boldsymbol{\varphi}_r$,

$$\mathbf{H} = \begin{bmatrix} \hat{\mathbf{h}}(1) & 0 & \ldots & 0 \\ 0 & \hat{\mathbf{h}}(2) & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \ldots & 0 & \hat{\mathbf{h}}(R) \end{bmatrix} \quad \text{and} \quad \boldsymbol{\Phi} = \begin{bmatrix} | & | & & | \\ \boldsymbol{\varphi}_1 & \boldsymbol{\varphi}_2 & \ldots & \boldsymbol{\varphi}_R \\ | & | & & | \end{bmatrix} .$$

Then, graph convolution is given by $\mathbf{x} * \mathbf{h} = \mathbf{\Phi H \Phi}^T \mathbf{x}$ and the GCN which defines a mapping from each graph at layer ℓ to its representation at layer $\ell + 1$ for $0 \leq \ell < L$ (the total number of layers in the model), is

$$\mathbf{x}_j^{(\ell+1)} := \boldsymbol{\psi}\Big(\sum_{i=1}^{c_\ell} \mathbf{x}_i^{(\ell)} * \mathbf{h}_{ij}^{(\ell)}\Big) = \boldsymbol{\psi}\Big(\sum_{i=1}^{c_\ell} \mathbf{\Phi H}_{ij}^{(\ell)} \mathbf{\Phi}^T \mathbf{x}_i^{(\ell)}\Big) \ .$$

In this CGN, the collection of filters $\hat{\mathbf{h}}(r)^{(\ell)}$ on the diagonals of $H_{ij}^{(\ell)}$ are the learned parameters for each layer. Again, considering $\mathbf{h}$ to be a 3×3 filter, for standard convolution we think of $\mathbf{h}(u - v)$ as being supported on $u \cup N(u)$ and in the graph case, call it a 1-hop filter because $N(u)$ is the 1-hop neighborhood of vertex u on the graph.

Expressivity While the "graph isomorphism problem" in graph theory remains a problem with unknown complexity, the Weisfeiler-Lehman (WL)-test [90] is able to determine if two graphs are non-isomorphic. Once reaching the stopping criteria for the algorithm, the resulting claim for the two graphs is either: (i) the graphs are not isomorphic or (ii) the graphs are possibly isomorphic. That is to say, the WL-test heuristic is necessary, but not sufficient to show that two graphs are isomorphic.

The WL-test determines if two graphs are non-isomorphic by producing a canonical form of each graph through iterations of recoloring the vertices of the graphs. Differing canonical forms indicate that the graphs are non-isomorphic. However, in the case that the canonical forms are identical, the test is not sufficient to determine that the two graphs are indeed isomorphic. In this setting, coloring of vertices is analogous to labels on the graph, such that two graphs are isomorphic (up to permutations of labels). In order to produce the canonical form of each graph, the graphs are initialized with an identical coloring of each vertex. Then, for each vertex consider the multiset consisting of its own color and all of the colors of neighboring vertices. In these multisets, elements can appear more than once and order does not matter. Using the multiset labels for each vertex, reassign each vertex a new color using a hash function. Further iterations continue the recoloring using hashes of the color multisets into new color labels. Once the coloring is stable from one iteration to the next, the algorithm terminates, and the relationship between the two

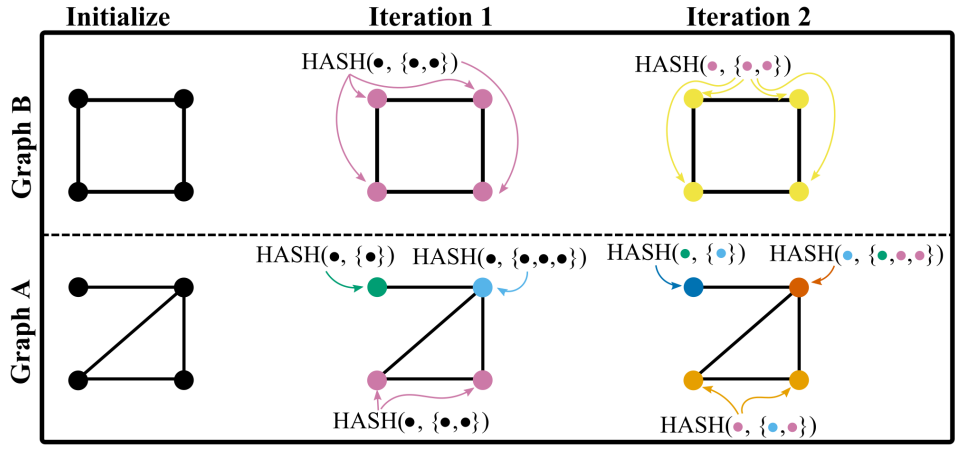


Figure 2.6 Graph A and Graph B are not isomorphic. The WL-test algorithm reaches the stopping condition after 2 iterations and the canonical forms of graphs A and B are different (represented by different colorings), resulting in the determination that Graph A and B are not isomorphic.

graphs is determined to be either non-isomorphic or inconclusive. Figure 2.6 shows an example of two graphs that are not isomorphic and have different canonical forms according to their WL-test coloring.

The expressive power of standard GNNs for graph level tasks can be measured by their relation to the WL graph isomorphism test. In this setting, expressivity refers to the ability of a GNN to distinguish between two graphs that are non-isomorphic: producing different embedded representations for graphs that are not isomorphic. GNNs typically apply a locally permutation-invariant function to aggregate the neighbor features for each node, which results in a permutation equivariant function on the entire graph [61]. If the local aggregator function is injective, the expressive power of the GNN is equivalent to the WL-test. Specifically, it has been shown that GNNs are no more expressive than the WL-test [93].

2.2.3 Simplicial Complex Neural Networks

The idea of using data with topological structure in combination with deep learning has gained significant traction in the topological data analysis community. A recent survey by Papillon et al. explores various architectures in this research space- coined “Topological Deep Learning" [72]. Here, we outline architectures for simplicial complexes specifically, from two early publications:

Simplicial Neural Networks (SNN) [29] and Message Passing Simplicial Neural Networks (MPSN) [11]. While GNNs leverage the local neighborhood of each node in the graph in order to learn node-level (and subsequently graph-level) embeddings, message passing neural networks on simplicial complexes require different notions of the local neighborhood of a simplex due to the higher dimensional structure. For a given simplex, the neighboring simplices can be specified by both upper- and lower- adjacency definitions. For example, an edge is adjacent both to its vertices (lower dimension) and triangles (higher dimension). This bidirectional dimension relation between simplices must be incorporated into the message passing framework to learn simplex embeddings.

If each of these adjacency types is considered independently, we have two separate neural networks for simplex dimension $p \in [0, \mathcal{P}^{(\ell)}]$. The upper adjacent version updates the features on p-simplices using the adjacency matrix and has the following architecture,

$$\mathbf{X}_{p,adj}^{(\ell+1)} = f(\mathbf{A}_p, \mathbf{X}_p^{(\ell)}; \Theta_{p,adj}^{(\ell)}), \tag{2.2.1}$$

where $\mathbf{A}_p$ is the adjacency matrix of p-simplices (recall definition of upper adjacency from Sec. 2.1.4), $\mathbf{X}_p^{(\ell)}$ contains features on p-simplices, and $\Theta_{p,adj}^{(\ell)}$ is a learnable weight matrix. It aggregates simplex neighborhood information, where a p-simplex is adjacent to another p-simplex if they are faces of a common $(p+1)$-simplex. The lower adjacent version updates features on p-simplices using co-adjacency to define which simplices are in its local neighborhood:

$$\mathbf{X}_{p,coadj}^{(\ell+1)} = f(\mathbf{C}_p, \mathbf{X}_p^{(\ell)}; \Theta_{p,coadj}^{(\ell)}), \tag{2.2.2}$$

where $\mathbf{C}_p$ is the coadjacency matrix of p-simplices (recall the equivalent definition of lower adjacency matrix from Sec. 2.1.4). In this network, it aggregates simplex neighborhood information, where a p-simplex is coadjacent to another p-simplex if they are cofaces of a common $(p-1)$-simplex.

However, these two example frameworks with separate message passing layers for adjacency and coadjacency (Equations 2.2.1 and 2.2.2) lack two desirable properties for message passing on a simplicial complex. In particular, it:

Property 2.1. Does not mix across different simplex dimensions p , and

Property 2.2. Does not synthesize adjacency and co-adjacency information.

To incorporate both of these requirements, instead of the previously described message passing layers, the Hodge Laplacian is used to define a message passing framework on simplicial complexes, which we describe in the subsequent paragraphs.

Simplicial neural network architectures Instead of using $\mathbf{A}_p$ and $\mathbf{C}_p$ separately for message passing, we utilize the $p-$dim Hodge Laplacian, L_p, which intrinsically includes both adjacency and co-adjacency structural information. Two foundational deep learning architectures for the simplicial complex domain that were recently proposed are: Simplicial Neural Networks (SNN) [29] and Message Passing Simplicial Neural Networks (MPSN) [11]. In this section, we outline the message passing framework on simplices and measure of model expressiveness proposed in [11]. Additionally, we derive an alternative notation for the MPSN.

SNNs are defined on the space of attributed simplicial complexes and extend convolution to act on higher-dimensional simplices using powers of the p-dimensional Hodge Laplacian,

$$\mathcal{F}_p^{-1}(\varphi_W) *_p T = \sum_{i=0}^{N} W_i L_p^i T\,,$$

where $\mathcal{F}_p^{-1}$ is the generalized inverse Fourier transform convolved with cochain $T \in \mathrm{Hom}(C_p(K_p), \mathbb{R})$. Here, L_p^i denotes the ith power of the p-dimensional Hodge Laplacian and convolution is defined using the filter $\varphi_W \in \mathbb{R}^{(|K_p|)}$, which is a function with small support parameterized by weights W. The choice of convolutional filter, φ_W, is parameterized as an N-degree polynomial of L_p, where N is chosen to be small which forces convolutions to be local.

MPSNs use a generalized message passing framework in which features on simplices of different dimensions interact [11]. In particular, the authors propose the following MPSN layer,

$$H_p^{out} = \psi\left(M_p H_p^{in} W_p + U_p H_{p-1}^{in} W_{p-1} + O_p H_{p+1}^{in} W_{p+1}\right)\,, \tag{2.2.3}$$

where H_p^{out} is the output feature matrix, ψ is an entry-wise activation function, W are learnable weight matrices, and M_p, U_p, and O_p are some choice of adjacency matrices corresponding to maps on cochain groups $M_n : C^p \to C^p$, $U_n : C^{p-1} \to C^{p-1}$, and $O_n : C^{p+1} \to C^{p+1}$.

To make the utility of boundary operators (and adjacency terms of the Hodge-Laplacian) in this message passing more clear, we rewrite the MPSN layer with the following construction. For use in developing the nervePool (Ch .3) simplicial pooling layer, it is useful to consider the MPSN in terms of this alternative notation. Define the following substitutions,

$$\begin{aligned} M_p &= \mathbf{L}_p^{(\ell)} & H_p^{out} &= \mathbf{X}_p^{(\ell+1)} \\ U_p &= (\mathbf{B}_p^{(\ell)})^T & H_p^{in} &= \mathbf{X}_n^{(\ell)}. \\ O_p &= \mathbf{B}_{p+1}^{(\ell)} \end{aligned}$$

Substituting this notation gives the following message passing layer,

$$\begin{aligned} \mathbf{X}_p^{(\ell+1)} &= \psi\left[\mathbf{L}_p^{(\ell)}\mathbf{X}_p^{(\ell)}W_p + (\mathbf{B}_p^{(\ell)})^T\mathbf{X}_{p-1}^{(\ell)}W_{p-1} + \mathbf{B}_{p+1}^{(\ell)}\mathbf{X}_{p+1}^{(\ell)}W_{p+1}\right] \\ &= \psi\left[\left((\mathbf{B}_p^{(\ell)})^T\mathbf{B}_p^{(\ell)} + \mathbf{B}_{p+1}^{(\ell)}(\mathbf{B}_{p+1}^{(\ell)})^T\right)\mathbf{X}_p^{(\ell)}W_p + (\mathbf{B}_p^{(\ell)})^T\mathbf{X}_{p-1}^{(\ell)}W_{p-1} + \mathbf{B}_{p+1}^{(\ell)}\mathbf{X}_{p+1}^{(\ell)}W_{p+1}\right]. \end{aligned}$$

Through rearranging terms by collecting those with the same left multiplied matrix, we equivalently have,

$$\mathbf{X}_p^{(\ell+1)} = \psi\left[(\mathbf{B}_p^{(\ell)})^T(\mathbf{B}_p^{(\ell)}\mathbf{X}_p^{(\ell)}W_p + \mathbf{X}_{p-1}^{(\ell)}W_{p-1}) + \mathbf{B}_{p+1}^{(\ell)}((\mathbf{B}_{p+1}^{(\ell)})^T\mathbf{X}_p^{(\ell)}W_p + \mathbf{X}_{p+1}^{(\ell)}W_{p+1})\right].$$

Then, concatenate the matrices $\mathbf{B}_p^{(\ell)}\mathbf{X}_p^{(\ell)}$ with $\mathbf{X}_{p-1}^{(\ell)}$ and similarly concatenate $(\mathbf{B}_{p+1}^{(\ell)})^T\mathbf{X}_p^{(\ell)}$ with $\mathbf{X}_{p+1}^{(\ell)}$ (and vertically concatenate their corresponding weight matrices) to define,

$$\Theta_{p,p-1}^{(\ell)} = \begin{bmatrix} W_p \\ W_{p-1} \end{bmatrix} \quad \text{and} \quad \Theta_{p,p+1}^{(\ell)} = \begin{bmatrix} W_p \\ W_{p+1} \end{bmatrix}.$$

The resulting message passing layer is,

$$\begin{aligned} \mathbf{X}_p^{(\ell+1)} &= \psi\left((\mathbf{B}_p^{(\ell)})^T\left[\mathbf{B}_p^{(\ell)}\mathbf{X}_p^{(\ell)}, \mathbf{X}_{p-1}^{(\ell)}\right]\Theta_{p,p-1}^{(\ell)} + \mathbf{B}_{p+1}^{(\ell)}\left[(\mathbf{B}_{p+1}^{(\ell)})^T\mathbf{X}_p^{(\ell)}, \mathbf{X}_{p+1}^{(\ell)}\right]\Theta_{p,p+1}^{(\ell)}\right) \\ &= MPSN(\mathbf{L}_p^{(\ell)}, \mathbf{X}_{p-1}^{(\ell)}, \mathbf{X}_p^{(\ell)}, \mathbf{X}_{p+1}^{(\ell)}; \Theta_p^{(\ell)}), \end{aligned} \tag{2.2.4}$$

where [,] denotes matrix concatenation. This is equivalent to Equation 2.2.3. Notably, the message passing scheme in MPSN can also be rewritten in terms of the convolutional layer in SNN, see [11, Appx. C] for proof.

In this construction, there are two learnable weight matrices, $\Theta^{(\ell)}_{p,p-1}$ and $\Theta^{(\ell)}_{p,p+1}$, addressing Property 2.1. Property 2.2, the mixing of features on different dimensional simplices, is also addressed through the use of the Hodge Laplacian operator. The left hand term of Eqn. 2.2.4, $(\mathbf{B}^{(\ell)}_p)^T \left[\mathbf{B}^{(\ell)}_p \mathbf{X}^{(\ell)}_p, \mathbf{X}^{(\ell)}_{p-1}\right]$ maps features on p-simplices to features on $(p-1)$- simplices, mixes dimensions p and $(p-1)$ by concatenation, and then maps them back to features on p-simplices by the left multiplication of $(\mathbf{B}^{(\ell)}_p)^T$. Similarly, the right hand term of Eqn. 2.2.4 does the opposite, mapping features on p-cells up a dimension, mixes dimensions by concatenation, and then maps back down to features on p-simplices by left multiplication by $\mathbf{B}^{(\ell)}_{p+1}$. When split in this fashion, these two terms correspond to the *down-up* and *up-down combinatorial Laplacian operators*. Both of these operators are adjacency-style matrices, which we will use to define pooling on simplicial complexes in Ch. 3.

Separation into two learnable weight parameter matrices allows for greater flexibility in each MPSN layer. The layer could learn to use just co-adjacency information to aggregate features (the left hand term mapping down to $(p-1)$- simplices and back up to p-simplex features), or similarly only use adjacency information (the right hand term mapping up to $(p+1)$-simplices and back down to $(p-1)$-simplex features). However, if both weight matrices learn non-zero parameters, then the layer is using the full p-dim Hodge Laplacian.

2.2.4 Pooling Layers

Pooling layers are an important consideration for neural network architectures, which have implications in training behavior and invariance properties of the model. These layers are designed to reduce the size of the hidden representation of a neural network and are typically paired with convolution layers. Overall, they play an important role in reducing the computational complexity of a model and can limit overfitting. In this section, we discuss pooling layers for CNNs and GNNs, in terms of relevance to invariance properties of the models and application to defining a pooling layer for simplicial neural networks (Ch. 3).

CNN pooling layers In CNN models, pooling layers are paired with convolution layers to reduce the size of the feature map, which is the hidden representation after a layer in the network. Reducing the feature map size consequently reduces the computational complexity of the model because the number of trained parameters is reduced. Commonly used pooling layers for this purpose are max pooling, min pooling, average pooling, and global pooling. Similar to the convolution layers, pooling layers are defined by a filter size, which determines the size of the regions in which to summarize features. For example, a pooling layer with 2×2 filter summarizes features in 2×2 chunks of the hidden representation, and as long as the stride is set to 1 in this case, the size of the output feature map would be half the size of the input hidden representation. Max pooling summarizes features within a filter region by replacing it with only the maximum pixel value. Similarly, min pooling replaces each filter region with the minimum pixel value within that region of the hidden representation. Because of this, max pooling is particularly useful when the model is dealing with images with dark backgrounds, because it selects the brighter pixels and ignores the background. The opposite is true for min pool, which is good for images with light backgrounds by selecting the minimum pixel value of the filter region. Average pooling is useful when harsh edges in the image are not as important for classification because it works by taking the average value within each pixel region. This has the effect of smoothing harsh edges within images. Global pooling refers to collapsing each channel of an image down to a single value. This pooling method can be done using any of the previous three local pooling methods (max, min, or average) and results in significantly compressed representations.

GNN pooling layers Pooling layers are interleaved with regular GNN layers in order to reduce the size of graphs. The primary motivation for including these coarsening layers is to reduce computational complexity, but pooling is also important to limit overfitting in the model. Additionally, pooling layers speed up the message passing between vertices that are far apart on the graph (where distance is measured by the shortest path distance between two vertices). Typically, the pooling methods for graphs are generalizations of standard pooling layers for grids (i.e. used in CNNs).

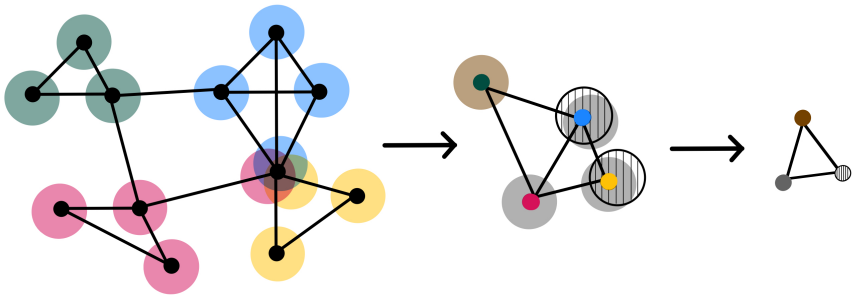


Figure 2.7 High-level illustration of the DiffPool method for graph pooling.
Original graph with vertex cluster assignments → pooled graph after 1 layer with new vertex cluster assignments → pooled graph after 2 layers.

Approaches for this task generally fall into two categories: clustering methods (e.g. DiffPool [96], MinCutPool [7]) and sorting methods (e.g. TopK Pool [33, 15, 49], SAGpool [55]). Additionally, Structural Deep Graph Mapper [9] is a more topologically motivated pooling method based on soft cluster assignments. It uses differentiable and fixed PageRank-based lens functions for the standard Mapper algorithm [82].

We focus on the widely used DiffPool method [96], which uses a hierarchical clustering scheme based on soft cluster assignments of vertices. For DiffPool, a cluster assignment matrix is learned at each layer and is used to coarsen the graph, which is the input for the subsequent layer. Figure 2.7 shows a high-level illustration of this graph pooling for two layers. Instead of directly clustering graph vertices, the vertex cluster assignment matrix $S^{(\ell)}$ is learned through a GNN using the adjacency and feature matrices. The entries of this cluster assignment matrix are learned probability values that are applied to adjacency matrices to coarsen the graph, and applied to feature vectors to update the features on vertices to their coarsened embeddings. At each layer, this is achieved using two separate GNN layers with learnable parameter matrices $\Theta_e^{(\ell)}$ and $\Theta_p^{(\ell)}$. For the first GNN, there is an embedding neural network which outputs $Z^{(\ell)}$, the learned vertex embeddings at layer ℓ,

$$\begin{aligned} Z^{(\ell)} &= GNN_{\ell,embed}(A^{(\ell)}, X^{(\ell)}; \Theta^{(\ell)}_{embed}) \\ &= \psi(AX^{(\ell)}\Theta^{(\ell)}_{neigh,embed} + X^{(\ell)}\Theta^{(\ell)}_{self,embed}) \end{aligned}$$

The other GNN is the pooling neural network which outputs $S^{(\ell)}$, the learned vertex cluster assignment matrix at layer ℓ; $S^{(\ell)}$ is a soft assignment of each vertex at layer ℓ to a meta-vertex in

the next coarsened layer $(\ell + 1)$,

$$\begin{aligned} S^{(\ell)} &= softmax\left[GNN_{\ell,pool}(A^{(\ell)}, X^{(\ell)}; \Theta^{(\ell)}_{pool})\right] \\ &= softmax\left[\psi(AX^{(\ell)}\Theta^{(\ell)}_{neigh,pool} + X^{(\ell)}\Theta^{(\ell)}_{self,pool})\right] . \end{aligned}$$

Then, using $S^{(\ell)}$ and $Z^{(\ell)}$, the graph is pooled by the following matrix operations:

$$\begin{aligned} X^{(\ell+1)} &= \left(S^{(\ell)}\right)^T Z^{(\ell)} \\ A^{(\ell+1)} &= \left(S^{(\ell)}\right)^T A^{(\ell)} S^{(\ell)} \end{aligned}$$

The result of applying $S^{(\ell)}$ to the adjacency matrix is a new, coarsened adjacency matrix $A^{(\ell+1)}$. Similarly, we get the corresponding matrix of vertex embeddings $X^{(\ell+1)}$ for each of the meta-vertices in the coarsened graph. The entire pooled graph is determined by its adjacency matrix and feature matrix, which can be used as input to subsequent layers in the network.

2.2.5 Invariant & Equivariant Representations

Invariance and equivariance, generally, refer to properties that a function can have with respect to transformations of their input data and the output representations. For a transformation T, a function $f(x)$ that is **equivariant** with respect to T satisfies,

$$f(T(x)) = T(f(x)) .$$

For the same transformation, we say that the function $f(x)$ is **invariant** with respect to T if

$$f(T(x)) = f(x) .$$

The convolution and pooling operators in CNNs (and GNNs) are important tools to facilitate learned representations that are equivariant and/or invariant. These properties are particularly desirable in the context of classification, where for example, certain transformations of the input object should not change the class label according to the model (invariant) or when transformations of the input object should change the output by an equivalent transformation (equivariant). Figure 2.8 shows examples of each of these properties. In the first example, we show an equivariant function

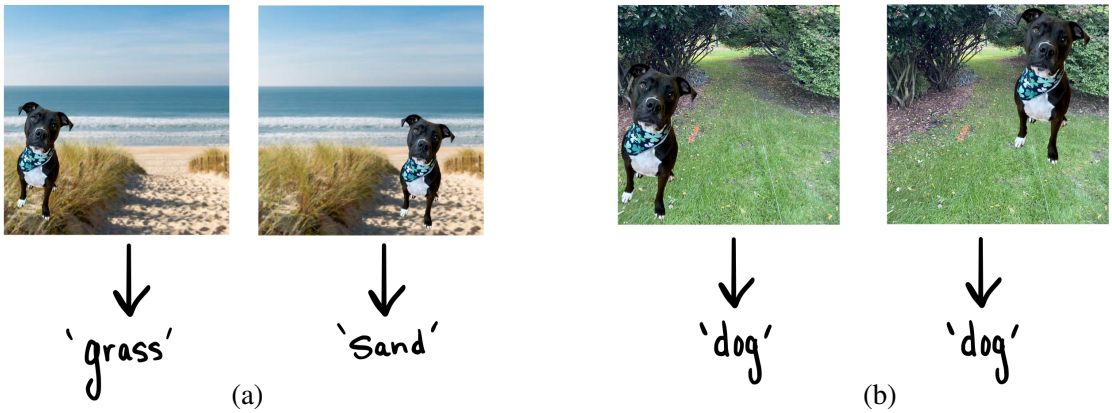


Figure 2.8 Model (a) on the left shows an equivariant function, in which transformation to the input (in this case translation of the animal) equivalently transforms the output classification result from 'grass' to 'sand.' Model (b) on the right shows an invariant function, in which translation to the input image does not affect the model output: 'dog.'

which takes input images and classifies where within the image the animal is sitting. Without any transformations, the model output is 'grass', however when the dog is translated across the image, then the output is "equivalently" translated to be 'sand.' The second example shows model invariance, where now the model task is to classify the type of animal in the image. The original image output is 'dog' and after applying a translation within the image, the classification result should still be 'dog.'

Let us specifically consider the case of translation, letting $x_t(n) = x(n-t)$ be the translation of input object x by translation t and $\Phi(x) \in \mathbb{R}^d$ be a representation of x (for example, the representation of x at the last hidden layer of a CNN). Then, we say that the representation $\Phi(x)$ is **translation equivariant** if

$$\Phi(x_t)(n) = \Phi(x)(n-t) \ .$$

Indeed, this is the case for convolution layers in a CNN due to the definition of convolution operator, which commutes with respect to translation, $T(f * g) = T(f) * g = f * T(g)$. By definition, a convolution layer on its own is translation equivariant [35]. This property is an important aspect of the rotation equivariant ECT-CNN described in Ch. 4. The representation $\Phi(x)$ is **translation invariant** if

$$\Phi(x_t) = \Phi(x) \ ,$$

and we say $\Phi(x)$ is translation invariant up to scale 2^W if

$$\|\Phi(x) - \Phi(x_t)\|_2 \leq c \cdot |t| \cdot 2^{-W} \cdot \|x\|_2,$$

where $0 \leq c \leq 1$ is a constant and $|t|$ is the magnitude of translation. This implies local translation invariance because for a small translation t, the representations $\Phi(x)$ and $\Phi(x_t)$ are nearly the same,

$$\|\Phi(x) - \Phi(x_t)\|_2 \leq c \cdot \frac{|t|}{2^W} \cdot \|x\|_2 << 1,$$

where t is small relative to the scale 2^W, i.e. $\frac{|t|}{2^W} << 1$ [41]. There are three commonly used approaches to encode invariance into deep learning architectures: 1) hardcoded equivariant architectures (e.g. convolution layers) 2) pooling layers to average features and 3) data augmentation. While the invariance and equivariance properties of models have notable implications, efforts to measure them often rely on model performance metrics like loss and accuracy, which are influenced by many factors beyond invariance of the model. To address this deficit, a collection of direct measurements of model invariance and equivariance is presented in [53].

CHAPTER 3

NERVEPOOL: A SIMPLICIAL POOLING LAYER

The content of this chapter is largely reproduced from an earlier preprint [64]. While there are minor changes to formatting, the content remains the same.

Design of deep learning architectures for tasks on spaces of topological objects has seen rapid development, fueled by the desire to model higher-order interactions that are naturally occurring in data. In this topological framework, we can consider data structured as simplicial complexes, generalizations of graphs that include higher-dimensional simplices beyond vertices and edges; this structure allows for greater flexibility in modeling higher-order relationships. Concepts from graph signal processing have been generalized for this higher-order network setting, leveraging operators such as Laplacian matrices [80, 77, 79]. Subsequently, there have been many developments regarding deep learning architectures that leverage simplicial (and cell) complex structure. These methods have been designed through the lens of both convolutional neural networks (CNNs) [29, 95, 14, 94, 76, 45] and message passing neural networks [11, 10]. For the purposes of this dissertation, we refer to neural networks within this general family as simplicial neural networks, and note that our work could be adapted and used for pooling within any of these architectures.

Typically, the pooling methods for graphs are generalizations of standard pooling layers for grids (i.e. those used in CNNs). As opposed to CNNs, which rely on a natural notion of spacial locality in the data to apply convolutions in local sections, the non-regular structure of graphs makes the spacial locality of graph pooling less obvious [96]. For this reason, many graph pooling layers for graph neural networks (GNNs) rely on local structural information represented by adjacency matrices, and coarsening operations applied directly on local graph neighborhoods. A recent survey of GNN pooling methods proposed a general framework for the operations which define different pooling layers: selection, reduction, and connection (SRC) [37]. Selection refers to the method in which vertices of the input graph are grouped into clusters; reduction computation is the aggregation of vertices into meta-vertices (and correspondingly aggregating features on vertices); connection refers to determining adjacency information of the meta-vertices and outputting the

pooled graph. This broad categorization of graph pooling methods into three computational steps provides a useful framework which can be extended in a natural way to describe simplicial complex pooling.

In the simplicial complex domain, the notion of spatial locality necessary for pooling is further complicated due to the inclusion of higher-dimensional simplices. This task of coarsening simplicial complexes requires additional considerations to those of the graph coarsening task, primarily to ensure that the pooled representation upholds the definition of a simplicial complex. Naturally, simplicial complex coarsening shares some challenges with the task of coarsening graph structured data: lack of inherent spacial locality and differing input sizes (varying numbers of nodes and edges). However, the additional challenge for coarsening in the simplicial complex setting is the addition of higher dimensional simplices, with face (and coface) relations in both dimension directions by the definition of a simplicial complex. There is also a notable computational challenge when dealing with simplicial complexes due to the inherent size expansion with the addition of higher-dimensional simplices. A simplicial complex with all possible simplices included is essentially the power set of its vertices. Thus, controlling the computational explosion when using simplicial complexes in deep learning frameworks motivates the use of a pooling layer defined on the space of simplicial complexes. These pooling layers can be interleaved with regular simplicial neural network layers in order to reduce the size of the simplicial complex, which reduces computational complexity of the model and can also limit overfitting.

Existing graph pooling approaches include cluster-based methods (e.g. DiffPool [96], MinCutPool [7]) and sorting methods (e.g. TopK Pool [33, 15, 49], SAGpool [55]). There are also topologically motivated graph pooling methods such as Structural Deep Graph Mapper [9], which is based on soft cluster assignments. It uses differentiable and fixed PageRank-based lens functions for the standard Mapper algorithm [82]. Additionally, there is a method for graph pooling which uses maximal cliques [59], assigning vertices to meta-vertices using topological information encoded in the graph structure. However, if directly generalized for simplicial complexes, this clique-based coarsening method never retains any higher dimensional simplices since the pooled output is always

a graph. Recently, general strategies for pooling simplicial complexes were proposed [21], which directly generalize different graph pooling methods to act on simplicial complexes: max pooling, TopK Pool [33, 15], SAGpool [55], and a TopK method separated by Hodge Laplacian terms. This proposed framework aims to generalize graph pooling methods for simplicial complexes, albeit using different tools from what we use here.

In this chapter we introduce NervePool, which extends existing methods for graph pooling to be defined on the full simplicial complex using standard tools from combinatorial topology. While in practice, the pooling operations are computed via a series of matrix operations, NervePool has a compatible topological formulation which is based on unions of stars of simplices and a nerve complex construction. Like graph pooling methods, the NervePool layer for simplicial complexes can also be categorized by the SRC graph pooling framework [37], with necessary extensions for higher dimensional simplices. The main contributions of this work are as follows:

- We propose a learned coarsening method for simplicial complexes called NervePool, which can be used within neural networks defined on the space of simplicial complexes, for any standard choice of graph pooling on the vertices.
- Under the assumption of a hard partition of vertices, we prove that NervePool maintains invariance properties of pooling layers necessary for simplicial complex pooling in neural networks, as well as additional properties to maintain simplicial complex structure after pooling.

3.1 NervePool Method

In this section, we describe the proposed pooling layer defined on the space of simplicial complexes for an input simplicial complex and vertex clustering. We define both a topologically motivated framework for NervePool (Section 3.1.1) and the equivalent matrix representation of this method (described in Section 3.1.2). In Fig. 3.1, we visually depict both the topological and matrix formulations, and their compatible output for an example simplicial complex and vertex clustering. Notation and descriptions used for simplicial complexes are outlined in Table 3.1.

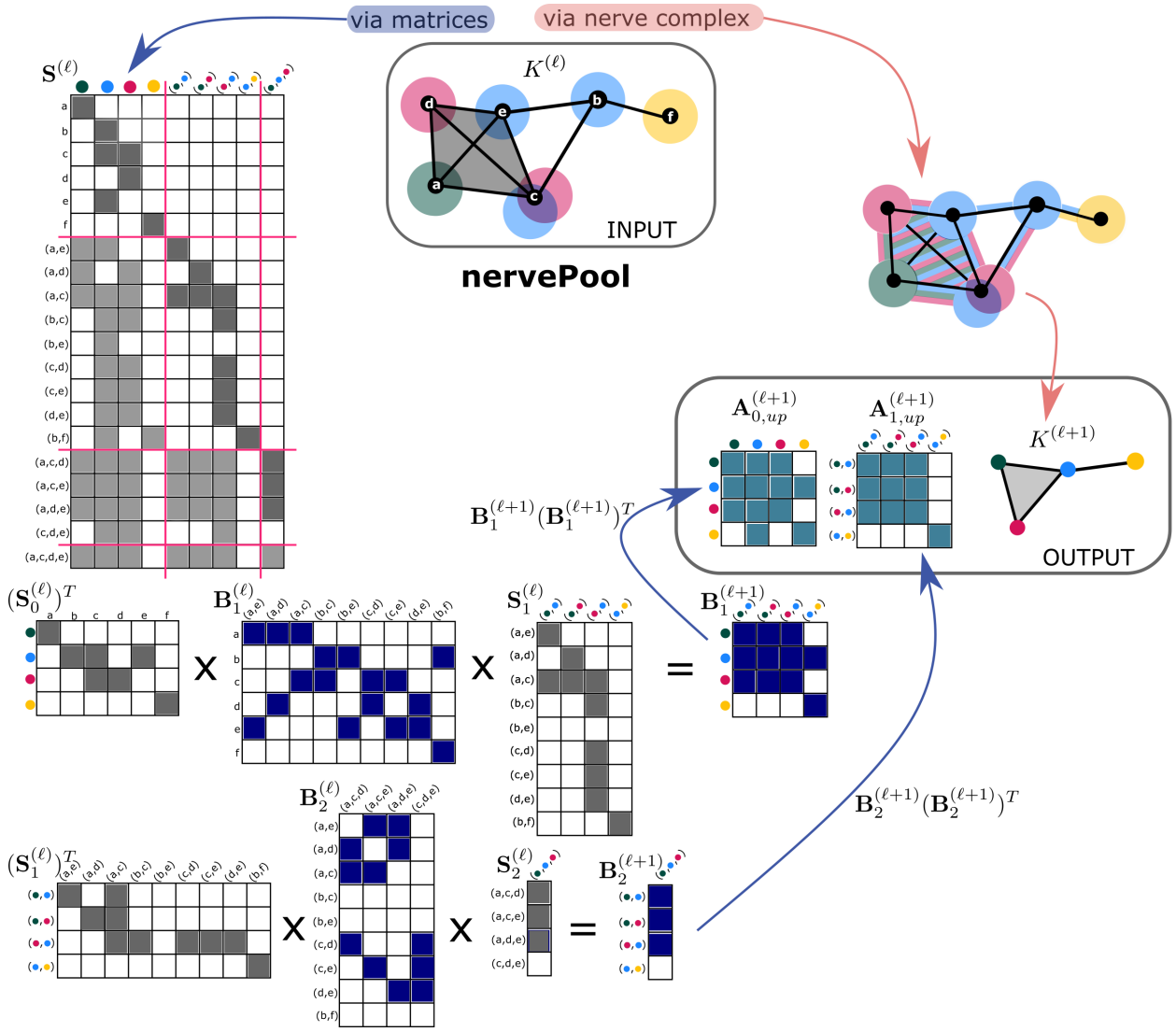


Figure 3.1 A visual representation of NERVEPOOL for an example simplicial complex and choice of **soft partition of vertices**. Shaded-in matrix entries indicate non-zero values, and elsewhere are zero-valued entries. Darker grey entry shading within the $\mathbf{S}^{(\ell)}$ matrix indicate the diagonal sub-blocks which are used for subsequent pooling operations. The output simplicial complex given by the matrix implementation is equivalent to the nerve complex output, up to weighting of p-simplices and the addition of "self-loops", as indicated by non-zero entries on the diagonal of adjacency matrices.

Notation	Description
$K^{(\ell)}$	A simplicial complex at layer ℓ
$n_p^{(\ell)}$	Number of p-dim simplices in $K^{(\ell)}$
$\mathcal{P}^{(\ell)}$	$\dim\left(K^{(\ell)}\right)$
$\mathcal{N}^{(\ell)} = \sum_{p=0}^{\mathcal{P}^{(\ell)}} n_p^{(\ell)}$	Total number of simplices for $K^{(\ell)}$
$d_p^{(\ell)}$	Number of features on p-dim simplices

Table 3.1 Simplicial complex notation and descriptions.

The input to NervePool is a simplicial complex and a learned partition of the vertices (i.e. a specified cover on the vertex set). For exposition purposes, we assume that the initial clusters can form a soft partition of the vertex set, meaning every vertex is assigned to at least one cluster but vertices can have membership in more than one cluster. However, in Section 3.2, theoretical proof of some properties require restriction to the setting of a hard partition of the vertex set. The initial vertex clusters give us a natural way to coarsen the underlying graph (1-skeleton) of the input simplicial complex, where clusters of vertices in the original complex are represented by meta-vertices in the pooled complex. However, in order to collapse both structural and attributed information for higher-dimensional simplices, we require a method to extend the clusters. NervePool provides a mechanism in which to naturally extend graph pooling methods to apply on higher-dimensional simplices.

3.1.1 Topological Formulation

Before defining the matrix implementation of simplicial pooling, we will first describe the topological formulation using the nerve complex. NervePool follows an intuitive process built on learned vertex cluster assignments which is extended to higher dimensional simplices in a deterministic fashion.

From the input cover on just the vertex set, however, we lack cluster assignments for the higher dimensional simplices. To define a coarsening scheme on the entire complex, we must extend the cover such that each of the simplices is included in at least one cluster using a notion of local neighborhoods around each simplex. A standard way to define local neighborhoods in a simplicial complex is the star of a simplex, which can be defined on simplices of any dimension.

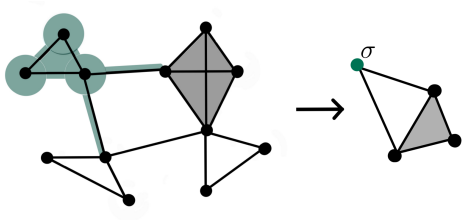


Figure 3.2 The three green highlighted vertices come from a single cover element U_i of a given example cover; the full cover example will be continued in Fig. 3.3. The extended cover $\widetilde{U}_i$ for this collection of vertices, defined by the union of the star of every vertex in the cluster, is shown by the highlighted green simplices. These simplices in $K^{(\ell)}$ all contribute information to the meta vertex $\sigma \in K^{(\ell+1)}$.

For NervePool, we only require the star defined on vertices, $St(v) = \{\sigma_p \in K^{(\ell)} \mid v \subseteq \sigma_p\}$, which is the set of all simplices in $K^{(\ell)}$ such that v is a vertex of σ_p. Using this construction, for each given vertex cluster $U_i = \{v_0, \ldots, v_N\}$, we extend it to the union of the stars of all vertices in the cluster,

$$\widetilde{U}_i = \bigcup_{v \in U_i} St(v)\,.$$

The resulting cover of the complex, $\mathcal{U} = \{\widetilde{U}_i\}$, is such that all simplices in $K^{(\ell)}$ are part of at least (but often more than) one cover element $\widetilde{U}_i$. Figure 3.2 shows an example of a cluster consisting of three vertices and the simplices $\widetilde{U}_i$ which contribute to $\sigma \in K^{(\ell+1)}$. Note that neither $St(v)$ nor $\widetilde{U}_i$ are simplicial complexes, because they are not closed under the face relation.

We use this cover of the complex to construct a new complex, using the nerve. Given any cover $\mathcal{A} = \{A_i\}_{i \in I}$, the nerve is defined to be the simplicial complex given by

$$\mathrm{Nrv}(\mathcal{A}) = \left\{\sigma \subseteq I \mid \bigcap_{i \in \sigma} A_i \neq \emptyset\right\},$$

where each distinct cover element A_i is represented as a vertex in the nerve. If there is at least one simplex in two distinct cover elements A_i and A_j, we add corresponding edge (A_i, A_j) in the nerve complex. Similarly, for higher dimensions if there exists a p-way intersection of distinct cover elements, a $(p-1)$-dimensional simplex is added to the nerve complex. This general nerve construction gives us a tool to take the extended cover of the simplicial complex $\mathcal{U} = \{\widetilde{U}_i\}$ and

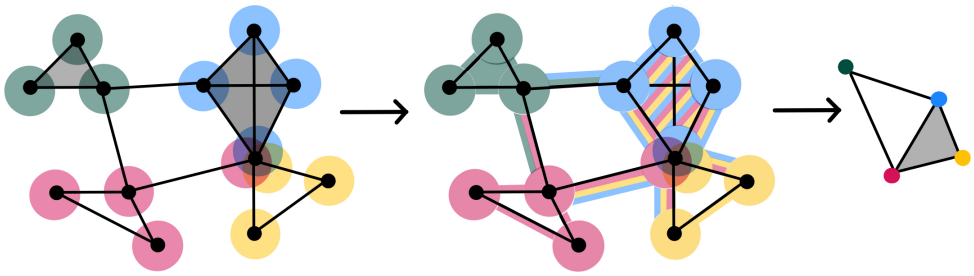


Figure 3.3 Illustration of NervePool on an example 3-dim simplicial complex, applied to coarsen the complex into a 2-dimensional simplicial complex. The left-most complex is the input simplicial complex, with vertex cluster membership indicated by color. The center complex depicts the extended clusters $\mathcal{U} = \{\widetilde{U}_i\}$ and the right-most complex is the pooled simplicial complex, determined by $Nrv(\mathcal{U})$.

define a new, pooled simplicial complex:

$$K^{(\ell+1)} := \mathrm{Nrv}(\mathcal{U}) \, .$$

Figure 3.3 shows the nerve of a 3-dimensional simplicial complex using four predetermined clusters of the vertices.

A notable feature of the extended covers that facilitate NervePool is that they are based on the learned vertex cluster assignments and as such, cover elements are not necessarily contractible. Vertex clusters that are spatially separated on the complex can result in cover elements that are non-convex. Thus, since open cover elements are not guaranteed to be convex, we cannot guarantee that the simplicial complex $K^{(\ell+1)} = \mathrm{Nrv}(\mathcal{U})$ is homotopy-equivalent to the original complex $K^{(\ell)}$ as would be needed to apply the nerve lemma [89].

3.1.2 Matrix Implementation

In practice, we apply the simplicial pooling method described above through matrix operations on constructed cluster assignment matrices, boundary matrices, and simplex feature matrices. Matrix notation used to define simplicial pooling is outlined in Table 3.2. See also Table 3.1 for simplicial complex notation.

Notation	Dimension	Purpose
$\mathbf{A}_p^{(\ell)}$	$n_p^{(\ell)} \times n_p^{(\ell)}$	Adjacency matrix for p-dim simplices at layer ℓ
$\mathbf{B}_p^{(\ell)}$	$n_{p-1}^{(\ell)} \times n_p^{(\ell)}$	p-dimensional boundary matrix. Maps features on p-dim simplices to the space of $(p-1)$-dim simplices
$\mathbf{X}_p^{(\ell)}$	$n_p^{(\ell)} \times d_p^{(\ell)}$	Features on p-dim simplices at layer ℓ
$\mathbf{S}^{(\ell)}$	$\mathcal{N}^{(\ell)} \times \mathcal{N}^{(\ell+1)}$	Block matrix for pooling simplices at layer ℓ
$\mathbf{S}_{q,p}^{(\ell)}$	$n_q^{(\ell)} \times n_p^{(\ell+1)}$	Sub-block of $\mathbf{S}^{(\ell)}$ matrix which maps q-simplices in $K^{(\ell)}$ to p-simplices in the pooled complex $K^{(\ell+1)}$. If $q = p$, we write $\mathbf{S}_q^{(\ell)}$

Table 3.2 Matrix notation with dimensions.

For this chapter, we assume that the input boundary matrices that represent each simplicial complex are non-oriented, meaning the matrix values are all non-negative. Since the nerve complex is an inherently non-oriented object (built using intersections of sets of simplices) and NervePool is formulated based on a nerve construction, it is necessary that the boundary matrices we use to represent each simplicial complex similarly do not take orientation into account; we use $\left|\mathbf{B}_p\right|$ for NervePool pooling operations. However, if the other layers of the neural network that the pooling layer is used within require orientation of simplices, it is sufficient to fix an arbitrary orientation (so long as it is consistent between dimensions of the complex). For example, using an orientation equivariant message passing simplicial neural network (MPSN) layer, we can fix an orientation of simplices after applying NervePool without affecting full network invariance with respect to orientation transformations [11].

Extend vertex cluster assignments From the input vertex cover, we represent cluster membership of each 0-simplex (vertex) in an assignment matrix $\mathbf{S}_0^{(\ell)}$. This $n_0^{(\ell)} \times n_0^{(\ell+1)}$ matrix keeps track of the vertex clusters with entries $\mathbf{S}_0^{(\ell)}[\sigma_0^{(\ell)}, U_i] > 0$ if the vertex $\sigma_0^{(\ell)}$ is in cluster U_i and zero entries elsewhere. Each row of $\mathbf{S}_0^{(\ell)}$ corresponds to a vertex in the original complex, and each column corresponds to a cover element (or cluster), represented as a meta-vertex in the next layer $\ell + 1$. Using these initial clusters which consist of vertices only, we then extend the cluster assignments to include all of the higher dimensional simplices of the complex in a deterministic way. From the original matrix $\mathbf{S}_0^{(\ell)} : n_0^{(\ell)} \times n_0^{(\ell+1)}$, we extend the pooling to all simplices resulting in the full

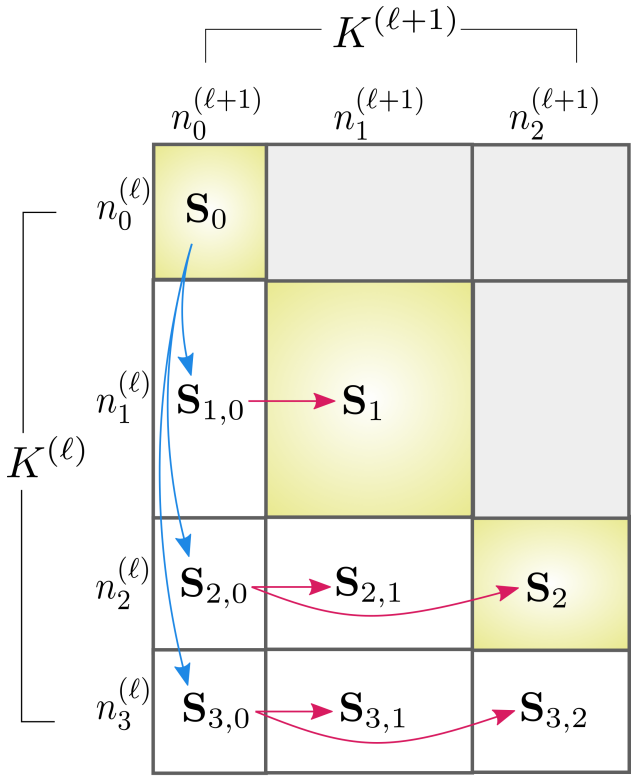


Figure 3.4 Visual depiction of $\mathbf{S}^{(\ell)} : \mathcal{N}^{(\ell)} \times \mathcal{N}^{(\ell+1)}$ block matrix. Sets of sub-blocks are used to map simplices of original simplicial complex to the pooled simplices. Diagonal sub-blocks (highlighted in yellow) are used directly for pooling.

matrix, $\mathbf{S}^{(\ell)} : \mathcal{N}^{(\ell)} \times \mathcal{N}^{(\ell+1)}$, where each of its sub-blocks $\mathbf{S}^{(\ell)}_{q,p} : n^{(\ell)}_q \times n^{(\ell+1)}_p$ map q-simplices in $K^{(\ell)}$ to p-simplices in the coarsened complex $K^{(\ell+1)}$. Figure 3.4 shows a visualization of this matrix extension from vertex cluster assignments to all dimension simplex cluster assignments. As visualized in this diagram, there are two directions by which information is cascaded through the matrix $\mathbf{S}^{(\ell)}$: extend *down* and extend *right*, both of which pass information to simplices of the next higher dimension. Only the diagonal sub-blocks of $\mathbf{S}^{(\ell)}$ are used to pool simplices downstream, so we limit the information cascading to the down and right updates, and ignore possible upper triangular entries. We pass pooling information to simplices of the next higher dimension, within the original simplicial complex via the *down arrow update*, $\mathbf{S}_0 \rightsquigarrow \mathbf{S}_{q,0}$, for $q > 0$ defined,

$$\mathbf{S}_{q,0}[\sigma_q, v_j] := \begin{cases} 1 & \text{if } \mathbf{S}_0[v_m, v_j] = 1 \text{ for any } v_m \in \sigma_q \\ 0 & \text{otherwise.} \end{cases} \tag{3.1.1}$$

Pooling information is passed to simplices of higher dimensions for the new simplicial complex via the *right arrow update*, $\mathbf{S}_{q,0} \rightsquigarrow \mathbf{S}_{q,p}$, for $0 < p \leq q$

$$\mathbf{S}_{q,p}[\cdot, \sigma_p] := \mathbf{S}_{q,0}[\cdot, v_a] \odot \mathbf{S}_{q,0}[\cdot, v_b] \odot \cdots \odot \mathbf{S}_{q,0}[\cdot, v_c]\,, \tag{3.1.2}$$

for σ_p the simplex given by $(v_a, v_b, \ldots, v_c)$ and where $\odot$ indicates the pointwise multiplication of matrix columns. If $\mathbf{S}_{q,p}[\cdot, \sigma_p] = \mathbf{0}$ (the zero vector) then $\sigma_p \notin K^{(\ell+1)}$ and that column is not included in $\mathbf{S}_{q,p}$.

Cluster assignment normalization Cluster assignments for all of the p-simplices should be probabilistic, since the underling partition of the vertices are learned cluster assignments. For example, if the vertex clustering method used follows that of DiffPool on the underlying graph, the cluster assignment matrix is given by a softmax applied to GNN output. In this scenario, the input cluster assignment matrix contains probabilistic assignments of each vertex to one or more clusters. To achieve this normalization across all dimensions of the complex, we row-normalize the lower triangular matrix $\mathbf{S}^{(\ell)}$. The result of this normalization is that for a given p-simplex in $K^{(\ell)}$, the total contributions to pooled q-simplices in $K^{(\ell+1)}$ sum to 1, where $q \leq p$, i.e. each row of the lower-triangular assignment matrix $\mathbf{S}^{(\ell)}$ sums to 1.

Pool with full cluster assignment matrix Only the diagonal sub-blocks of $\mathbf{S}^{(\ell)}$ are directly used to pool simplices of a complex $K^{(\ell)}$ to the pooled complex $K^{(\ell+1)}$. Sub-blocks of the pooling matrix correspond to mapping p-dimensional simplices in $K^{(\ell)}$ to pooled p-simplices in $K^{(\ell+1)}$. In particular, the p-dimensional boundary matrices of the pooled simplicial complex are computed using

$$\mathbf{B}_p^{(\ell+1)} = \left(\mathbf{S}_{p-1}^{(\ell)}\right)^T \mathbf{B}_p^{(\ell)} \mathbf{S}_p^{(\ell)}.$$

Then, the pooled simplicial complex is entirely determined by the upper adjacency matrices (Equation 2.1.1), $\mathbf{A}_{up,p}^{(\ell+1)} = \mathbf{B}_{p+1}^{(\ell+1)} \left(\mathbf{B}_{p+1}^{(\ell+1)}\right)^T$, either on their own or normalized using the absolute difference with the degree matrix $\mathbf{D}_p = \sum_{j=0}^{n_p} B_{p+1}$,

$$\mathbf{A}_{up,p}^{(\ell+1)} = \left| \mathbf{D}_p - \mathbf{B}_{p+1}^{(\ell+1)} \left(\mathbf{B}_{p+1}^{(\ell+1)}\right)^T \right|.$$

The set of pooled boundary matrices, or alternatively the set of upper adjacency matrices, give all of the structural information necessary to define a simplicial complex. Note that the specific adjacency representation is a choice, and one could use alternative notions of adjacency described in Section 2.1.4 to summarize the pooled simplicial complex. Simplex embeddings are also aggregated according to the cluster assignments in $\mathbf{S}^{(\ell)}$ to generate coarsened feature matrices for simplices in layer $\ell + 1$. We compute these pooled embeddings for features on p-simplices by setting

$$\mathbf{X}_p^{(\ell+1)} = (\mathbf{S}_p^{(\ell)})^T \mathbf{X}_p^{(\ell)}, \tag{3.1.3}$$

where $\mathbf{X}_p^{(\ell)}$ are the features on p-simplices. The coarsened boundary matrices and coarsened feature matrices for each dimension can then be used as input to subsequent layers in the network.

The matrix implementation of NervePool, using cluster assignment matrices multiplied against boundary matrices, is consistent with the set-theoretical topological nerve construction outlined in Section 3.1.1. Given the same complex and initial vertex cover, the output pooled complex using the nerve construction is structurally equivalent to the simplicial complex described by $\mathbf{A}_{up,p}^{(\ell+1)}$. Refer to Theorem 3.2.1 and associated proof for a formal description of this equivalence.

3.1.3 Note on differentiability

For training of a simplicial neural network, and specifically to apply NervePool layers, the computations defining NervePool need not be continuous and differentiable. Gradients are not defined on the higher-order simplices of each complex, only on the underlying graph. So, to use gradient descent and perform backpropagation for network training, the simplicial complex is simply recomputed after the learned vertex pooling stage using the NervePool method. Equations 3.1.1 and 3.1.2 facilitate this deterministic computation of coarsening for the higher dimensional simplices.

3.1.4 Training NervePool

The required input to NervePool is a partition of the vertices of a simplicial complex, learned using only the 1-skeleton (underlying graph) of the complex. Since the learned aspect of NervePool is limited to the vertex cluster assignments ($\mathbf{S}_0^{(\ell)}$) and higher-dimensional cluster assignments

Graph Pooling	**Select**	**Reduce** ($\forall\, p$)	**Connect** ($\forall\, p$)
MinCutPool [7]	$\mathbf{S}_0^{(\ell)} = MLP(\mathbf{X}_0^{(\ell)})$	$\mathbf{X}_p^{(\ell+1)} = (\mathbf{S}_p^{(\ell)})^T \mathbf{X}_p^{(\ell)}$	$B_p^{(\ell+1)} = (\mathbf{S}_{p-1}^{(\ell)})^T \mathbf{B}_p \mathbf{S}_p$
NFM [3]	Factorize: $\mathbf{A}_0^{(\ell)} = \mathbf{WH} \rightarrow \mathbf{S}_0^{(\ell)} = \mathbf{H}^T$	$\mathbf{X}_p^{(\ell+1)} = (\mathbf{S}_p^{(\ell)})^T \mathbf{X}_p^{(\ell)}$	$B_p^{(\ell+1)} = (\mathbf{S}_{p-1}^{(\ell)})^T \mathbf{B}_p \mathbf{S}_p$
LaPool [68]	$\begin{cases} \mathbf{V} = \lVert \mathbf{L_0 X}_0 \rVert_d \\ \mathbf{i} = \{i \mid \forall j \in \mathcal{N}(i) : \mathbf{V}_i \geq \mathbf{V}_j\} \\ \mathbf{S}_0^{(\ell)} = \text{SparseMax}\left(\beta \frac{\mathbf{x}_0 \mathbf{x}_{0,i}^T}{\lVert \mathbf{x}_0 \rVert \lVert \mathbf{x}_{0,i} \rVert}\right) \end{cases}$	$\mathbf{X}_p^{(\ell+1)} = (\mathbf{S}_p^{(\ell)})^T \mathbf{X}_p^{(\ell)}$	$B_p^{(\ell+1)} = (\mathbf{S}_{p-1}^{(\ell)})^T \mathbf{B}_p \mathbf{S}_p$

Table 3.3 Trainable Graph Pooling Methods in the SRC framework, with necessary adjustments for higher-dimensional use within NervePool. MLP is a multi-layer perceptron defined on the vertex features, β a regularization vector, and $\mathbf{i}$ a vector of indices. Table adapted from [37].

are a deterministic function of $\mathbf{S}_0^{(\ell)}$, the loss functions used for cluster assignments are defined on only vertices and edges of the complex. Any trainable graph pooling method that learns a soft partition of the graph (DiffPool [96], MinCutPool [7], StructPool [98], etc) can be used in conjunction with NervePool. However, there are minor adjustments necessary to modify these graph pooling methods for application to NervePool simplicial complex coarsening (primarily in terms of computing embeddings for multiple dimensions).

The broad categorization of graph pooling methods into selection, reduction, and connection (SRC) computations [37] provides a useful framework to classify steps of NervePool and necessary extensions for the higher dimensional simplices. The selection step refers to the method for grouping vertices of the input graph into clusters, which can be used directly in the case of NervePool on the 1-skeleton for the initial vertex covers. The reduction computation, which aggregates vertices into meta-vertices (and correspondingly features on vertices aggregated), must be generalized and computed for every dimension of the simplicial complex. In the case of NervePool, this is done using the diagonal sub-blocks of $\mathbf{S}^{(\ell)}$ to perform the reduce step at each dimension. Similarly, the connect step requires generalization to higher dimensions and computation of new boundary maps to output the pooled simplicial complex. Table 3.3 summarizes trainable graph pooling methods, modified for use within NervePool simplicial complex pooling as the learned-vertex clustering input method.

Select	$\mathbf{S}_0^{(\ell)} = softmax[MPSN_{\ell,pool}(\mathbf{B}_1^{(\ell)}, \mathbf{X}_0^{(\ell)}, \mathbf{X}_1^{(\ell)})]$
Reduce ($\forall p$)	$\mathbf{X}_p^{(\ell+1)} = (\mathbf{S}_p^{(\ell)})^T \cdot MPSN_{\ell,embed}(\mathbf{B}_p^{(\ell)}, \mathbf{B}_{p+1}^{(\ell)}, \mathbf{X}_{p-1}^{(\ell)}, \mathbf{X}_p^{(\ell)}, \mathbf{X}_{p+1}^{(\ell)})$
Connect ($\forall p$)	$\mathbf{B}_p^{(\ell+1)} = (\mathbf{S}_{p-1}^{(\ell)})^T \mathbf{B}_p \mathbf{S}_p$

Table 3.4 DiffPool adjustment for NervePool in the SRC [37] framework.

NervePool extended using DiffPool graph pooling For DiffPool [96] graph pooling, the vertex clusters are learned using a GNN, so for the simplicial complex extension, $\mathbf{S}_0^{(\ell)}$ can be learned with an MPSN layer [11]. Using diagonal block matrices of the extended matrix $\mathbf{S}^{(\ell)}$, the reduce step is applied for every dimension of the complex. Finally, the connections of the pooled complex are determined by multiplying cluster assignment matrices against boundary matrices, for each dimension of the complex. NervePool (specifically using DiffPool for vertex cluster assignments) is summarized in Table 3.4 in terms of the necessary select, reduce, and connect steps. Additionally, Fig. 3.5 shows an overview flowchart of NervePool, as implemented via matrices, when the choice of vertex clusters is based on DiffPool-style graph pooling. To extend DiffPool on graphs to simplicial complex coarsening, we must use separate MPSNs to pool features on simplices for each dimension and an additional MPSN to find a soft partition of vertices. These separate message passing networks (*embedding MPSNs* and the *pooling MPSN*) require distinct learnable parameter matrices $\Theta_{p,embed}^{(\ell)}$ and $\Theta_{pool}^{(\ell)}$.

For each pooling layer, a collection of *embedding MPSNs* take as input the features on simplices (for the current dimension $\mathbf{X}_p^{(\ell)}$, a dimension lower $\mathbf{X}_{p-1}^{(\ell)}$, and a dimension higher $\mathbf{X}_{p+1}^{(\ell)}$) and boundary matrices (for the current dimension $\mathbf{B}_p^{(\ell)}$ and one dimension higher $\mathbf{B}_{p+1}^{(\ell)}$). The output of which is an embedding matrix $\mathbf{Z}_p^{(\ell)} \in \mathbb{R}^{n_p^{(\ell)} \times d_p^{(\ell)}}$ for every dim $0 \leq p \leq \mathcal{P}^{(\ell)}$ of the complex $K^{(\ell)}$. Here, $n_p^{(\ell)}$ is the number of p-simplices at layer ℓ, and $d_p^{(\ell)}$ is the number of p-simplex features. These matrices are the learned p-simplex embeddings defined by

$$\mathbf{Z}_p^{(\ell)} = MPSN_{\ell,embed}(\mathbf{B}_p^{(\ell)}, \mathbf{B}_{p+1}^{(\ell)}\mathbf{X}_{p-1}^{(\ell)}, \mathbf{X}_p^{(\ell)}, \mathbf{X}_{p+1}^{(\ell)}; \Theta_{p,embed}^{(\ell)}),$$

where $\Theta_{p,embed}^{(\ell)}$ is the matrix of learnable weights at layer ℓ. Note that for the *pooling MPSN*, which is restricted to dimension $p = 0$, the message passing reduces to a function

$$MPSN_{\ell,pool}(\mathbf{B}_1^{(\ell)}, \mathbf{X}_0^{(\ell)}, \mathbf{X}_1^{(\ell)}).$$

The second component of NervePool, illustrated by the right-side branch in Fig. 3.5, determines the pooling structure through cluster assignments for each simplex. For an input simplicial complex, $K^{(\ell)}$, with features on p-simplices, $\{\mathbf{X}_p^{(\ell)}\}_{p=0}^{\mathcal{P}^{(\ell)}}$, we use an MPSN on the underlying graph of the simplicial complex. In the same fashion as the DiffPool method for graph pooling (Equation 5 in [96]), we generate an assignment matrix for vertices (vertex cover) via

$$\mathbf{S}_0^{(\ell)} = softmax\left[MPSN_{\ell,pool}(\mathbf{B}_1^{(\ell)}, \mathbf{X}_0^{(\ell)}, \mathbf{X}_1^{(\ell)}; \Theta_{pool}^{(\ell)})\right],$$

where the output $\mathbf{S}_0^{(\ell)} \in \mathbb{R}^{n_0^{(\ell)} \times n_0^{(\ell+1)}}$ is the learned vertex cluster assignment matrix at layer ℓ. Note that the MPSN for dimension $p = 0$ acting on the 1-skeleton of $K^{(\ell)}$, reduces to a standard GNN. Learning the pooling assignment matrix requires only this single MPSN, for dimension $p = 0$ simplices. From this single pooling MPSN, $\mathbf{S}_0^{(\ell)}$ gives a soft assignment of each vertex at layer ℓ to a cluster of vertices in the next coarsened layer $\ell + 1$.

In terms of network training, the DiffPool learning scheme for $\mathbf{S}_0^{(\ell)}$ uses two additional loss terms added to the normal supervised loss for the entire complex. These loss terms are added from each NervePool layer to the simplicial complex classification loss during training of the entire network to encourage local pooling of the simplicial complex and to limit overlapping cluster assignments. The first term is a link prediction loss, which encourages clustering of vertices which are spatially nearby on the simplicial complex,

$$\mathcal{L}_{LP} = ||\mathbf{A}_0^{(\ell)}, \mathbf{S}_0^{(\ell)}(\mathbf{S}_0^{(\ell)})^T||_F,$$

where $||\cdot||_F$ denotes the Frobenius norm. The second term is a cross entropy loss, which is used to limit the number of different clusters a single vertex can be assigned to, given by

$$\mathcal{L}_E = \frac{1}{n_0^{(\ell)}} \sum_{i=1}^{n_0^{(\ell)}} H(\mathbf{S}_0^{(\ell)}(i, \cdot)),$$

where H is the entropy function and $\mathbf{S}_0^{(\ell)}(i, \cdot)$ the i-th row of the vertex cluster assignment matrix at layer ℓ. Both of the additional loss terms $\mathcal{L}_{LP}$ and $\mathcal{L}_E$ are minimized and for each NervePool layer, added to the total simplicial complex classification loss at the end of the network. Training a

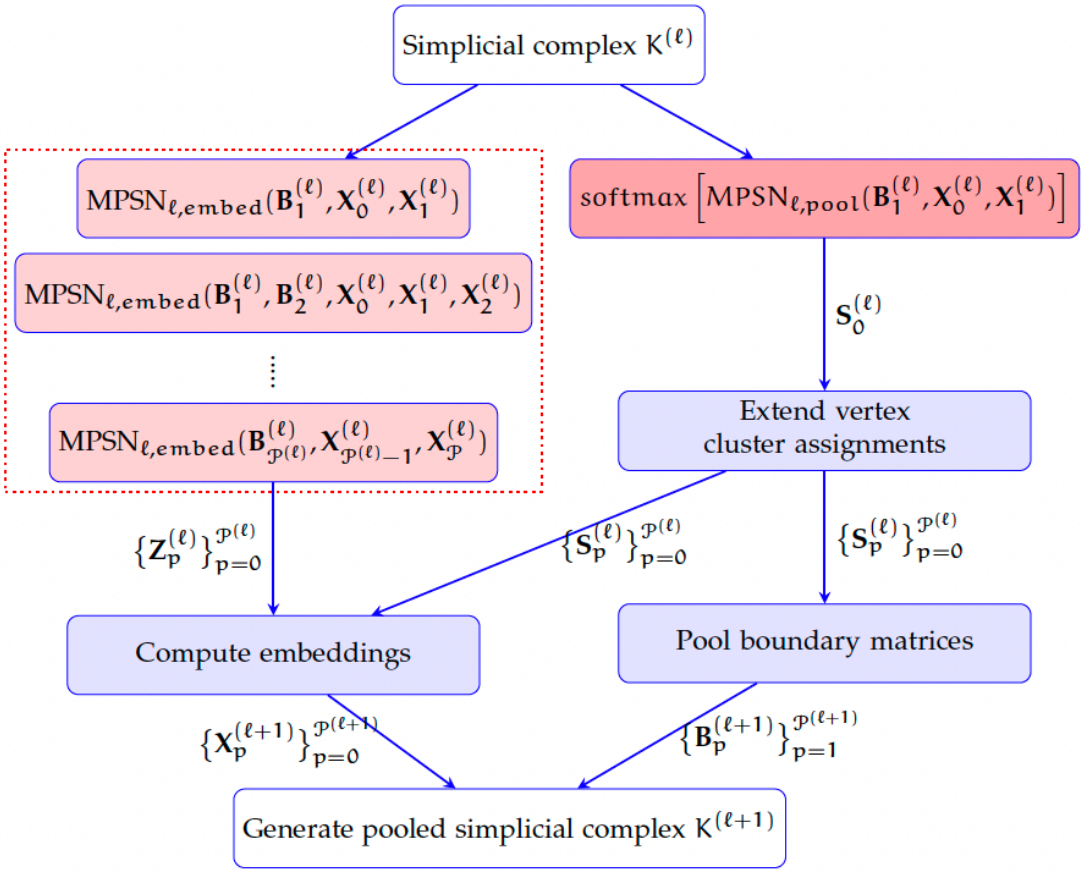


Figure 3.5 Example NervePool architecture, using DiffPool as the motivating vertex cluster method. Takes input simplicial complex $K^{(\ell)}$ and returns the pooled simplicial complex $K^{(\ell+1)}$. The left-side branch uses a collection of MPSNs to compute embeddings for each dimension (Reduce step). The right-side branch illustrates using an MPSN to compute vertex cluster assignments (Select step), and then extending assignments to higher dimensional simplices so that this structural information facilitates collapsing of simplices when applied against boundary matrices (Connect step).

NervePool layer reduces to learning a soft partition of vertices of a graph, since the extension to coarsening of higher-dimensional simplices are deterministic functions of the vertex clusters.

3.2 NervePool Properties

In this section, we describe observational and theoretical properties of NervePool as a learned simplicial complex coarsening method, including a formally defined identity function. In practice, most learned graph pooling methods output soft partitions of the vertex set, meaning a vertex can

be included in multiple clusters. However, extraneous non-zero entries in the "boundary" matrices[1] built by NervePool emerge from vertices assigned to multiple clusters. While the result is a matrix that does not in fact represent a simplicial complex, they do not seem to pose an issue in practice, so long as the output simplicial complex is represented by its adjacency matrices, and not the "boundary" matrices. The unintended entries of "boundary" matrices do not affect the adjacency matrix representation because if one is present in the matrix, then there must also be non-zero entries present for the edges that appear in the adjacency matrix.

As a result, in practice NervePool can be used in a hard- or soft-partition setting. However to prove equivalence of the nerve construction to the matrix implementation and simplex-permutation invariance, we must assume a hard vertex clustering, i.e. where each vertex is a member of exactly one cluster. In this regime, NervePool has more theoretical justification and we can derive the matrix implementation in a way such that it produces the same pooled simplicial complex output as the nerve construction, provided they are given the same input complex and vertex cover. Additionally, assuming a hard partition of the vertex set allows us to prove it is a simplex permutation invariant pooling layer. To work in this more restricted setting, given a soft-partition input, one could simply threshold appropriately to ensure that each vertex is assigned to a single cluster, choosing the cluster it is included in with the highest probability. See the example of Fig. 3.6 for the resulting matrices in the case of a hard partition, which uses the same complex as the soft partition example of Fig. 3.1. In this restricted case, the boundary matrix output of NervePool does not have any extra non-zero entries that we see in the case of soft partitions on the vertex set.

We note one technical distinction between the standard graph representation as an adjacency matrix, and the generalized adjacency matrices we use here. In the standard literature, the diagonal of the adjacency matrix of a graph will be zero unless self loops are allowed. However, the adjacency matrices constructed here have a slightly different viewpoint. In particular (at least in the hard partition setting), an entry of an upper-adjacency matrix for dimension p simplices is

[1] We use quotes around boundary because the matrices generated by NervePool are not necessarily true boundary matrices. They are boundary matrices with possible extra non-zero entries which do not affect the output adjacency representations.

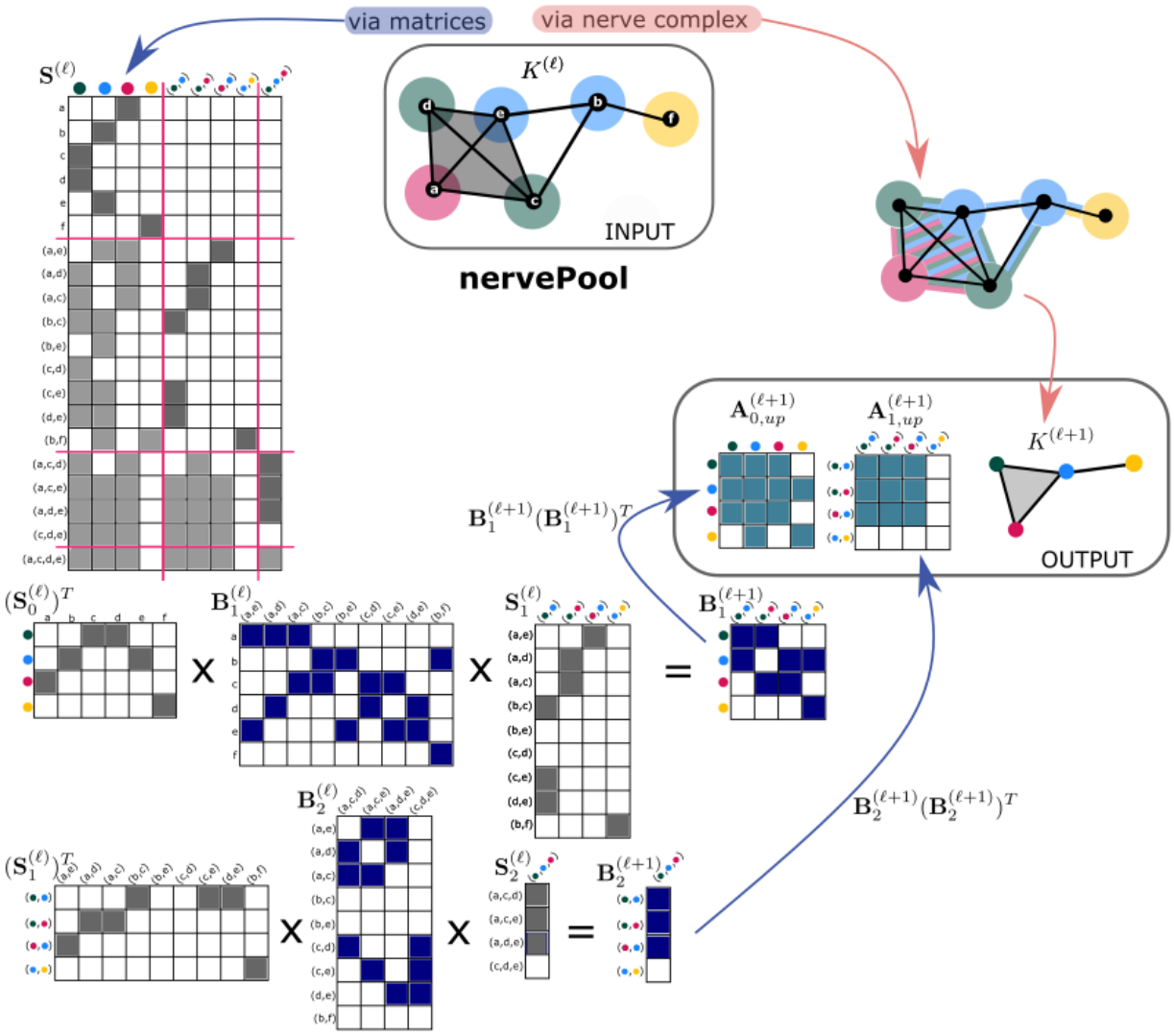


Figure 3.6 A visual representation of NERVEPOOL for an example simplicial complex and choice of vertex clusters. Note this is the same simplicial complex as Fig. 3.1, but using a **hard partition of the vertex set**. Matrix entries indicate non-zero values, and elsewhere are zero-valued entries. Darker grey entry shading within the $\mathbf{S}^{(\ell)}$ matrix indicate the diagonal sub-blocks which are used for subsequent pooling operations.

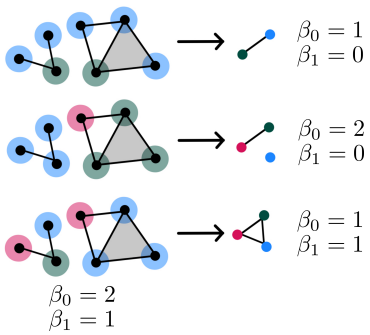


Figure 3.7 An input simplicial complex with three different choices of initial vertex covers (top, middle, bottom). Each of these cover choices produce pooled simplicial complexes of different homology using NervePool, as indicated by the Betti numbers for dimensions 0 and 1. The first initial vertex cover (top) produces NervePool output which changes Betti numbers in both dimensions. The second cover (middle) produces an output complex with the same 0-dim Betti number. The third cover (bottom) produces an output complex with the same 1-dim Betti number.

non-zero if the two simplices share a common higher dimensional coface. In the case of a graph, the 0-dimensional upper-adjacency matrix would then have non-zero entries on the diagonal since a vertex that is adjacent to any edge is thus upper adjacent to itself.

Another important technical note is that with no assumptions on the input clustering, we cannot make promises about maintaining the topological structure of the simplicial complex heading into the pooled layer. Since the vertex cluster assignments are largely task-dependent, any collection of vertices can be grouped together in a cluster if it minimizes the loss function with respect to a given task, with no regard for their spatial locality on the simplicial complex. This can lead to potentially un-intuitive behaviour in the simplex pooling for those used to working in the setting of the nerve lemma [89], specifically learned vertex groupings which are not localized clusters. For example, Fig. 3.7 shows three different initial vertex covers of same input simplicial complex, resulting in three different NervePool outcomes of simplicial complexes with different homology.

In addition to interpretability of the diagonal entries, the hard partition assumption gives further credence for connecting the topological nerve/cover viewpoint with the matrix implementation presented. We prove this connection in the next theorem, which hinges on the following construction. Assume we are given the input cover of the vertex set $\{U_i\}_{i=1}^{n_0^{(\ell+1)}}$, have constructed the cover of the simplicial complex using the stars of the vertices $\mathcal{U} = \{\widetilde{U}_i\}_{i=1}^{n_0^{(\ell+1)}}$ and have built the pooled

simplicial complex using the nerve construction $K^{(\ell+1)} = \text{Nrv}(\mathcal{U})$. There is a natural simplicial map $f : K^{(\ell)} \to K^{(\ell+1)}$ where a vertex $v \in K^{(\ell)}$ is mapped to the vertex in $K^{(\ell+1)}$ representing the (unique due to the hard partition assumption) cover element containing it. As with any simplicial map, simplices in $K^{(\ell)}$ may be mapped to simplices in $K^{(\ell+1)}$ of a strictly lower dimension. Our assumption of using only the block diagonal of the $\mathbf{S}^{(\ell)}$ matrix means that the matrix construction version will essentially ignore these lower dimensional maps, instead focusing on the portions of the simplicial map that maintain dimension. That being said, one could make some different choices in the pipeline to maintain these simplices with the trade-off of additional computation time required.

Theorem 3.2.1 (Equivalence of Topological and Matrix Formulations)**.** Given the same input simplicial complex and hard partition of the vertex set (cluster assignments), the topological nerve/cover viewpoint and matrix implementation of NervePool produce the same pooled simplicial complex.

Proof. The proof that these two methods result in the same simplicial complex is broken into two parts:

(i) **The maps are the same.** We will show that the matrix representation $\mathbf{M}$ of the natural simplicial map $f : K^{(\ell)} \to \text{Nrv}(\mathcal{U}) = K^{(\ell+1)}$ discussed above, is equal to the matrix $\mathbf{S}^{(\ell)}$ of simplex cluster assignments used for pooling in the matrix formulation.

(ii) **The complexes (boundary matrices) are the same.** The boundary matrices which represent the nerve of the cover $\mathcal{U}$ are equal to the boundary matrices computed using the boundary and cluster assignment matrices from layer ℓ. That is, we will show

$$\mathbf{B}_{\text{Nrv}(\mathcal{U}),p} = \left(\mathbf{S}_{p-1}^{(\ell)}\right)^T \mathbf{B}_p^{(\ell)} \mathbf{S}_p^{(\ell)}$$

and thus represent the same complex which we can denote as $K^{(\ell+1)}$.

Proof of (i): We prove equivalence of the simplicial map $f : K^{(\ell)} \to K^{(\ell+1)}$ to the matrix $\mathbf{S}^{(\ell)}$ of simplex cluster assignments by showing equivalent sparsity patterns of their matrices. The simplicial map is defined as $f(\sigma) = \{f(v)\}_{v \in \sigma}$, removing duplicates where they occur and viewing the result as a simplex in $K^{(\ell+1)}$. Writing U_v for the (unique because of the hard cover

assumption) cluster containing vertex v (and noting that there could be different representatives), let $\{U_{v_0}, \cdots, U_{v_p}\}$. Let $\mathbf{M}$ be the matrix that represents this operator, so by definition, $\mathbf{M}[\sigma, \tau] \geq 1$ iff $\tau \leq f(\sigma)$, i.e. if τ is a face of $f(\sigma)$. By the nerve construction, for cover elements U_{v_i} representing vertices of τ, $\bigcap_{i=0}^{p} \widetilde{U}_{v_i} \neq \emptyset$.

By definition of Eqns. 3.1.1 and 3.1.2, to fill in entries of the $\mathbf{S}^{(\ell)}$ matrix, an entry $\mathbf{S}[\sigma, \tau]$ is nonempty iff $\mathbf{S}[\sigma, U_v] \neq 0$ for all $U_v \in \tau$. By definition, this means that σ has at least one vertex in each of the cover elements U_v, and thus $\sigma \in \bigcap \widetilde{U}_{v_i}$. From the previous discussion, this happens iff $M[\sigma, \tau] \geq 1$.

Proof of (ii): Assume we are given a pair $\tau_{p-1} \leq \tau_p$ simplices in $\mathrm{Nrv}(\mathcal{U})$, so that

$$\mathbf{B}_{Nrv(\mathcal{U}),p}[\tau_{p-1}, \tau_p] \neq 0.$$

We will first show that if this occurs, the matrix multiplication results in a nonzero entry, $\mathbf{B}_p^{(\ell+1)}[\tau_{p-1}, \tau_p] \neq 0$, as well. If $\{U_{v_0}, \cdots, U_{v_p}\}$ are the cover elements representing vertices of τ_p, we know that there exists a $\sigma \in \bigcap_{i=0}^{p} \widetilde{U}_{v_i}$. By the star construction, this implies that every U_{v_i} must contain at least one vertex of σ, and because of the hard cluster assumption, we then have $\dim(\sigma) \geq \dim(\tau_p) = p$. Let $\sigma_p \leq \sigma$ be a face of σ with exactly one vertex per cover element, and note that $\sigma_p \in \bigcap_{i=0}^{p} \widetilde{U}_{v_i}$ as well. Finally, writing $\tau_{p-1} = \{U_{v_0}, \cdots, \widehat{U}_{v_i}, \cdots, U_{v_p}\}$, let $\sigma_{p-1} \leq \sigma_p$ be the face which does not have a vertex in U_{v_i}. As we have assumed non-negative entries in all matrices and because there exist non-zero entries in $\mathbf{S}_{p-1}^{(\ell)T}[\tau_{p-1}, \sigma_{p-1}]$, $\mathbf{B}_p^{(\ell)}[\sigma_{p-1}, \sigma_p]$, and $\mathbf{S}_p^{(\ell)}[\sigma_p, \tau_p]$, the resulting matrix multiplication

$$\mathbf{B}_p^{(\ell+1)} = \left(\mathbf{S}_{p-1}^{(\ell)}\right)^T \mathbf{B}_p^{(\ell)} \mathbf{S}_p^{(\ell)}$$

will also have a non-zero entry in $[\tau_{p-1}, \tau_p]$.

Conversely, if the pair τ_{p-1} is not a face of τ_p, there is at least one vertex U_{v_1} in τ_{p-1} which is not in τ_p, and using the dimension difference, there are at least two vertices U_{v_2}, U_{v_3} which are in τ_p but not τ_{p-1}. Given any $\sigma_p \in K^{(\ell)}$, we will show that at least one of $\mathbf{S}_{p-1}^{(\ell)T}[\tau_{p-1}, \sigma_p]$, and $\mathbf{S}_p^{(\ell)}[\sigma_p, \tau_p]$ are zero, resulting in a zero entry in the matrix multiplication. If $\mathbf{S}_p^{(\ell)}[\sigma_p, \tau_p] = 0$ we are done, so assume this entry is not zero. By definition, this means that $f(\tau_p) = \sigma_p$. Following

the same argument above, this means that σ_p has a vertex in each cover element U_v of τ_p. Given any $\sigma_{p-1} \leq \sigma_p$, there is exactly one vertex U_v missing. However, either U_{v_2} or U_{v_3} must still be included in σ_{p-1}. These were found because they are not in τ_{p-1}, thus $f(\sigma_{p-1}) \neq \tau_{p-1}$. As the $\mathbf{S}^{(\ell)}$ matrix represents the simplicial map f, we then have that $\mathbf{S}_{p-1}^{(\ell)T}[\tau_{p-1}, \sigma_{p-1}] = 0$ as required. □

Theorem 3.2.2 (NervePool identity function)**.** There exists a choice of cover (cluster assignments) on the vertices, $\mathbf{S}_0^{(\ell)}$, such that NervePool is the identity function, i.e. $K^{(\ell+1)} = K^{(\ell)}$ and the pooled complex is equivalent to the original, up to re-weighting of simplices. In particular, this choice of $\mathbf{S}_0^{(\ell)}$ mapping is such that each distinct vertex in $K^{(\ell)}$ is assigned to a distinct vertex cluster in $K^{(\ell+1)}$.

Proof. For the proof, we show a bijection between the simplicial complexes using the set-theoretic representations, thus showing that the resulting layer is the same up to reweighting of the simplicies. Suppose each vertex in the original complex is assigned to a distinct cluster, so that $U_i = \{v_i\}$, and then $\widetilde{U_i} = \mathrm{St}(v_i)$. For any simplex $\sigma = \{v_{i_0}, \cdots, v_{i_d}\}$ in $K^{(\ell)}$, we denote $\widetilde{\sigma} = \{\widetilde{U_{i_0}}, \cdots, \widetilde{U_{i_d}}\}$. Note that $\widetilde{\sigma}$ must be a simplex in $K^{(\ell+1)}$ since σ is in the intersection of the stars $\bigcap_{i=i_0}^{i_d} \mathrm{St}(v_i)$.

Injectivity is immediate by definition. We show the bijectivity of the set map $K^{(\ell)} \to K^{(\ell+1)}$ sending $\sigma \mapsto \widetilde{\sigma}$. To check surjectivity, note that for any $\widetilde{\sigma} \in K^{(\ell+1)}$ with $\{\widetilde{U_{i_0}}, \cdots, \widetilde{U_{i_d}}\}$, the intersection $\bigcap_{i=i_0}^{i_d} \mathrm{St}(v_i)$ must contain some simplex τ. By definition, this means that $v_i \leq \tau$ for all $i \in \{i_0, \cdots i_d\}$. Then $\sigma = \{v_{i_0}, \cdots, v_{i_d}\} \leq \tau$, and by the closure of the simplicial complex, $\sigma \in K^{(\ell)}$. □

Permutation invariance A key property of graph structured data and graph neural networks is that the re-ordering of vertices should not affect the network output. Different representations of graphs by their adjacency information are equivalent, up to arbitrary reordering of the vertices. This property implies a permutation invariant network is necessary for most tasks such as graph classification, and subsequently permutation invariant pooling layers are necessary. Other types of invariance, including orientation of simplices, are not relevant for NervePool, due to the assumption of non-oriented simplices.

Theorem 3.2.3 (NervePool Permutation Invariance)**.** NervePool is a **permutation invariant** function on the input simplicial complex and partition of vertices. Equivalently,

$$\text{NervePool}(P\mathbf{S}_0^{(\ell)}, K^{(\ell)}) = \text{NervePool}(\mathbf{S}_0^{(\ell)}, K^{(\ell)}),$$

for any permutation matrix on vertices $P \in \{0,1\}^{n_0^{(\ell)} \times n_0^{(\ell)}}$.

Proof. Permutations of simplices correspond to re-ordering of the vertex list of a complex. Note that the re-ordering of vertices of a simplicial complex induces a corresponding re-ordering of all its higher dimensional simplices. Consider a simplicial complex defined by its set of simplicies $K := \{\sigma_0, \ldots, \sigma_n\}$. Consider an alternate labeling of all simplices in K and call this new simplicial complex $K' := \{\sigma'_0, \ldots, \sigma'_n\}$. Then, there exists an isomorphism between the sets K and K', and specifically an isomorphism between the vertex sets of each. This map between vertices in K and vertices in K' can be represented by a permutation matrix $P \in \{0,1\}^{n_0^{(\ell)} \times n_0^{(\ell)}}$. Then, $\text{NervePool}(P\mathbf{S}_0^{(\ell)}, K^{(\ell)}) = \text{NervePool}(\mathbf{S}_0^{(\ell)}, K^{(\ell)})$ since the simplex sets define the same simplicial complex structure and the vertex cluster assignments are permuted accordingly. □

3.3 Code and implementation notes

NervePool code is available at `www.github.com/sarah-mcguire/nervePool`. The implementation takes an input simplicial complex (given by either a list of simplices or set of boundary matrices) and partition of vertices (array assignment matrix $\mathbf{S}_0^{(\ell)}$) and returns the pooled simplicial complex. There is also built-in functionality to visualize each simplicial complex using `NetworkX` [38]. In the case of hard clusters of the vertex set for pooling, the complex visualization can also include highlights around each vertex, with color indicating each vertex's cluster membership. The current implementation of NervePool is limited to input simplicial complexes with maximum dimension 3 (where the largest simplices are tetrahedra). Additionally, vertices are labeled and tracked using ASCII characters, so the maximum number of vertices which a simplicial complex can have is 128 in the current version.

3.4 Conclusions and Future Directions

In this chapter, we have defined a method to extend learned graph pooling methods to simplicial complexes. Given a learned partition of vertices, NervePool returns a coarsened and simplified simplicial complex, where simplices and signals defined on those simplices are redistributed on the output complex. This framework for simplicial complex coarsening is a very flexible method because the input partition of the vertex set can come from any choice of standard graph pooling method or learned vertex clustering. We show that there is a choice of input cover on the vertices such that NervePool returns the same simplicial complex (up to re-weighting) and that when used in the context of a simplicial neural network with hard vertex clusters, it is a simplex-permutation invariant layer. Additionally, we prove the equivalence of the nerve/cover topological interpretation and matrix implementation using boundary matrices for the setting restricted to hard vertex partitions. This pooling layer has potential applications in a range of deep learning tasks such as classification and link prediction, in settings where the input data can be naturally modeled as a simplicial complex. NervePool can help to mitigate the additional computation cost of including higher dimensional simplices when using simplicial neural networks: reducing complex dimension, while redistributing information defined on the simplicial complex in a way that is optimized for the given learning task.

A limitation of this method is that using standard graph pooling methods for the initial clustering of vertices limits the learned influence of higher-dimensional simplices. Future work in this direction to include more topological information in the learning of vertex clusters could enhance the utility of this method for topologically motivated tasks. For graph pooling methods with auxiliary loss terms such as DiffPool [96] and MinCutPool [7], adjustment or inclusion of additional auxiliary topological loss terms to encourage vertex clusters with topological meaning would be an interesting line of inquiry. Taking into account the higher-dimensional structure for the learned pooling on underlying graphs could allow NervePool to coarsen simplicial complexes, tuned such that specified topological structure is retained from the original complex.

While the current assumptions of NervePool include loss functions only defined on the 1-

skeleton, if additional auxiliary loss terms defined on higher-order simplices were utilized, then the functions defining NervePool would need to be continuous and differentiable for gradient computations. This is automatic for Eq. 3.1.2, the right arrow update, since it defines a series of Hadamard (entry-wise) products, which are differentiable. However, Eq. 3.1.1 is an “on or off" function, which is not differentiable. To address this, one could define a relaxation of the update rule which closely approximates the rule with a differentiable function (e.g. sharp sigmoidal or capped Relu-type functions). Such considerations for gradient computation could be addressed in future work, adjusting loss function choices to preserve desired topological structure in the pooled representation of simplicial complexes.

In this work, to compute embeddings of features on p-simplices $\{\mathbf{X}_p^{(\ell+1)}\}_{p=0}^{\mathcal{P}^{(\ell+1)}}$, we choose to apply separate coarsening for each dimension of the input simplicial complex in Equation 3.1.3. This choice enforces that information sharing for the signals defined on simplices only affect signals on pooled simplices within a given dimension. Alternatively, future modifications could utilize the entire block matrix $\mathbf{S}^{(\ell)}$ applied against a matrix containing signals on simplices of all dimensions, which would allow for signal information to contribute to pooled simplices amongst different dimensions.

Additional future work on simplicial complex coarsening should include experimental results to identify cases where including simplicial pooling layers improve, for example, classification accuracy. Also, comparison experiments at different layers of a simplicial neural network would be useful to determine if the pooled complexes are reasonable and/or meaningful representations of the original simplicial complexes, with respect to a given task.

CHAPTER 4

A CNN FOR ECT DATA

In this chapter, we propose a neural network architecture which exploits the inherent structure of directional transform data. Directional transform data refers to a class of summary statistics in which topological summaries are computed from a collection of directions around the input data. In this chapter, we restrict the setting to 2D input data and focus on a choice of directional transform with provable properties as a sufficient statistic: the Euler characteristic transform (ECT).

4.1 Introduction

Here, we propose a framework within the overall theme of synthesizing deep learning and TDA to combine directional transform data with convolutional neural networks. In this work, we are interested in deep learning on an input space of topological signatures with spherical structure relating the signatures. The inherent structural information in certain data types is leveraged for both message passing and convolution style neural networks. Convolutional Neural Network (CNN) architectures rely on the grid-like structure of input data such as images (2-D grid) and time series (1-D grid) [35]. GNNs leverage input data structure in a different way, however, since non-Euclidean structure of certain datasets lends them to be modeled as graphs. In this setting, relationships on graphs are defined in terms of adjacency relations instead of Euclidean grids. We propose to apply a variant of the CNN architecture to leverage the structure of directional transform data using topological signatures.

Current methods using directional transforms (see Sec. 2.1.7 for definition) typically aggregate the topological summaries from each direction in a straightforward way (e.g. sum, concatenation, etc) [2, 44]. However, there is additional structure, namely circular directions $\omega \in \mathbb{S}^{d-1}$ inherent to the directional transform, that is lost when the statistics are aggregated and directly applied to traditional machine learning methods (e.g. Support Vector Machine (SVM), k-nearest neighbor algorithm). To leverage the structural relationship between the transforms in different directions, we propose using a CNN architecture that considers the relationship between directions as structured input.

If one wishes to use standard pairwise distance comparisons with directional transform data (and visualize these distances using multidimensional scaling, for example), then the input data must be pre-aligned. While the original PHT work suggested support of the comparison of unaligned shapes because of its property as an injective map [84], previous applications of directional transforms such as those in [2] require aligned input data, or use standard alignment techniques such as Procrustes methods [36] or alignment using classic shape statistics for centering, scaling, and rotating as described in [84]. However by using a CNN to model the data, the assured rotational equivariance of the model circumvents the need to pre-align data. In this chapter, we prove the ECT-CNN pipeline achieves rotation equivariance. This property is particularly useful when there is no canonical choice of alignment for the data, or when doing so comes at great computational cost. In example leaf shape datasets which we will explore in Ch. 5, this is not as necessary; it is reasonable to align most leaf shapes because one can determine the direction of growth from the leaf stem and align accordingly. However, in other biological applications, no such natural alignment is available. For example, one might wish to compute the ECT of protein structures which have no natural orientation. The complexity of proteins, and vast variety amongst all known protein structures make their alignment not only a computational challenge, but a practical one as well. A method which circumvents the preprocesing step of alignment is especially valuable in these scenarios where alignment is unfeasible.

4.2 Related Work

Beyond the applications on simple test data for proof of concept on variations of the ECT, there has been additional efforts to use the ECT for analysis of real data, specifically through methods of machine learning.

Developments using the ECT in this space have included Euler integral related applications [63] and [88] as well as topological loss terms [87, 78]. In their 2021 paper, Amézquita and coauthors use the ECT to quantify the shape of barley seeds [2]. For classification, they concatenate all of the ECT vectors into a high dimensional vector, perform dimensionality reduction and then pass the result into an SVM. Due to the use of an SVM, this method requires that the data be pre-

aligned. SVM models classify d-dimensional vectors by finding a $(d-1)$-dimensional hyperplane which reasonably separates the vectors by the desired class label, maximizing the distance from the hyperplane to the nearest vector on each side. Determining the hyperplane which separates classes requires that the vectors be comparable, and thus dimensions of each vector must maintain a consistent ordering. Additionally, because they concatenate the ECT into a single high dimensional vector, information about which direction each of the ECCs was from is lost in the classification step. In another paper, Jiang and coauthors define a weighted version of the ECT that uses a simplicial complex with an associated weight function as input [44]. They motivate the so-called WECT by brain tumor data analysis, because the data often comes in the form of segmented grayscale images, which could be represented as weighted simplicial complexes in this way. In their experimental results, they apply the WECT to a classification task of MNIST (handwritten digits) and similarly use an SVM model to classify the ECT vectors. In both of these applications, using an SVM for classification necessitates alignment of the data and means that the directional information is not being used for the classification model.

The ECT has also been incorporated into deep learning methods, although not by using it as a static shape descriptor (i.e. as input to the model). In their recent paper, Röell and Rieck use the ECT to define a layer within a neural network and additionally claim it can be used as a loss term [78]. In order for the ECT layer to be updated using backpropogation, it must be differentiable. Thus, they modify the ECT to define a differentiable ECT which is differentiable with respect to the directions chosen and the coordinates of the input data and coin the method the Differentiable Euler Characteristic Transform (DECT). This method is in contrast to what we suggest here because we focus on the use of ECT data as a fixed shape descriptor, and how to leverage the inherent structure of the data to use it as input to deep learning classification methods.

Separate from ECT use in machine learning tasks, there are existing efforts to define CNNs for input data which are not restricted to signals on Euclidean grids. In [47], the authors propose CyCNN, which converts input images with Cartesian coordinates to polar coordinates and uses cylindrical convolution layers for classification. In [22], the authors propose a Spherical CNN,

which applies convolutional neural networks to spherical signals. They define a spherical cross-correlation (for convolution operations) which is rotation-equivariant, replacing filter translations with rotations in the sphere. Using the generalized Fourier transform, which projects a real valued function onto a set of orthogonal basis functions, the translations directly act like spherical harmonics. The Spherical CNN architecture is defined on the space of real-valued functions, however the space of a topological signature is not limited to $\mathbb{R}$; using directional transform data as input to Spherical CNNs would thus require further investigation.

4.3 Method

The novelty of this pipeline relies on being able to retain the directional information which is inherent to the directional transform data when applying it to tasks such as classification. Let $K \in \mathbb{R}^2$ be a simplicial complex, with an embedding in Euclidean space given by the function $g : V(K) \to \mathbb{R}^2$ which assigns geometric coordinates to each vertex in K. By interpolating all higher dimensional simplices from the vertex embeddings, we have an embedding for all simplices $\sigma \in K$ which maps $g : K \to \mathbb{R}^2$. The ECT is defined by the collection of functions for each direction ω,

$$f_\omega : K \to \mathbb{R}$$
$$\sigma \mapsto \max_{v \in \sigma} \langle g(v), \omega \rangle.$$

As opposed to flattening the ECT data into a single, high-dimensional vector, we instead represent it as a matrix with dimensions (D, J), where D is the number of directions and J is the number of thresholds used to compute the ECT. Given a bounding box $[-T, T]$ of K, for fixed, evenly spaced directions $\{\omega_i\}_{i=1}^{D} \in \mathbb{S}^1$ and thresholds denoted by $\{t_j\}_{j=1}^{J}$, where $t_1 = -T$ and $t_J = T$, the ECT matrix has entries

$$M_K[i, j] = \chi(f_{\omega_i}^{-1}(-\infty, t_j)). \tag{4.3.1}$$

This matrix representation resembles a single channel 2D image, which is well-suited for classification tasks using standard CNNs. However, we think of the 2D ECT data as living on a cylinder, $\mathbb{S}^1 \times \mathbb{R}^2$, so some considerations are required to ensure that spatial structure is retained in the image

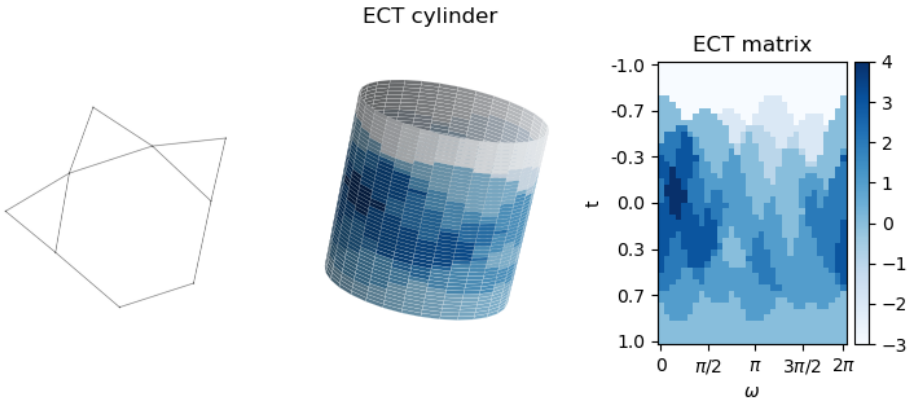


Figure 4.1 An example graph (left) with the computed ECT using 32 directions and 48 thresholds represented as a cylinder (center) and the corresponding 2D matrix representation (right). Heatmap colors in both ECT representations indicate the Euler Characteristic value at a specific threshold t and direction ω.

representation. Figure 4.1 shows an example embedded simplicial complex, its ECT on a cylinder, and the corresponding ECT matrix representation.

Cylinder padding To facilitate the illusion that the input images are cylindrical, we use padding defined by built in methods as part of the PyTorch package. For each of the sides, where we want the left edge to be identified with the right edge of the image, we use circular padding which uses copies of columns from the right side to pad the left and copies of the left side to pad the right. Figure 4.2 depicts this circular padding and subsequent zero padding. On the top and bottom of each image, we pad with zero-valued pixels, which is a standard choice for computer vision tasks. There are a few parameters involved in padding that can be adjusted to have different effects on the output feature map. The padding size, p, which represents how many pixels deep we want to pad our image, is one such parameter. Stride, s, is a parameter that controls how far the kernel is shifted within the image during convolution. If we assume k is the kernel size, or the size of the filter we are using for convolution, then defining the padding size to be

$$p = \lfloor \frac{k}{2} \rfloor$$

has the effect of preserving the feature map size as long as the stride is $s = 1$.

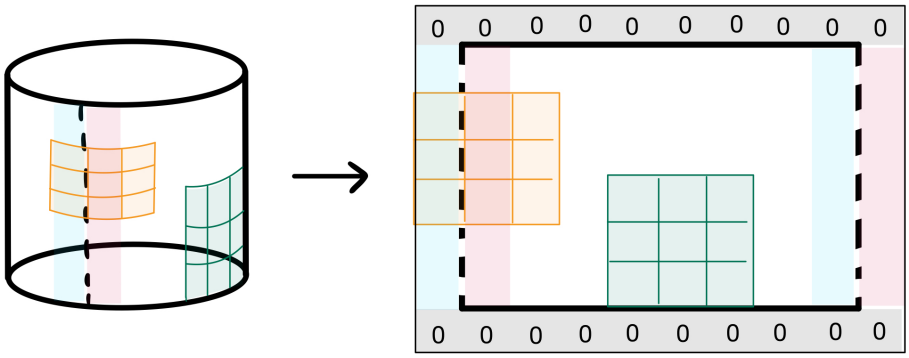


Figure 4.2 Diagram showing the equivalence between rotation on the cylinder and translation on the flattened 2D image representation of the cylinder. Circular padding is used to identify the left and right edges of the 2D image.

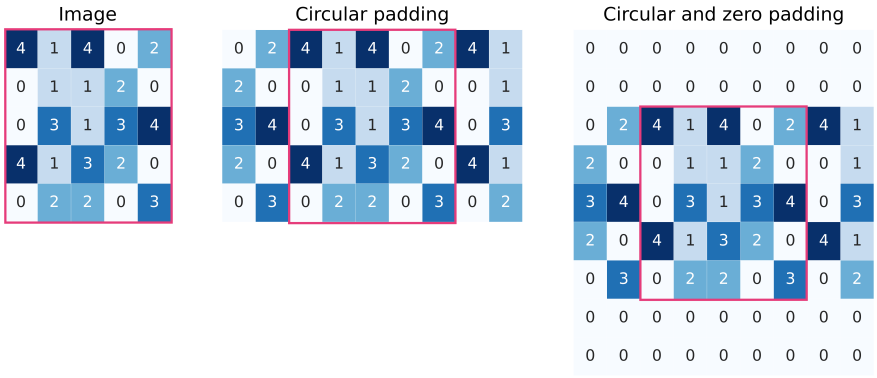


Figure 4.3 Example toy image of size 5×5 with cylinder padding applied. Zero padding is applied to the top and bottom of the image and circular padding is applied to the left and right. Choice of padding size in this example is two, however the choice depends on desired effect on the output feature map size as described in Sec. 5.2.

In this case, the output feature map is the same size as the input to the convolutional layer. Figure 4.3 shows an example of circular and zero padding, with a stride of one and padding size such that the feature map size is the same as the input size. When $s = 1$, the kernel stops at every pixel and is centered there for convolution. When $s > 1$, the feature map shrinks after the convolution step. Reduced feature maps can be useful for dimension reduction and reducing computation time because the filter is applied fewer times. However it has a trade-off and can cause the model to lose information, especially fine-grained details depending on image content.

PyTorch Implementation The input to each of the models we consider here are arrays and vectors of Euler characteristic values; as such, the values can be positive or negative valued integers. For the purposes of CNN models, we need to treat input data as an images, and thus rescale each ECT "image" to have pixel values in $[0, 255]$ which is a standard representation for greyscale image data. We achieve rescaling through,

$$\text{new\_array} = (\text{array} - \min(\text{array})) * \left(\frac{255}{\max(\text{array})\text{-}\min(\text{array})}\right),$$

which allows us to work with the ECT matrices as images in PyTorch and be able to further normalize them without having to account for negative valued pixels. For normalization, we use the following built in PyTorch transformations. Assuming the input image data has pixel values in $[0, 255]$, we use ToTensor() to convert the numpy array to a tensor with pixel values rescaled to $[0, 1]$. Then, we use the built in Normalize($\mu = 0.5, \sigma = 0.5$) function, which assumes tensor input with $[0, 1]$, and choose $\mu = 0.5$ and $\sigma = 0.5$ such that pixel values in each image are normalized to $[-1, 1]$,

$$\text{image} = \frac{\text{image} - \mu}{\sigma}.$$

Additionally, we use a third transform on the image data, introducing random rotations, to reduce bias in the model. The motivating purpose of using this ECT-CNN pipeline for classification analysis is that it is not necessary to pre-align the data. Indeed, to prevent the model from fitting to any incidental alignment in the data, we apply a random rotation to each image before training. In the case of the MPEG-7 dataset, and as is often the case in image-type datasets, there are some classes for which most of the images appear to be aligned in a natural way. However, an arbitrary dataset does not necessarily have a natural choice of alignment between classes of shapes. To prevent any unintentional alignment from influencing the model, and to encourage the training to focus on the shape of the data instead, we apply a random rotation to each image. This is done through the built in PyTorch RandomRotation() transform for the shape images (with pixel fill values set to be the same as the background pixel values to avoid artificial image edges visible after rotation) and using a custom transform for the ECT matrices. Our custom ECT rotation transform

works by first randomly selecting a rotation angle, then correspondingly translating the columns of an existing ECT matrix. The result is a transformed version of the ECT matrix which represents the ECT matrix exactly computed for that rotation of the original complex. This translation transform is useful because it allows an easy way to perform "rotations" of the input pixel-based images without having to worry about considerations to actually rotate a non-vector image file format (centering, alignment, image edges and size, etc).

In the case of the SECT-CNN, we additionally apply a custom transform to the ECT image, to convert the standard ECT matrix to the smooth Euler characteristic transform. The SECT, defined in Sec. 2.1.7, is computed from the ECT matrix by first taking the average of each direction ω (i.e. column of the matrix). Then, the column-wise mean is subtracted from the associated column so that each column is centered. Then, we integrate from the first threshold to the last threshold, which is computed using a cumulative sum from the bottom of the matrix to the top. The resulting matrix is the same size a the ECT, and we similarly normalize following the process described above.

Architecture We use a simple CNN architecture with two convolutional layers paired with max pooling layers, followed by two fully connected layers for classification. This model architecture is depicted in Fig 4.4. Architectural parameter choices such as the number of channels were chosen somewhat arbitrarily, as opposed to being selected to optimize the classification accuracy results. Indeed, we utilize this CNN architecture to evaluate the method of evaluating ECT data in this way, and not with an eye towards achieving state-of-the-art classification results. For specific applications of the ECT-CNN to other datasets, the best architecture parameter choices would vary, and selecting them to optimize model performance could be done through grid-search or other standard methods.

4.4 Translation Equivariance and Invariance

Representing the ECT data as a matrix, with its cylindrical property accounted for using padding in convolutional layers, we are able to use a CNN as a model for classification that is rotational

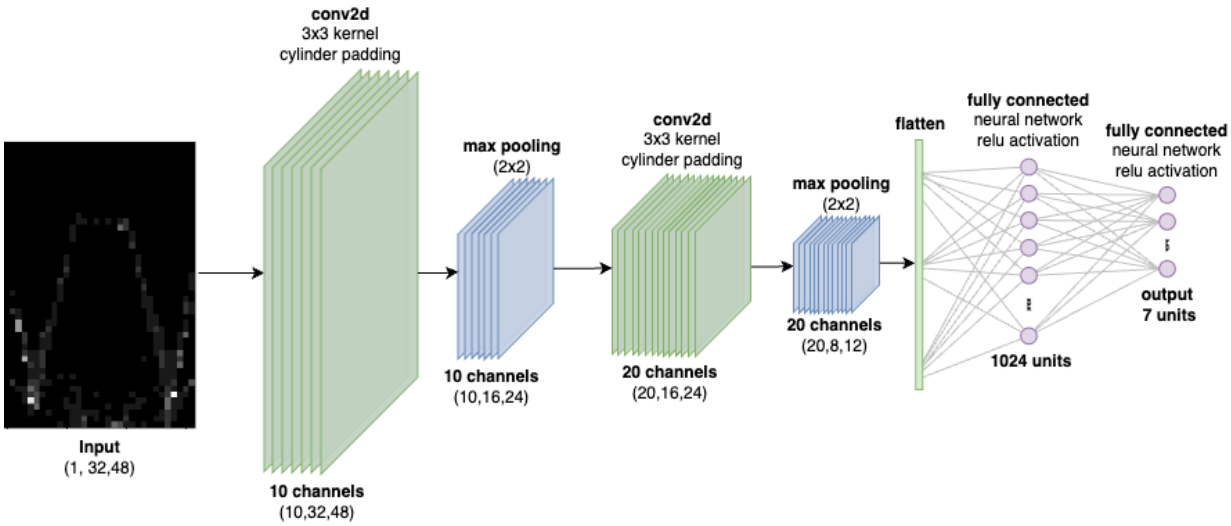


Figure 4.4 CNN architecture used for classification of MPEG7 ECT data consisting of two convolutional layers, each paired with a max pooling layer followed by two fully connected layers.

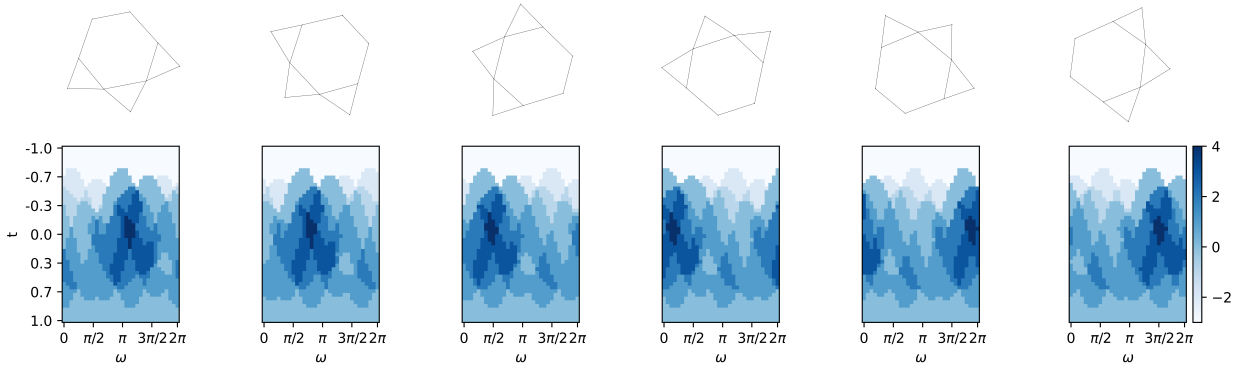


Figure 4.5 An example graph (top) and computed ECT matrix (bottom), recomputed for rotations of the graph. As the input graph is rotated (clockwise from left to right), the values of the ECT matrix are translated, as seen by the left-to-right shift of the patterns within the image.

equivariant. Rotations of the input simplicial complex correspond to translations in the ECT matrix, as shown in Fig. 4.5. As shown in the subsequent lemmas, rotation on the cylinder is the same as translation in the flat-matrix, so we are able to rotate the input shape and the effect is translation in the ECT image on the right, as shown in Fig. 4.5. The ECC for direction $\omega \in \mathbb{S}^1$ of the rotated complex K_θ is equal to the ECC for direction $\omega - \theta$ of the original complex K. We then can show that rotation of the input complex corresponds to horizontal translations in the ECT matrix.

Lemma 4.4.1. Rotation of complex corresponds to translation of ECT For an embedded simplicial complex K and the same complex rotated by θ, K_θ,

$$\mathrm{ECT}(K_\theta)(\omega) = \mathrm{ECT}(K)(\omega - \theta).$$

Proof. Let $K \subseteq \mathbb{R}^2$ be the input simplicial complex with embedding given by $g : V \to \mathbb{R}^2$ Let $\omega \in \mathbb{S}^1$ be a choice of direction in which to rotate the simplicial complex. Then, we can define the rotated simplicial complex by applying the rotation θ to the given embedding function g of K,

$$\begin{aligned} g_\theta : K &\to \mathbb{R}^2 \\ v &\mapsto R_\theta \cdot g(v), \end{aligned}$$

where R_θ is the rotation transformation matrix which operates on a vector to produce the rotated vector for a fixed coordinate axes,

$$R_\theta = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}.$$

We refer this rotated embedded simplicial complex as $K_\theta \subseteq \mathbb{R}^2$ for simplicity later. Consider the Euler Characteristic Transform, $\mathrm{ECT}(K_\theta)$, and specifically the function with respect to direction ω,

$$f_\omega^{K_\theta} : t \mapsto \chi((f_\omega^{K_\theta})^{-1}(-\infty, t]) \ .$$

Consider also the function on the original orientation of K with respect to the direction $\omega - \theta$,

$$f_{\omega-\theta}^{K} : t \mapsto \chi((f_{\omega-\theta}^{K})^{-1}(-\infty, t]) \ .$$

The Euler Characteristic curve, $\mathrm{ECT}(K)(\omega - \theta)$, is defined by

$$\begin{aligned} f_{\omega-\theta} : K &\to \mathbb{R}^2 \\ \sigma &\mapsto \max_{v \in \sigma} \langle g_\theta(v), \omega - \theta \rangle. \end{aligned}$$

Since $g_\theta(v) = R_\theta \cdot g(v)$,

$$f_{\omega-\theta}(\sigma) = \max_{v \in \sigma} \langle R_\theta \cdot g_\theta(v), \omega - \theta \rangle$$

The vectors $\langle R_\theta \cdot g(v), \omega - \theta \rangle$ and $\langle g(v), \omega \rangle$ are equivalent,

$$\begin{aligned}\langle R_\theta \cdot g_\theta(v), \omega - \theta \rangle &= \langle R_\theta^{-1} R_\theta \cdot g_\theta(v), R_\theta^{-1} \cdot (\omega - \theta) \rangle \\ &= \langle g(v), R_\theta^{-1} \cdot (\omega - \theta) \rangle \\ &= \langle g(v), \omega \rangle,\end{aligned}$$

since we can verify that $R_\theta^{-1} \cdot (\omega - \theta) = \omega$:

$$\begin{aligned}R_\theta^{-1} \cdot (\omega - \theta) &= \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} T\cos(\omega - \theta) \\ T\sin(\omega - \theta) \end{pmatrix} \\ &= \begin{pmatrix} T[\cos\theta \cdot \cos(\omega - \theta) - \sin\theta \cdot \sin(\omega - \theta)] \\ T[\sin\theta \cdot \cos(\omega - \theta) + \cos\theta \cdot \sin(\omega - \theta)] \end{pmatrix}\end{aligned}$$

by trig product identities and Ptolemy's difference identities,

$$\begin{aligned}&= \begin{pmatrix} T[\frac{\cos(\theta+\omega-\theta)+\cos(\theta-(\omega-\theta))}{2} + \frac{-\cos(\theta-(\omega-\theta))+\cos(\theta+\omega-\theta)}{2}] \\ T[\sin\theta(\cos\omega \cdot \cos\theta + \sin\omega \cdot \sin\theta) + \cos\theta(\sin\omega \cdot \cos\theta - \cos\omega \cdot \sin\theta)] \end{pmatrix} \\ &= \begin{pmatrix} T[\frac{\cos\omega+\cos\omega}{2}] \\ T[(\sin\theta \cdot \cos\omega \cdot \cos\theta) + (\sin^2\theta \cdot \sin\omega) + (\sin\omega \cdot \cos^2\theta) - (\cos\theta \cdot \cos\omega \cdot \sin\theta)] \end{pmatrix} \\ &= \begin{pmatrix} T\cos\omega \\ T\sin\omega(\sin^2\theta + \cos^2\theta) \end{pmatrix} \\ &= \begin{pmatrix} T\cos\omega \\ T\sin\omega \end{pmatrix} \\ &= \omega.\end{aligned}$$

Then, $f^K_{\omega-\theta} : K \to \mathbb{R}^2$ is equivalent to $f^{K_\theta}_\omega : K_\theta \to \mathbb{R}^2$ and we have that

$$\mathrm{ECT}(K_\theta)(\omega) = \mathrm{ECT}(K)(\omega - \theta).$$

The direction ω relative to K rotated by θ is the same as the direction $\omega - \theta$ relative to K with its original orientation (see example Fig. 4.6). □

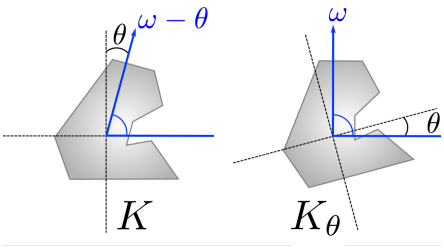


Figure 4.6 Visual depiction of the effect of rotation of shape K by θ relative to a second angle ω. The ECC of K computed with respect to direction $\omega - \theta$ is the same as the ECC of K_θ computed with respect to direction ω.

We have shown that rotations of the input simplicial complex correspond to translations across the collection of functions which encompass the ECT in the continuous setting. However, in practice the ECT requires subsampling of directions on $\mathbb{S}^1$ so computation requires discretization by selecting a finite number of directions and finite number of thresholds to at which to compute the Euler characteristic. It is expected to introduce error through subsampling directions for the ECT computation, and consequently introduce error in the rotation equivariance property of the ECT-CNN model. However, in the the continuous setting by using the Euler Characteristic curve in all possible directions, rotation of the input image exactly corresponds to translation of the ECT matrix. We recall the matrix approximation described in Sec. 4.3 such that the ECT matrix has entries given by Eqn. 4.3.1 and show that input simplicial complex rotation corresponding to ECT translation transfers through the matrix discretizations of the ECT.

Let K be an embedded simplicial complex and M_K be the Euler Characteristic Transform matrix representation of K. If K is rotated by θ, then the resulting ECT matrix representation M_K is equivalent to the ECT matrix representation M_{K_θ} with columns shifted according to the angle θ (i.e. rotation of the input complex corresponds to horizontal translation of the ECT matrix). We assume that the sampled directions are fixed and evenly spaced such that $\Delta\omega = \omega_i - \omega_{i-1}$ for all $i \leq D$, which gets rid of stability issues common with the ECT [62].

Lemma 4.4.2. Matrix representations are equivalent For fixed, evenly spaced directions $\{\omega_i\}_{i=1}^D$ such that $\Delta\omega = \omega_i - \omega_{i-1}$ is the difference between directions, there exists some $k \in \mathbb{Z}$ such that the

rotation of K can be written $\theta = \Delta\omega \cdot k$, and the matrix representations of the ECT are equivalent,

$$M_{K_\theta}[i, j] = M_K[i - k, j].$$

Proof. From the previous lemma, we have that $f^K_{\omega-\theta} = f^{K_\theta}_\omega$ for some rotation θ of K and the ECT computed for directions $\omega - \theta$ and ω, respectively. Let $\omega = \omega_i$, where i denotes the direction index. By the construction of fixed, evenly spaced directions, we can write the rotation angle as a index of the sampled directions, $\theta = \Delta\theta \cdot k$ for some $k \in \mathbb{Z}$. Then, $f^K_{\omega_i-(\Delta\omega\cdot k)} = f^{K_\theta}_{\omega_i}$ and we can write $f^K_{\omega_{i-k}} = f^{K_\theta}_{\omega_i}$. The matrix representations of each respective ECT (each $M \in \mathbb{R}^{D\times J}$ matrices, where D is the number of directions $\omega \in \mathbb{S}^1$ sampled and J is the number of thresholds), have entries given by

$$M_{K_\theta}[i, j] = \chi(f^{-1}_{\omega_i}(-\infty, t_j))$$
$$M_K[i - k, j] = \chi(f^{-1}_{\omega_{i-k}}(-\infty, t_j))$$

Since $f^{K_\theta}_{\omega_i} = f^K_{\omega_{i-k}}$, we have that the matrix representations are equivalent,

$$M_{K_\theta}[i, j] = M_K[i - k, j].$$

The column at index ω in the rotated shape is the same as the column at index $\omega - \theta$ in the original orientation of the shape. Thus, rotation of input complex K corresponds to horizontal translation within M, the matrix representation of the ECT. □

The subsequent corollary follows directly from the previous two lemmas, as an immediate consequence of the ECT matrix construction.

Corollary 4.4.3. Rotation Equivariant ECT-CNN For fixed, evenly spaced directions $\{\omega\}_{i=1}^D$ the 2D ECT-CNN with cylindrical padding is rotation equivariant.

Given a translation equivariant CNN, the ECT-CNN model proposed is equivariant to rotations of the input object. By Lemma 4.4.1, the ECC for direction $\omega \in \mathbb{S}^1$ of the rotated complex K_θ is equal to the ECC for direction $\omega - \theta$ of the original complex K. By Lemma 4.4.2, rotations of the

input shape correspond to horizontal translations within the matrix representation of the ECT and the matrix representations of the ECT are the same. Then, since input object rotations are encoded as translations of the ECT image, and since CNNs are translation equivariant, the ECT-CNN is rotation equivariant.

Translation invariance of CNNs: what we can and cannot say The convolution and pooling operators in CNNs (and GNNs) are important to facilitating learned representations that are equivariant and invariant. These properties are particularly desirable in the context of classification, where for example, certain transformations of the input object should not change the class label according to the model (invariant) or when transformations of the input object should change the output by an equal amount (equivariant). Let us specifically consider the case of translation and recall the definitions for invariance and equivariance presented in Ch. 2.

Definition 4.4.4 (**translation invariance**)**.** Let $x_t(n) = x(n-t)$ be the translation of input object x by translation t and $\Phi(x) \in \mathbb{R}^d$ be a representation of x. Then, $\Phi(x)$ is translation invariant if

$$\Phi(x_t) = \Phi(x).$$

Definition 4.4.5 (**translation equivariance**)**.** Let $x_t(n) = x(n-t)$ be the translation of input object x by translation t and $\Phi(x) \in \mathbb{R}^d$ be a representation of x. Then, $\Phi(x)$ is translation equivariant if

$$\Phi(x_t)(n) = \Phi(x)(n-t).$$

Indeed, translation equivariance is achieved in convolution layers of a CNN due to the definition of the convolution operator, which commutes with respect to translation. By definition of convolution, a convolution layer on its own is translation equivariant [35]. Recall the functional analysis definition of convolution for two functions x and h, defined by reflecting one of the functions about the y-axis and shifting, then taking the integral of the product of the two functions,

$$x * h(u) := \int_{-\infty}^{\infty} x(v)h(u-\tau)dv \ .$$

By this definition, convolution commutes with respect to translation because the choice of which function you reflect and shift does not change the integral result.

However, convolution does not commute with respect to other transformations such as rotation and scaling. This property is particularly important because it means that convolutional layers in a CNN are not inherently rotation- or scale- equivariant. Thus, we cannot expect CNN models to produce invariant or equivariant representations of rotated or re-scaled inputs, but do expect CNNs to provide equivariant (and potentially invariant) representations for translated inputs.

Convolutional layer output is translation equivariant, so features translated in the input signal are equivalently translated in the feature map. Most standard choices of pooling layers also produce equivariant representations, passing the equivariance through from the convolutional layer. However, pooling layers can increase the translation invariance level of the internal representation, depending on choices of filter size for pooling. This occurs because the pooling allows information in the feature maps to be condensed down (through averaging, max function, etc), coarsening the feature map representation, and reducing the relevance of where in the feature map the information is. In the most extreme case, pooling down to a single pixel must yield a translation invariant representation. For smaller pooling filter sizes, however, pooling functions such as MaxPool(), MinPool(), and AvgPool() have the effect of producing equivariant representations, passing the equivariance from the convolutional layers through to pooling layer output. Small filters pool local features of the convolutional layer output, so feature information that is in a representation $\Phi(x)$ is translation invariant if $\Phi(x_t) = \Phi(x)$.

Despite the limited ability to prove translation invariance in our context, and in the context of standard CNNs, it is generally accepted that a well-trained CNN model having convolution layers paired with pooling layers can achieve translation invariance, and this translation invariance is formed during training [8]. However, it is not proved how translation invariance of standard CNNs occurs in practice and the conditions under which a model is considered "well-trained" enough to be approximately translation invariant are not provided. A common method to introduce translation invariance in CNNs is through image augmentations (i.e. additional training instances added of

images with spatial movements). This suggests the need for the CNN to be trained with objects presented at different locations and large, diverse datasets in order to achieve invariance [8, 40, 52]. While in theory, convolution layers guarantee equivariant representations, the closeness to invariant representations achieved in practice pairing pooling layers seems to be limited by other factors of the model including overfitting, data augmentation, and dataset size and diversity.

From Lemma 4.4.1 and subsequent corollary, it seems natural that a desirable CNN model for classification, trained on translated ECT images, should be rotation invariant. That is to say, to avoid pre-aligning data, we would like to have a pipeline designed such that rotations of the input simplicial complex do not affect the output classification. While we do not guarantee translation invariance of the proposed ECT-CNN method, we see model performance results that suggest it is achieved in practice.

In testing of our ECT-CNN method in Sec. 4.5 and Ch. 5, we compare against using a standard CNN to classify image versions of each dataset as a baseline. However, CNNs are not architecturally rotation-invariant; passing un-aligned (randomly rotated) image-versions of our input data into a standard CNN will not produce representations that are invariant to such rotations. For that reason, we do not expect these models to perform well, however we see boosts in accuracy for larger datasets. This is largely because there are more samples of each class, represented at multiple orientations due to random rotations in the data loader, so the size and diversity of the training dataset inflates the model's ability to distinguish between classes for the same reason data augmentation methods would: there is more data to train on and the risk of overfitting to a specific orientation is reduced.

4.5 Application to simple shape dataset

To show the utility of this ECT-CNN method, we compute the ECT and pass the representations into a simple CNN for a shape dataset of images MPEG-7 [81] commonly used for shape classification. Replicating the subset of the dataset used in [54, 84], we restrict to using 7 out of the 70 total classes for classification: 'bone', 'fork', 'fountain', 'glass', 'hammer', 'heart', 'key'. Each of the classes contains 20 samples, resulting in a relatively small dataset for exposition. Samples in this dataset are binary images of varying size, so in order to compute the ECT, we first use pixel

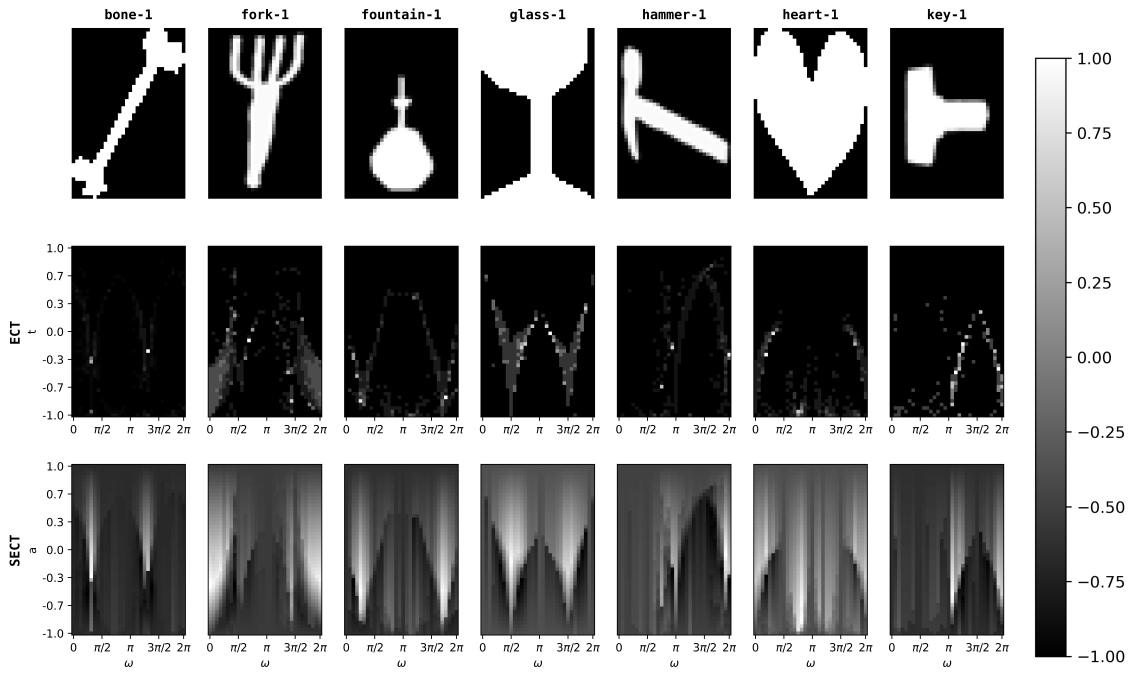


Figure 4.7 Samples of each class used for classification in the MPEG7 dataset: 'bone', 'fork', 'fountain', 'glass', 'hammer', 'heart', 'key' (top), the corresponding ECT images (middle), and corresponding SECT images (bottom). Pixel values in the ECT and SECT images are in $[-1, 1]$ due to the normalization described in Sec. 4.3.

values of the image to represent the array as a cubical complex as described in Sec. 2.1.7. Then, for each binary image, represented as a cubical complex, we compute the Euler Characteristic for 32 directions and using 48 thresholds. ECT data for each sample is represented as a $[1, 48, 32]$ tensor, and passed into the CNN as a 1-channel image. We additionally compute the SECT from each ECT image to use for classification. Recall that the SECT replaces the integer-valued Euler Characteristic Curve with a continuous function, producing a transformed version of the ECT (see Sec. 2.1.7 for its definition). Samples of each image class, along with their corresponding ECT and SECT images are shown in Fig. 4.7.

Classification results Classification results for the ECT-CNN pipeline on MPEG-7 are summarized in Table 4.1, separated into three categories, further described in this section: TDA-based methods, a traditional CNN method, and combining TDA with traditional methods. Each of the reported classification accuracies are computed using 10-fold cross validation to reduce the influence of specific train/test splits on model performance and the standard deviation of the 10 runs is

MPEG-7 Classification accuracy, by input data and model				
	Model	epochs	lr	accuracy±std
TDA	32×48 ECT-CNN	25	10^{-3}	93.57% ± 6.74
	32×48 SECT-CNN	25	10^{-3}	87.86% ± 7.86
	1536×1 ECT SVM	-	-	57.86% ± 13.34
Traditional	32×48 shape image CNN	25	10^{-3}	98.57% ± 2.86
Both	32×48 shape image + ECT-CNN	25	10^{-3}	100.00% ± 0.00

Table 4.1 Summary of classification results on the MPEG-7 dataset for different combinations of input data and model used. The number of epochs and learning rate for each trained model are also noted. Each reported accuracy and standard deviation is computed with 10-fold cross validation.

also reported.

The first section of results in the table shows the various ways we use ECT data for classification. The first model performance presented is the ECT-CNN, which is trained using the regular ECT images and achieves an average test accuracy of 93.57% ± 6.74 over 10-folds. In addition to the standard ECT data, we also transform the ECT image to get the SECT. Without changing any of the model architecture parameters, we train another CNN model using the SECT versions of MPEG-7 training data. The resulting test accuracy on this second model shows a sharp drop from the ECT-CNN accuracy to 87.86% ± 7.86. This behaviour is interesting because the SECT and ECT encode the same information, in the sense that you can derive one representation from the other and vise versa, so we expect that a CNN trained on one data type could perform as well as a CNN performed on the other. A likely explanation for the gap in performance is due to the specific model architecture parameters chosen. Features in the SECT images, because of smoothing, are generally more global and encompass a larger scale of the image than the features in the ECT images. Increasing the convolution filter size in the model would enable it to learn the larger-scale features of the SECT, and thus produce a more accurate model. Indeed, we verify this behavior by training the SECT-CNN model using the same parameters as before, with 10-fold cross validation, but increasing the convolution kernel size. We similarly train the ECT-CNN with varying kernel sizes. Figure 4.8 shows a pattern of the SECT-CNN classification accuracy increasing as we increase the kernel size, however the model performance decreases when the same is applied to

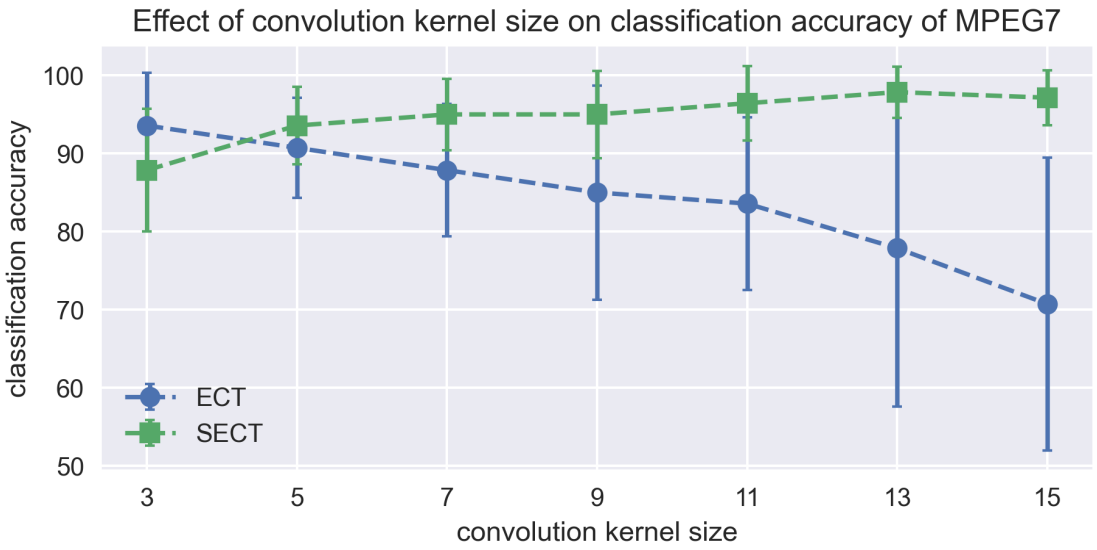


Figure 4.8 Classification accuracy of the ECT-CNN (blue circles) and SECT-CNN (green squares) for varying convolution kernel sizes, each averaged over 10-fold cross validation with the standard deviation of each accuracy result plotted as error bars.

training the ECT-CNN model due to the differences between the ECT and SECT in the scale of structure within the images.

The primary benchmark comparison we consider in this series of model comparisons is classifying the ECT data using a linear SVM model. From the ECT image, we flatten the image into a 1536×1 vector, so each of the Euler Characteristic curves is concatenated to the one from the previous direction. SVM classification does not give us a rotation equivariant pipeline because if you change the orientation of the data, recompute the ECT, and flatten the data into a vector, all association from one direction to the next has been lost. The input to these models is unaligned data, so low classification accuracy using an SVM is expected. As expected, the classification accuracy ($57.86\% \pm 13.34$) is quite low with high variation in performance across folds, indicated by the high standard deviation.

As a baseline comparison to traditional non-TDA methods, we also pass the original shape images into a CNN of the same overall architecture. We use the same overall model architecture, without the cylinder padding. Before training the model, we first reduce the resolution of the MPEG7 images to $[32, 48]$, so that the input image sizes are consistent across different model, data-types for more consistent comparison of the classification methods. Also to maintain consistency, we

apply random rotations to each of the shape images, so input data is truly un-aligned. Traditional CNNs are not rotation invariant, yet we see that the shape image CNN model outperforms the ECT-CNN. This is likely due to the distinct edges present in the original silhouette images, which are not as apparent in the ECT images. CNNs are formulated to extract edge information, and the ECT representations on their own are more pixelated.

Finally, we combine the TDA and traditional methods, stacking the original MPEG7 image with two copies of the ECT image into a three channel image of size $[3, 32, 48]$. Input to a CNN model is assumed to be either a single channel (greyscale) or three channel (color image with red, green, blue channels) image, by default. To combine the ECT representation with the original silhouette image, each individually represented as a single channel image, we choose to stack the ECT and original image into a standard three channel representation. We choose to use two copies of the ECT image instead of two copies of the original image, but note that swapping these yields similar classification results. Now, using a 3-channel image instead of the previously used single channel images, we pass the stacked image into the same CNN model for training. Combining the traditional image CNN with the ECT data yields a higher classification accuracy; in fact, we achieve 100% test accuracy after just 15 training epochs. While the TDA method ECT-CNN on its own did not outperform the standard image CNN, we are able to boost the standard model performance by including the ECT data.

4.6 ECT parameter considerations

In this series of ECT computation and model comparisons, there are a number of parameters which are hand selected, yet could influence various aspects of model performance. To isolate experiments of interest to evaluate the ECT-CNN, we restrict to providing experimental results without much parameter optimization. However, in this section we discuss methods for more data-optimized architectures.

One such set of parameters which were hand-selected are the resolution of the ECT images. The number of directions D sampled and the number of thresholds J used for the ECT computation were selected to be 32 and 48, respectively. The addition of more directions would mean that the

discrete ECT more closely approximates the continuous ECT. We can see the effect of this in the sampling artifacts that appear in the ECT of the same input, computed at different orientations (refer to Fig. 5.9 in Ch. 5). There are small pixels that appear and disappear as you recompute the ECT for the image at a different orientation because the resolution of the ECT is not as fine-scaled as the refined boundaries of the input shape. However, for the same choices of D, and J, the example shown in Fig. 4.5 does not show sampling artifacts: the translation in the image as the input data orientation changes is smooth because the scale of structure in the input data is much larger relative to the resolution of ECT parameters. Generally, increasing the number of directions sampled and the number of thresholds at which to compute the Euler Characteristic would make translation appear more smooth in the ECT images because it allows us to pick up changes in topological structure at a finer scale. Smoother translation of ECTs results in a CNN model that more closely approximates a rotation equivariant pipeline.

4.7 Discussion & Future work

The presented CNN architecture used for the ECT-CNN produces a rotation equivariant pipeline in practice, with rotation invariance coming mostly from training considerations (dataset size, pooling filter size, etc). There are limitations to consider when using a simple, out of the box CNN like this for ECT inputs. Future work in this domain should include adjustments to the architecture to force the model into having translation invariance built-in to the architecture. Using a truly architecturally translation invariant CNN model would allow the ECT-CNN pipeline to have the desired rotation invariance property that motivates the use of un-aligned data.

While we describe the classification of ECT data using a CNN, the same idea could potentially be applied for the use of other topological signatures in conjunction with the directional transform. Using persistent homology as the choice of topological summary results in representation of data in the form of the PHT, there is a trade-off: topological information gain at the cost of more expensive computation than the ECT. Unlike the ECT, which is a collection of directions and corresponding Euler characteristic vectors, the PHT is a collection of directions with corresponding persistence diagrams. As is well-documented in the machine learning for TDA literature, the space

of persistence diagrams is not well-suited for machine learning due to properties such as having non-unique means [1, 4] and thus require vectorization. In order to use persistence diagrams in standard machine learning pipelines, and also to use them in the context of this directional transform neural network, the diagrams must first be vectorized. Various methods of persistence diagram vectorization exist, including widely-used persistence images [1], persistence landscapes [13], template function featurization [74], and silhouettes [18]. One choice of vectorization, the Betti curve, would provide a vector representation necessary for input to the directional transform neural network at the cost of notable topological information loss. The Betti curve maps each persistence diagram to an integer-valued curve, closely related to the Euler characteristic curve which is equivalent to alternating sums of Betti numbers. Using the PHT would also introduce additional considerations to maintain the desired rotation equivariance property. The choice of vectorization of persistence diagrams would have to be formed carefully, such that rotations of the input simplicial complex correspond to translation of features in the PHT "image". Overall, it is possible to use PHT data instead of the ECT in a similar pipeline, however, notable preprocessing would be required to get to the point of directional transform "images" to be passed into the CNN.

Another natural direction for future work is the consideration of input data in higher dimensions; in this work, we have restricted to the setting of 2D input data which we can represent as an embedded simplicial complex in $\mathbb{R}^2$. If we consider the case of 3D input data, perhaps represented as a point cloud $x_0, x_1, ..., x_n \in \mathbb{R}^3$, we can still represent the data as an embedded simplicial complex and compute the Euler Characteristic Transform as a faithful representation of the shape of the data [84]. While the mathematics of the ECT are clear to generalize to higher dimensions, the generalization of representing ECT data as an image for input to a CNN is not obvious. For 3D data, instead of the ECT data living on a cylinder $\mathbb{S}^1 \times \mathbb{R}$, where we can represent it as a 2D tensor with two edges identified, now we have ECT data that lives on a hyper-cylinder $\mathbb{S}^2 \times \mathbb{R}$. Representing a 3D tensor in PyTorch is simple, however in order to do so, we require a regular grid pattern to arrange each of the Euler Characteristic Curves. Sampling directions on $\mathbb{S}^2$ in a regular grid pattern would permit you to construct such a 3D tensor, however there is no natural way to sample the sphere in this way.

Being able to directly generalize our outlined ECT-CNN method to 3D would require being able to have a coherent grid on $\mathbb{S}^2$. A recent preprint [69] uses 3D input data with a neural network for a similar goal of classification of ECT data. Instead of leveraging translation invariance properties of CNNs, they use a Graph Neural Network as an intermediate step before passing the data to a CNN because GNNs allow for more flexibility in the underlying data structure-specifically not restricting it to live on a grid. In their pipeline, they circumvent the issue of sampling directions on $\mathbb{S}^2$ in a regular grid by representing the data ECT data as a signal on a graph, where graph vertices are defined to be the sampled ECT directions on the sphere. It would be interesting in future work to find a more direct connection from the 3D input information to a CNN.

CHAPTER 5

APPLICATIONS OF THE ECT CNN

In this chapter, we apply the Euler Characteristic Transform-Convolutional Neural Network (ECT-CNN) to two different leaf shape datasets, exhibiting the utility of convolutional neural networks (CNNs) for classification of Euler Characteristic Transform (ECT) data. While the details of the ECT-CNN method are presented in Ch. 4, here we provide motivation for its application to study the shape of leaves in a rotation invariant manner. One of the primary features of the ECT-CNN pipeline is that it does not require the input data to be prealigned, as is the case for many traditionally used shape classification methods. Depending on the application, pre-aligning data can be computationally challenging, and in some cases infeasible. While neither of these cases apply to the leaf shape datasets presented here, avoiding a preprocessing step of alignment is a valuable feature of the ECT-CNN.

In the plant biology community, measuring and understanding the shape of leaves has important consequences to phylogenetic and evolutionary relationships, adaptations, and plant characteristics. As such, the effort to quantify the naturally occurring widespread variation in leaf morphology is very important. Often, traditional measures are used by biologists to quantify the shape of leaves. Many such methods are direct measures of leaf size including length, width, and aspect ratios of leaf size. Other more complicated shape measures involve measuring concavity and curvature of leafs, geometric morphometric methods, and Elliptical Fourier Descriptors [20]. We propose the use of the ECT as a simple yet powerful measure of shape to complement the standard statistical methods that are traditionally used to study plant morphology.

Previous applications of the ECT to biologically-relevant data often take a single vector representation of the ECT and use that with SVM models for classification. In [2], the authors the ECT to quantify the shape of barley seeds. They concatenate all of the ECC vectors into a high dimensional vector, perform dimensionality reduction using principal component analysis, and then pass it into a SVM for classification. Importantly, because they concatenate the ECT into into a single vector, the relationship between the directions each ECC is computed from is lost. Because

of using the SVM classifier, this method requires that the data is pre-aligned; indeed alignment is a requirement for interpretability of which directions were most useful in the classification. As a baseline comparison to our ECT-CNN method and to show which is gained, we also apply an SVM classifier to ECT vectors, despite using non-oriented input data. Choosing between SVM and CNN models for classification of ECT data comes with a trade-off between accuracy and interpretability. With SVM models applied to vectorized ECT data of oriented data, the feature weights give insight into which directions of the ECT were most useful for discrimination between classes in the classification. CNN models of ECT data are able to leverage the additional structure given by directions on $\mathbb{S}^1$, yielding higher accuracy. Additionally, because of the rotation equivariance property of the ECT-CNN which was discussed in Sec. 4.4, there is no need to pre-align or orient the input data before ECT computation. The boost in accuracy, however comes at a cost of losing interpretability.

5.1 Datasets

In this section, we describe two leaf-shape datasets that will be used to experimentally validate the utility of the ECT-CNN pipeline proposed in Ch. 4. Each of these datasets have been collected and curated with the goal of further understanding plant morphology and the role of plant shape in biological development and evolution. In previous work, more traditional and statistical measures have been used to quantify the shape of the leaves in these datasets [20, 56].

5.1.1 Maracuyá (Passiflora) leaf graph dataset

In this leaf dataset, Chitwood and Otoni amassed a collection of 3300 leaf samples from 40 different Maracuyá species [20]. Maracuyá, also commonly referred to as Passiflora, is a plant genus with large amounts of variation between its species. Each of the leaves in the dataset are pre-aligned using Procrustes analysis and are represented by 15 landmark points around the leaf shape. The landmark point positions are designated according to the leaf vasculature, sinuses, and lobes such that the relative positioning of all of these variations is adjusted in the landmark coordinates. Figure 5.1 from [20] shows example leaves for each species and samples of each overlayed to show the representation of each leaf. We use coordinates $(x, y) \in \mathbb{R}^2$ of the 15 landmark points on each leaf sample to represent the data as an embedded graph. Each of the landmark points is represented

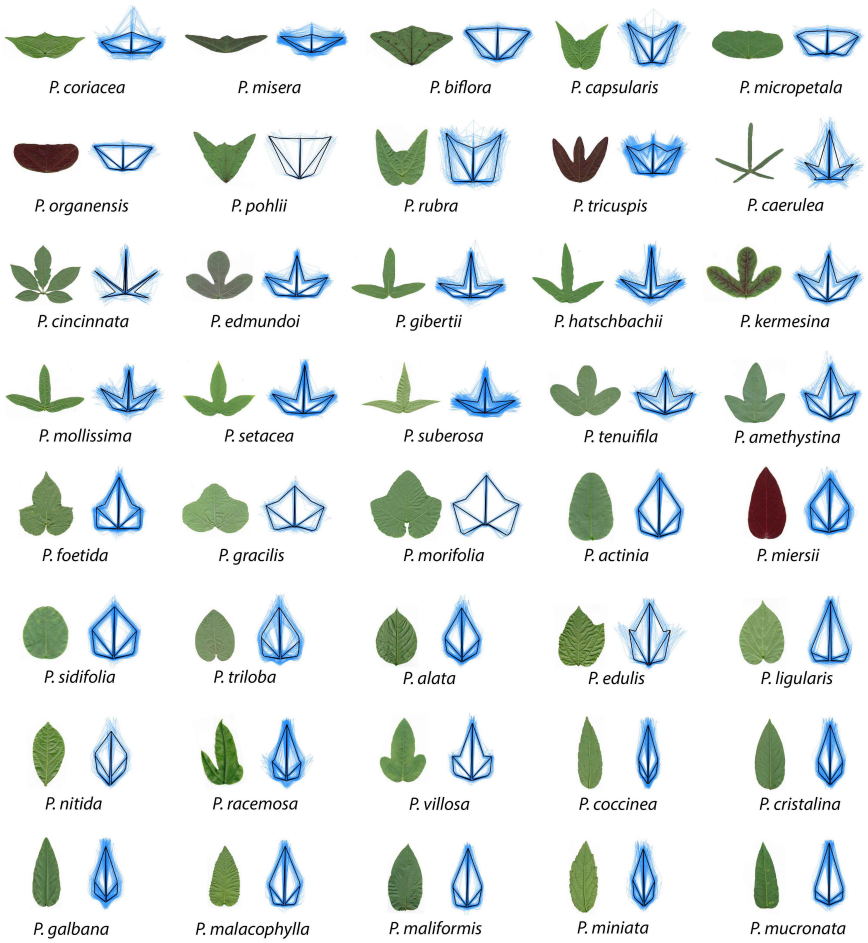


Figure 5.1 Showing samples of each represented Maracuyá species. Reproduced from [20], Creative Commons Attribution License by the Authors.

in the graph as a vertex, and 24 edges are included between vertices according to a general vein structure of leaves (see examples of the graph structure in Fig. 5.2). The result is an embedded graph representation of 15 vertices and 24 edges for each leaf sample, which can then be used for classification according to species label. For the ECT computation of all graphs in the dataset, we first find a bounding box for the entire dataset using all xy-coordinates of the graphs. The global bounding box ensures that the spacing between thresholds is equal for each direction so that the scale of structure within each sample is accounted for in the ECT computation. Alternatively, the ECT can be computed using a fixed number of thresholds, evenly dividing the distance between the minimum and maximum filtration values of the shape for that specific direction. This construction can result in differently spaced thresholds for different directions and thus is useful in settings where

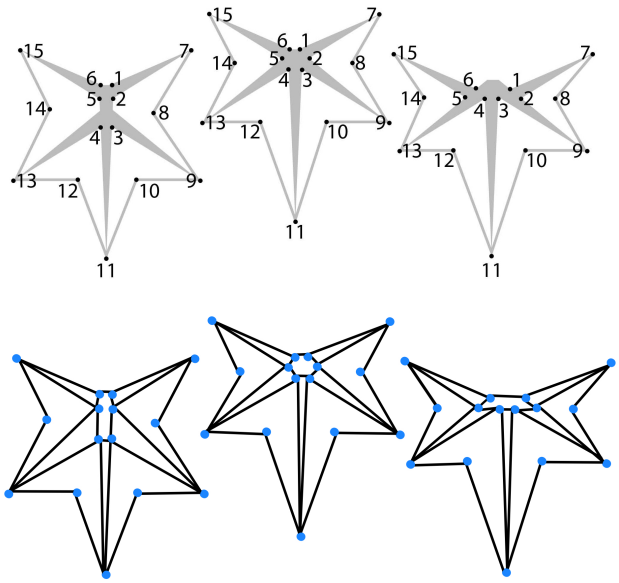


Figure 5.2 Three examples of Maracuyá landmarks (top) and their corresponding graph representation (bottom).

global size differences between samples should not be taken into account.

Note that while there is notable variation between the shape of leaves from different Maracuyá species, the graph representation is too simple and averages out the differences. This behavior can be seen in 5.1, where the shape of the actual leaves look quite distinct between different classes, however the landmark representations are very similar, especially considering the variation within classes shown by the blue "shadow" outlines.

5.1.2 Leaf family outline dataset

The second leaf shape dataset we use to test the ECT-CNN method is a set of leaf outlines, curated by [56]. See [56, Table 1] for the sources and dataset authors for each of the different plant types represented in this large compiled dataset; the dataset we use represents a subset of the one described in the table, representing leaf samples from all of the leaf types listed except the Climate and LeafSnap datasets. This dataset consists of leaf outlines curated from a collection of published and unpublished datasets, representing 14 different plant groups.

Samples in the dataset are coordinates $(x, y) \in \mathbb{R}^2$ that form outlines of leaves. The data is pre-centered and scaled to account for any sampling resolution differences between leaf samples.

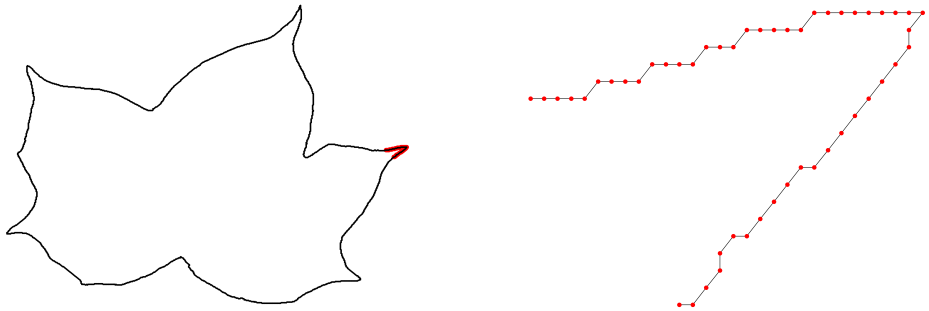


Figure 5.3 An example graph representation of a leaf outline from the Cotton class, zoomed in (right, in red) to show graph structure and the subgraph highlighted in red on the full outline graph.

In the published version of the dataset, these coordinates are not ordered, however in order to represent the leaf outlines as a graph, we require the coordinates be ordered such that the outline of each leaf can be traced out by the graph representation. To address the lack of ordering of the coordinates, we use the dataset version from [91], in which they use 2-nearest neighbor graphs to order all coordinates so as to trace the outline of each leaf. Note that the total number of leaf samples we use in our analysis is less than the original dataset as described in [56], due to omissions of some samples in the coordinate re-ordering step of preprocessing. Using coordinates for each leaf, ordered such that connecting subsequent coordinates in forms the leaf outline, we build a graph representation of each leaf sample. Each of the coordinates of the leaf are represented as vertices in the graph, with edges between vertices that are adjacent in the leaf outline and the graph embedding defined by the coordinates $(x, y) \in \mathbb{R}^2$ for each vertex. Figure 5.3 shows an example of the graph representation for this dataset.

We apply our ECT-CNN pipeline to this dataset for the task of plant type classification (a 14-class classification problem). Apart from the large number of samples in the dataset, a notable feature of the dataset is its class imbalance, shown in Fig. 5.4. Tomato leaves, while only one out of fourteen classes for our classification task, represent more than 50% of the dataset. Selected samples of each leaf outline are shown in Fig. 5.5.

To ensure that the spacing between thresholds is consistent for each direction, and that the scale

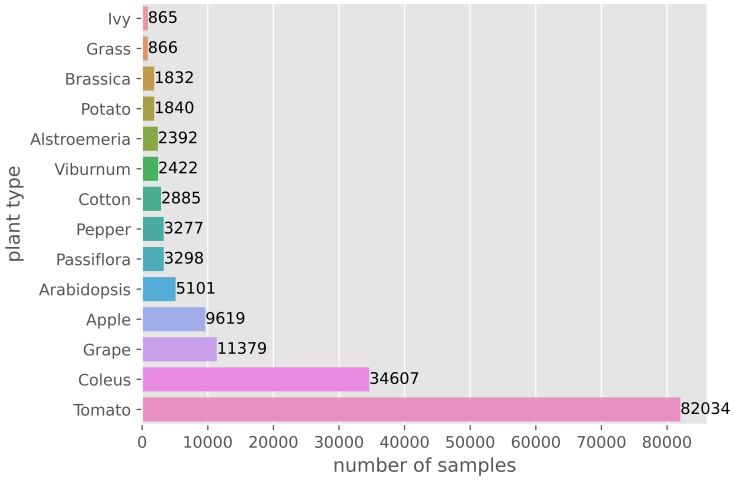


Figure 5.4 Distribution of plant type labels in the leaf outline dataset. The entire dataset consists of 162, 417 samples.

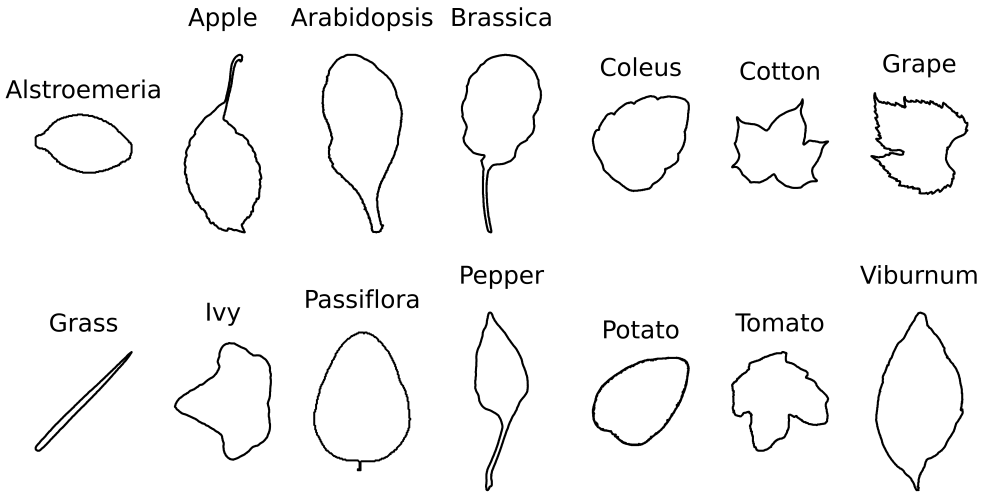


Figure 5.5 Example leaf outlines of each plant type used for classification. Note that there is variation within each of these classes and the samples shown are randomly selected to represent each respective class.

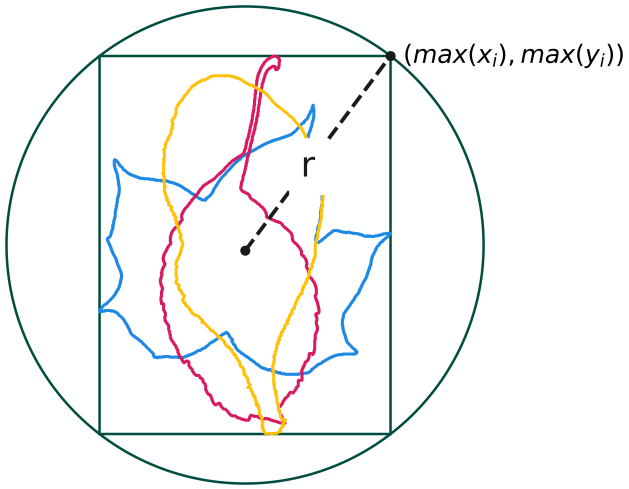


Figure 5.6 An example bounding box for a small collection of leaf outline graphs. In practice, we apply this procedure to the entire dataset to get a single, global bounding ball for the dataset.

of structure in each sample is taken into account, we define a single distance between thresholds and use the same number of these to compute the Euler Characteristic Curve for each direction of each sample. This distance between thresholds is decided using a bounding ball of all coordinates of every leaf outline in the dataset with radius defined to be half of the Euclidean distance between the points $(\min X, \min Y)$ and $(\max X, \max Y)$, where X and Y represent the set of $x-$coordinates and $y-$coordinates of all points in the dataset, respectively. Figure 5.6 shows the bounding box for example leaf outline graphs. Despite the re-scaling and centering of all data as part of preprocessing, there are some leaf samples that are larger than most. This results in a dataset bounding box which does not tightly fit most leaf samples, causing bands of constant zero Euler Characteristic values in the ECT matrices for the dataset (as visible in Fig. 5.10).

5.2 Architecture and hyperparameters

For ECT computation, the parameter choices are the number of thresholds at which to compute the Euler Characteristic and the number of directions sampled to compute the Euler Characteristic curve. We use 48 thresholds for each of the 32 sampled directions, resulting in ECT matrices of

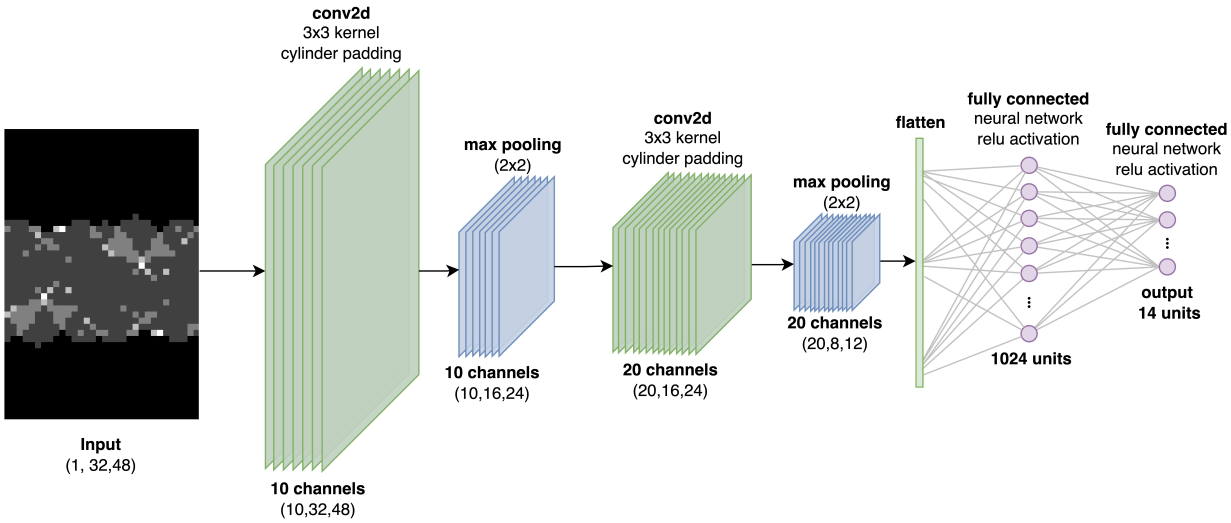


Figure 5.7 CNN architecture used for classification of leaf outline ECT data consisting of two convolutional layers, each paired with a max pooling layer followed by two fully connected layers.

size 32×48 (which we equivalently refer to as ECT images with resolution 32×48). While there may be choices of these parameters that encode more fine scaled structure and thus more closely encode the shape of the input, we note that the effect of such choices is largely dependent on the input data. We consider the influence of threshold and direction numbers on ECT injectivity and the ECT-CNN rotation invariance properties in Ch.4.

We use a simple CNN architecture for classification models, depicted in Fig. 5.7, consisting of two convolutional layers paired with max pooling layers, followed by two fully connected layers for classification. For both types of CNN models: ECT classification and leaf outline image classification, the overall architecture is the same. To maintain consistency for comparison of the different input types, we maintain the same model parameters as much as possible. Using the described architecture of convolution and pooling layers, all of the CNN-style models have $2,010,204$ trainable parameters, except where otherwise noted that the learning rate and input data size are different.

There are necessary differences between CNN models of the ECT images and regular images, however. For ECT image classification, we use cylinder padding as described in Ch 4, while for standard image classifiers we use all zero padding of the same size. In some cases, the learning rate is adjusted (and denoted in reported classification results for that model). One such case is

using a higher resolution image for classification. In this setting, using a learning rate that is too high causes testing and validation accuracy to peak quickly and then sharply drop off and stay low for the remainder of the training epochs. By decreasing the learning rate in these cases, we achieve smoother loss curves with better overall accuracy in the model because the model avoids getting stuck in a sub optimal region of the parameter landscape.

Cylinder padding is done using PyTorch with a combination of 2D circular and zero padding, selecting the padding size such that the size of the feature map is preserved (i.e. the input data size is the same as the output feature map size). We achieve this (assuming stride=1) by taking the padding size to be $\lfloor \frac{k}{2} \rfloor$, where k is the kernel size. For example, for input image of size 5×5, if the kernel size is 5, stride is 1, and the padding size is 2, the output feature map size is 5×5.

5.3 Results

In this section, we describe the results of using the ECT-CNN pipeline for classification on the previously described leaf graph dataset and leaf outline dataset. We also provide comparison using the SECT variation as input to the model and baseline comparison of classification of the ECT vector representation using an SVM model. To compare each of these TDA-based methods to traditional classification methods, we also train a CNN model for classification of image versions of the original input samples. Finally, we combine the TDA and traditional methods to train a model which classifies 3-channel images consisting of stacked ECT images and the traditional image version of samples.

5.3.1 Leaf graph dataset

To apply the ECT-CNN pipeline, its model variations, and comparisons, we have four versions of dataset samples. Three of these input data types are depicted in Fig. 5.8: the image version of leaf graphs, the ECT of leaf graphs, and the SECT of leaf graphs. The fourth representation used as input to a classification model is the ECT vector, which we get by column-wise flattening of the 32×48 ECT image, resulting in a 1536×1 vector.

Classification results of the leaf graph dataset using ECT-CNN model (and comparison models) are summarized in Table 5.1. We use the Maracuyá species as a label, resulting in a 40-class

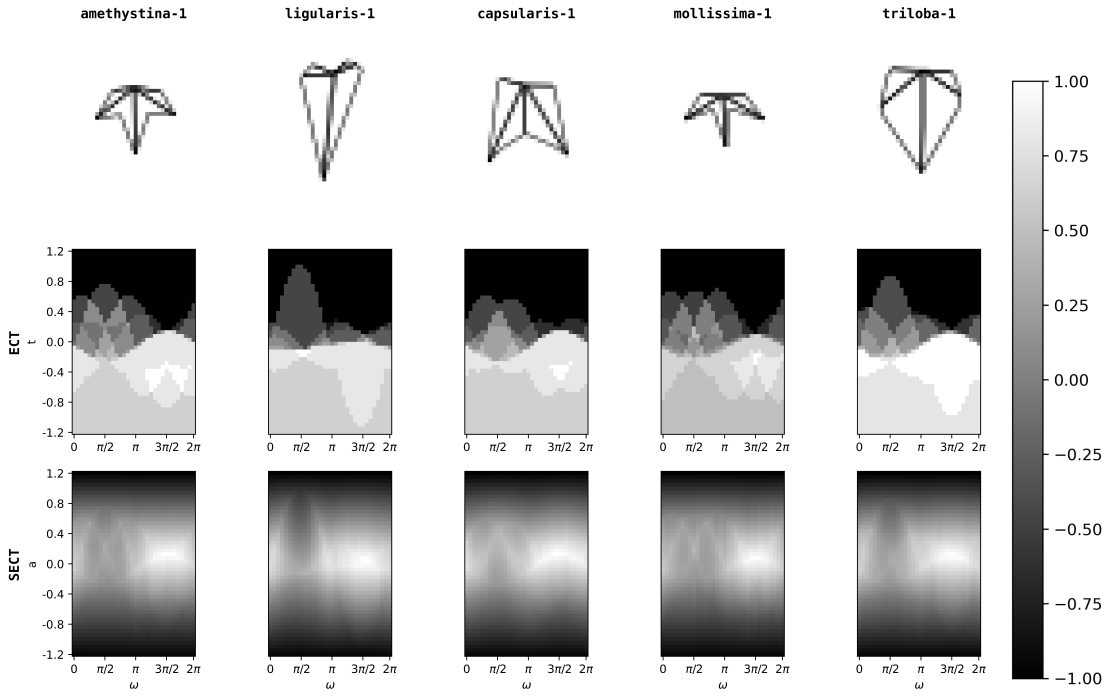


Figure 5.8 Samples from five classes used for classification in the leaf graph dataset: 'amethystina', 'ligularis', 'capsularis', 'mollissima', 'triloba'(top), the corresponding ECT images (middle), and corresponding SECT images (bottom). Pixel values in the ECT and SECT images are not the raw Euler characteristic values. The images are normalized to $[-1, 1]$ as part of preprocessing for CNN classification.

Leaf graph dataset classification accuracy, by input data and model				
	Model	epochs	lr	accuracy±std
TDA	32×48 ECT CNN	100	10^{-3}	71.05% ± 1.34
	32×48 SECT CNN	100	10^{-3}	23.29% ± 21.52
	1536×1 ECT SVM	-	-	32.33% ± 2.57
Traditional	32×48 shape image CNN	100	10^{-3}	59.66% ± 2.06
Both	32×48 shape image + ECT CNN	100	10^{-3}	71.10% ± 2.40

Table 5.1 Summary of classification results on the leaf graph dataset for different combinations of input data and model used. Each reported accuracy and standard deviation is computed with 10-fold cross validation.

classification problem. Each of the reported classification accuracy results are computed using 10-fold cross validation to reduce the influence of a specific train/test split on reported model performance. The accuracy results shown are the average test set accuracy and standard deviation over the 10 folds.

TDA-based models Our primary method of interest, the ECT-CNN, uses the ECT images as input to a CNN. These models are trained for 100 epochs, resulting in an average testing accuracy of 71.05% ± 1.34 across 10-folds. A crucial property of the ECT-CNN pipeline is its equivariance to rotations of the input data as described in Sec. 4.4, so following the method described in Ch. 4, we apply random rotations (or associated translations in the case of ECT images) to each of the input samples before training. Preprocessing of the leaf graph dataset, including the placement of landmark points using Procrustes alignment, causes the embedded graph representations to all be aligned. Training the CNN models on aligned data would risk potential overfitting to the orientation of leaf samples, as opposed to the model relying on the signal of interest, namely the leaf shape quantified by the ECT. Eliminating alignment of the leaf images and ECT images reduces the risk of the model fitting to leaf-alignment and allows us to more accurately evaluate the rotation equivariance of the ECT-CNN pipeline. Thus, for all of the classification models described, we must apply the random rotations to samples before training to remove the rotational alignment.

As noted in the classification results of the MPEG7 dataset in Ch. 4, the poor SECT CNN model performance can be attributed to model parameters chosen to keep the comparison consistent across different input data type experiments. Similarly, we could expect to boost the SECT CNN performance to match the ECT-CNN performance by increasing the kernel size for convolution so that the global features of the SECT images are captured in the convolutional layers.

To provide a comparison to the standard choice of classification model for ECT data, we classify Maracuyá species using an SVM. See these results included in Table 5.1 as a TDA method. The low accuracy of this model (32.33% ± 2.57) is expected considering the use of unaligned input data. Vectors passed to this model are very high-dimensional, yet the poor model performance can be largely attributed to the lack of alignment in the input data. Applying the same SVM model for classification of the ECT vectors, without random rotations of the input samples, yields much better performance: 63.63% accuracy with a standard deviation of 2.47%.

Traditional model An interesting result is the relatively poor performance of the traditional CNN classifying image versions of the leaf graphs (see Table 5.1). Images of the leaf graphs used for classification are intentionally low resolution; we use 32×48 image representations of the graphs to maintain a more direct comparison to the ECT-CNN, where the ECTs are computed using 32 directions and 48 thresholds in each direction. Keeping all other model architecture choices the same, using different sized input images would cause exponentially different numbers of trainable parameters in the model. Having a model with more trainable parameters can increase the model's ability to approximate a function and thus increase model performance; consequently, having too many trainable parameters can cause model overfitting to the training data and an inability to generalize well to unseen testing data. To avoid the influence of the number of trainable parameters and associated performance trade-offs, we use image representations of the same image size as the ECT images.

The choice of pooling layer in the CNN architecture could also impact the expressiveness of the CNN model. In general, MaxPool layers, which coarsen the feature map output of a convolution layer, are effective on images with a dark background. By picking out pixels with the maximum values within each filter-size section of the feature map, these pooling layers retain information from the light-pixels. Conversely, MinPool layers are effective on images with light backgrounds because they are able to extract the dark pixels when they select the minimum pixel values within sections of the feature map. With this behaviour in mind, we re-trained the traditional CNN model on the leaf graph images, swapping the MaxPool layers with MinPool layers in our architecture. However, there was no effect on classification performance of the model, despite the light pixel backgrounds of the leaf graph images. The same shape information is available in the original leaf graph image or if we took the negative of the image because it is a binary image outline. Thus, swapping the pooling layer type did not have the effect of boosting the model's performance on classifying the images.

For consistency of model architectures across the different input data types, we use a CNN model for comparison. However, a more direct (and well-suited for the input data representation) baseline

comparison for this dataset would be to use a graph neural network for classification. Before any ECT computation or transformations to the data, we are working with embedded graphs, and the most well-suited deep learning architecture for graph classification is a graph neural network. Using this method, we could pass the graph representation directly into a model, instead of converting the graph representation to an image and passing it to a CNN. This analysis method could be useful as an additional avenue for species classification of the dataset, however it would not be directly comparable to the CNN-style networks we propose for classification of ECT data.

TDA combined with traditional model The combined model (TDA and traditional) notably improves from just the image classification using a traditional CNN. Combining the leaf graph image with the ECT images yields only slightly better performance than the CNN model for the ECT alone. As was also the case for the MPEG7 dataset in Ch. 4, the combined ECT and traditional images outperform all considered methods. Interestingly, the combined model, which now uses a 3-channel image compared to the 1-channel images for the other CNN models, starts to overfit beyond 15 epochs. The dataset is relatively small with only 3300 leaf samples, distributed across 40 classes (the Maracuyá species). The dataset's limited number of samples per class and simplified representation of each leaf as a graph contribute to the restricted classification model performance observed across the different input types and models considered.

Within species variation: heteroblasty This leaf graph dataset consists of many vines of leaves sampled for each species, with varying numbers of leaves sampled from each vine. In addition to the species labels of samples, the order of the leaves as they come out from the vine is also recorded in the dataset. A difference in leaf shape occurring over its lifespan is called heteroblasty, in this case denoted by the integer count of the leaf from the base of the vine. Older leaves are at the base of the vine and younger leaves at the tip. In general, the oldest leaves from a shoot have a different shape than the youngest leaves at the growing tip of a vine, which accounts for the large amounts of shape variation within a single species. The varying heteroblasty, and subsequent leaf-shape variation, make classification by Maracuyá species a more difficult task; as is the case with any model, having

low between-class variance and high within-class variance makes classification challenging. An alternative analysis route of biological interest would be classification by heteroblasty across all Maracuyá species in the dataset. A model for this task would be useful to understand the relationship of leaf age and shape within the Maracuyá genus.

Dataset limitations Representing each sample as a graph using landmark points is a notable limitation of this method. As discussed in the previous paragraph, classifying the dataset by species already presents a challenge due to the high within-class variance and low between-class variance. The embedded graph representations using the same vertices, edges, and adjacency information for each sample means that the only variation between samples is restricted to the landmark coordinates (which give us an embedding for each graph). The simplicity of this representation is useful because the resulting graphs are easy to construct and relatively small, so the ECT computation is not very expensive. However, relying on only the landmark coordinates does not give a lot of information to build particularly meaningful and discriminatory representations. Additionally, classifying the dataset by species is a 40-class problem, which presents a challenge, in particular because of the high variance within classes.

5.3.2 Leaf outline dataset

In contrast to the previous leaf graph dataset, the leaf outline dataset is much larger and samples in this dataset have more refined boundaries than the simplified graph representations. Leaf outline samples of this dataset are still represented as embedded graphs, yet the scale of structure is more fine-grained because the graphs themselves are much larger: on the order of 10^3 vertices per graph, as opposed to the 15 vertices of the previous dataset (Fig. 5.3 shows an example of embedded graph size and the scale of structure in the leaf outline dataset). The smaller scale of features presents as noise in the correspondence between rotation of the input leaf outlines and translation within the ECT image representations. Figure 5.9 shows the rotation of a cotton leaf outline and the corresponding ECT images. In this dataset, because we are still only using 32 directions and 48 thresholds for the ECT computation, and because of the fine-grained outline information, the

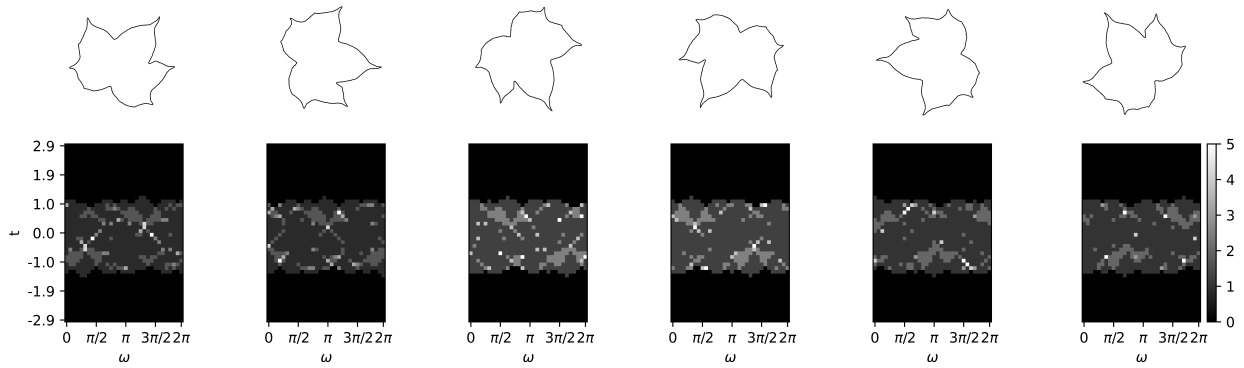


Figure 5.9 An example leaf outline from the Cotton class (top) and associated ECT image (bottom), recomputed for rotations of the leaf outline. As the input is rotated (clockwise from left to right), the values of the ECT matrix are translated, as seen by the left-to-right shift of the patterns within the image.

translation does not appear smooth. Increasing the resolution of the ECT images (i.e. increasing the number of directions and thresholds) would reduce the sampling artifacts in the ECT images as the input outline is rotated. Examples of three out of four of the input data types are depicted in Fig. 5.10: the image version of leaf outlines, the ECT of leaf outlines, and the SECT of leaf outlines. The vectorized ECT, the fourth representation used as input to a classification model, is a 1536×1 vector.

Classification results of the leaf outline dataset using the ECT-CNN model (and comparison models) are summarized in Table 5.2. We use the different plant types as labels, resulting in a 14-class classification problem. Each of the reported classification accuracy results are computed using 10-fold cross validation to reduce the influence of a specific train/test split on reported model performance. The accuracy results shown are the average test set accuracy and standard deviation over the 10 folds.

TDA-based models The first section of results in the table show the TDA-based methods, the various ways we use ECT data for classification. First, the ECT-CNN, which is trained using the regular ECT images and achieves an average test accuracy of 88.98% ± 0.28 over 10-folds. In addition to the raw ECT data, we also transform the ECT image to get a smooth version: the SECT. Without changing any of the model architecture parameters, we train this additional CNN model

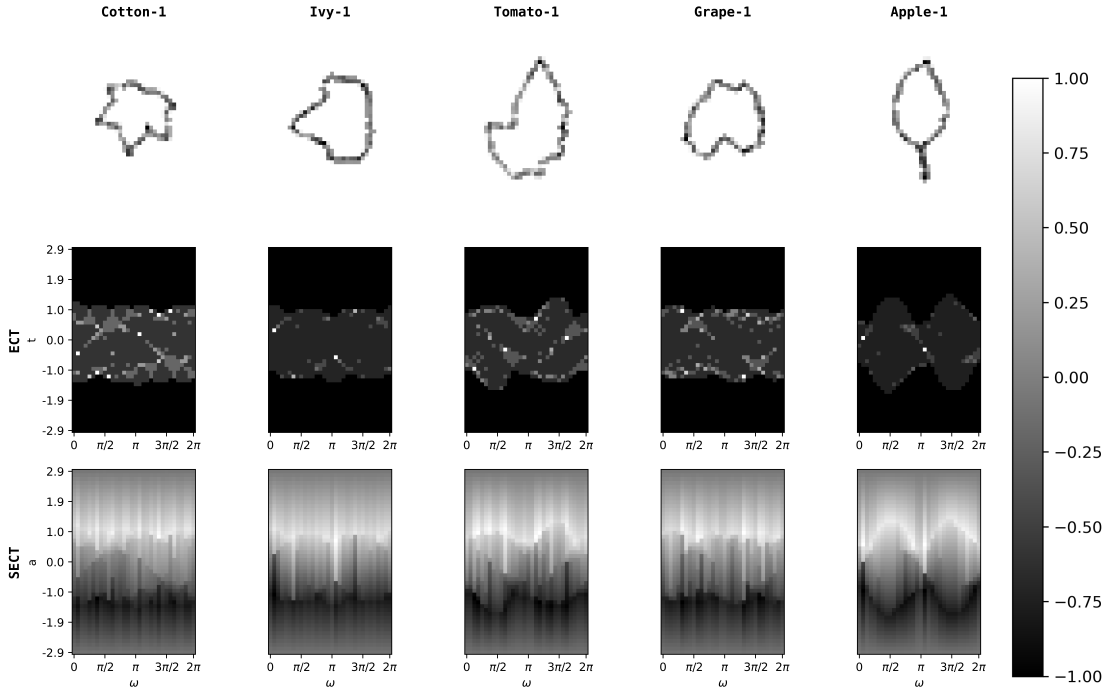


Figure 5.10 Samples from five classes used for classification in the leaf outline dataset: 'Cotton', 'Ivy', 'Tomato', 'Grape', 'Apple' (top), the corresponding ECT images (middle), and corresponding SECT images (bottom). Pixel values in the ECT and SECT images are not the raw Euler characteristic values. The images are normalized to $[-1, 1]$ as part of the CNN preprocessing.

Leaf outline dataset classification accuracy, by input data and model				
	Model	epochs	lr	accuracy
TDA	32×48 ECT CNN	15	10^{-3}	88.98% ± 0.28
	32×48 SECT CNN	15	10^{-3}	86.42% ± 0.50
	1536×1 ECT SVM	-	-	70.08% ± 3.72
Traditional	32×48 Leaf outline CNN	15	10^{-3}	90.38% ± 0.25
	90×90 Leaf outline CNN	15	10^{-4}	91.62% ± 0.27
Both	32×48 Leaf outline + ECT CNN	15	10^{-3}	94.95% ± 0.14

Table 5.2 Summary of classification results on the leaf outline dataset for different combinations of input data and model used. Each reported average accuracy with standard deviation is computed with 10-fold cross validation.

using the SECT versions of leaf outline training data. The resulting test accuracy on this second model shows a slight drop from the ECT-CNN accuracy to 86.42% ± 0.50. As was also the case for the MPEG7 and leaf graph datasets, we observe the drop in model performance after smoothing the ECT images due to the scale of features changing in the images. Because of the smoothing, features in the SECT images are generally more global and encompass a larger scale of the image than the features in the ECT images. All model architecture choices were kept consistent from each of the two models, however increasing the convolution filter size for the SECT model would likely boost the overall performance. To compare the ECT-CNN to the standard method of classification for ECT data, we also use a SVM to classify vectorized versions of the ECT data. As expected, the classification accuracy (70.08% ± 3.72) is quite low with higher variation in performance across folds, indicated by the high standard deviation. For all of the CNN models applied to this dataset, however, we see very low variation in performance across folds due to the large size of the dataset and ability of the deep learning models to achieve more stable performance as a result.

Traditional model The baseline comparison to the CNN classifying image versions of each sample slightly outperforms our ECT-CNN method on this leaf outline dataset. We see a small difference of about 1.25% classification accuracy on average between the methods. However, it is crucial to recall that CNNs are not inherently equivariant to rotations or changes in scale. While we see that the traditional CNN models performs well and has high classification accuracy on the leaf outline images, the model is not truly rotation equivariant. For the traditional CNN to be rotation equivariant, it would need training data samples of each class representing the entire 360° spectrum of rotations. CNNs can approximate rotation equivariance only up to the degree of rotations represented in samples of the training data. Similarly, CNNs are not inherently scale invariant, so approximating a function that is would require many training samples, representing large scale variations. In this dataset, because of its large size and ability to train on many samples, representing many rotations of each class, the traditional CNN model is able to perform well and approximate a rotation equivariant function. CNNs are, however, inherently translation equivariant.

Relying on this property, the ECT-CNN model is able to provide a rotation equivariant pipeline without the same requirement for large amounts of varied training data to act as data augmentation to achieve rotation equivariance.

TDA combined with traditional model Finally, we see that the best performing classification method for the leaf outline dataset is achieved by combining the TDA-based method using the ECT images with the traditional method using standard images of each sample. As was the case in both previously discussed datasets, we achieve the best average classification accuracy by stacking two copies of the ECT image with a single copy of the sample image, forming 3-channel image samples.

5.4 Discussion and future work

Overall, there are two primary factors influencing the better classification performance on the leaf outline dataset versus the leaf graph dataset: 1) the large size of the dataset and 2) the refined boundaries in the outline representations of leaf shape.

Stability of the ECT One of the limitations of the ECT as a shape descriptor tool is its output sensitivity to small perturbations of the input data [19]. In [62], they introduce a stability result for the ECT which is independent from the triangulation of the shape and propose a smoothing method for embeddings of 1D CW complexes with added noise which affords stability and consistent statistical estimator for the ECT of noise data. It is worth noting that the stability issues apparent to the ECT also occur using standard persistent homology and the persistent homology transform, stability results of which are shown in [23] and [84], respectively. In [43], the authors further consider the stability problem of the ECT for non-homeomorphic shapes.

For example, fix points $\{(x_i, y_i)\}_{i=0}^{n} \in \mathbb{R}^2$ connected to form a cycle and compute the Euler Characteristic Curve for a fixed direction. Then add noise to the data, for example, a single point (x_{n+1}, y_{n+1}) added separate from the cycle. The ECC of the new point cloud will be very different than the original, regardless of the proximity of the noise data point to any other point in the data. Figure 5.11 shows the large effect of this small perturbation on the output ECC for a fixed direction. In both of our applications to leaf shape datasets, we avert the issue of ECT stability for

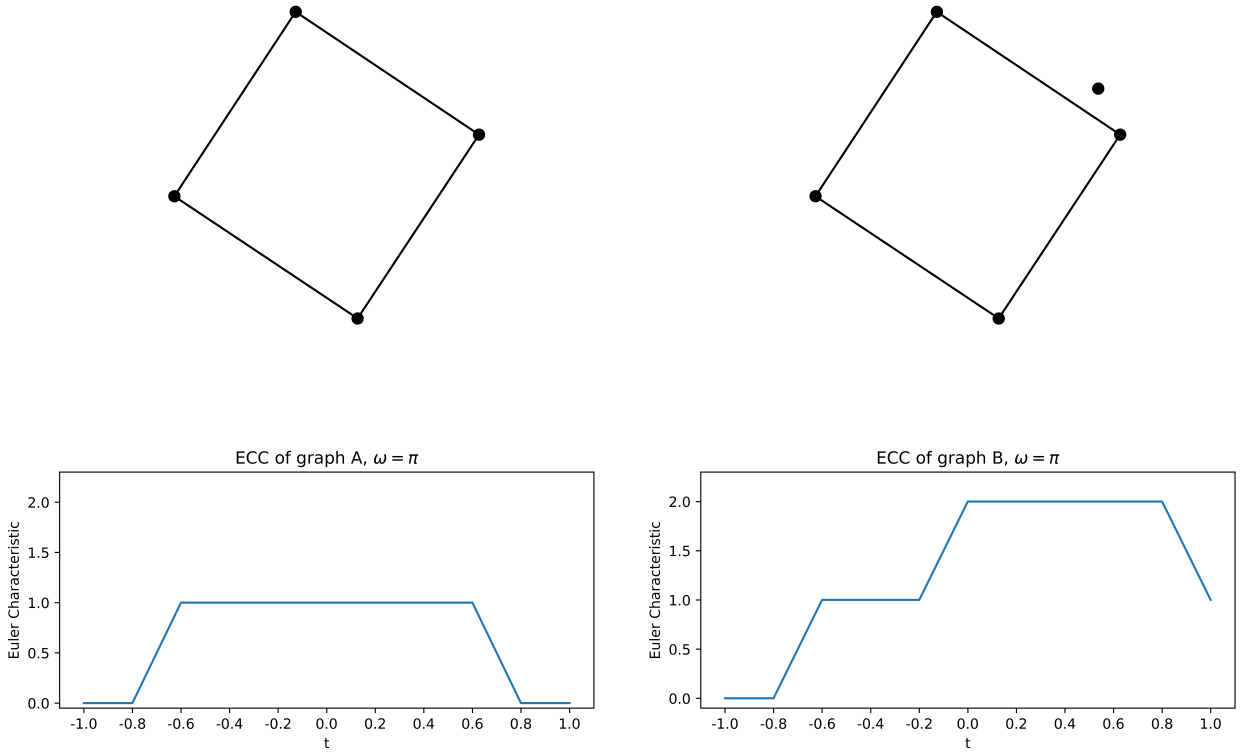


Figure 5.11 Graph A (left, top) and a small perturbation of the same graph, Graph B (right, top). Euler Characteristic Curves (bottom) of each graph computed from fixed direction $\omega = \pi$, showing the stability issue of the Euler Characteristic Curve for non-homeomorphic shapes.

non homeomorphic shapes. The leaf graphs are all homeomorphic representations of leaves due to having the same graph structure (vertices, edges, and adjacency information); the embeddings of each graph vary, but they are homeomorphic to each other. For the leaf outline dataset, while we have varying graph structures, each of the leaves is represented as a cycle graph from the outline of the leaf. Again, the graph embeddings of the dataset vary, yet the leaf outlines are all homeomorphic to $\mathbb{S}^1$. Application of the ECT for these specific datasets avoids known stability issues for non-homeomorphic data due to the consistent graph structure and outline structure used to represent the data.

We note that for most of the CNN models on the leaf outline dataset, the training converges to high accuracy in the first few epochs. Convergence to a high accuracy early in training can indicate redundancy in the dataset, and thus we could likely subsample from each class to reduce

computation time without compromising the discriminatory power of the model. Subsampling would also reduce the computational cost and training time for each model. We do not report training time for each of the different input data types used because the model architectures and input data sizes are kept consistent across the different data types. As a result of keeping the number of trainable parameters constant, the dataset size is the largest driver of computation time (for both ECT computation and CNN model training). As a much larger dataset (with larger graph representations of each sample), the leaf outline dataset takes much longer to compute the ECT and to train CNN models than the leaf graph dataset. While the computational time is not prohibitive, it would be useful in future work to parallelize the code for ECT computation. Additionally, for the use of large datasets, especially those larger than the leaf outline dataset, using GPUs would provide necessary speed ups for training the CNN models.

CHAPTER 6

CONCLUSION

In this dissertation we describe two distinct ways to use topological input data in combination with deep learning models by developing and implementing a pooling method for neural networks on the domain of simplicial complexes, and proposing a CNN framework for the analysis of directional transform data. For each of these data types, in particular both abstract and geometric simplicial complexes, their associated topological structure is exploited in different ways to support their use within their respective deep learning architectures.

In Ch. 3, we propose a pooling layer for neural networks defined on the domain of simplicial complexes. We provide a framework to coarsen and simplify a simplicial complex, given any choice of graph pooling method, or other partition on its vertex set. Following the analogous property for graph pooling (i.e. vertex-permutation invariance), we show that NervePool is a simplex-permutation invariant layer, assuming the choice of input cover on the vertices is a hard partition. Also assuming the restricted setting of a hard partition for the initial vertex clusters, we prove the equivalence of the nerve of a cover (topological) interpretation and the matrix implementation using boundary matrices. Applications of neural networks for the domain of simplicial complexes can be very computationally expensive due to the storage and computation on simplices of higher dimensions. NervePool is a valuable and necessary tool, defining a pooling layer to use within the various neural networks designed for simplicial complexes. It can help to mitigate the computational cost by reducing the dimension of the complexes and also by redistributing signals on simplices in a task-specific manner.

In future work on NervePool, adjustments to the initial vertex clusters to include more topological information would increase the influence of higher-dimensional simplices on the pooling scheme. One potential way to introduce more topologically motivated clustering could be through the use of auxiliary topological loss terms in conjunction with the standard graph pooling methods currently used for vertex clustering. Taking into account the higher-dimensional structure for the learned pooling on underlying graphs in this way, could facilitate model tuning such that

user-specified topological structure is retained from the original complex. Additionally, future modifications to NervePool could use more information from the cluster assignment block matrix to allow for signals on p-simplices to contribute to pooled simplices amongst different dimensions (in contrast to the current version which applies separate coarsening for each dimension of the input simplicial complex). Most importantly, future work on simplicial complex coarsening should include experimental results to identify cases where including simplicial pooling layers can improve model performance as well as comparisons of internal representations to determine if the pooled complexes are meaningful representations of the original simplicial complexes, with respect to a given task.

Chapter 4 proposes the use of CNNs to model ECT data in a framework called the ECT-CNN. Due to the construction of 2D ECT data, we are able to represent its discrete computations as a matrix, and further consider these matrices to be images. Images are well-suited for classification using CNNs, and modeling ECT data in this way facilitates more use of the directional information in classification. While previous applications of the ECT have relied on SVM models for classification of flattened ECT vectors, modeling the data in this way requires the pre-alignment of data before ECT computation. SVM classification prohibits a rotation-equivariant pipeline because changes to the orientation of the input data before computing the ECT are not retained in the flattened representation; there is no association from one direction to the next in the ECT vectorization. By using a CNN with cylindrical padding, we achieve a rotation equivariant pipeline, eliminating the requirement that data be pre-aligned. Rotation equivariance is particularly useful for applications of the ECT on data which has no natural orientation, or for which aligning is computationally expensive.

Potential directions to extend the ideas presented here include the use of CNNs to model directional transform data using choices of topological representation other than the ECT. So long as there is a reasonable way to represent the directional transform data as a 2D matrix, the same pipeline could be applied for classification. As described in Ch. 4, there are challenges associated with representing persistent homology transform data in this way, for example. The ECT-CNN

pipeline presented here is limited to 2D input data, however considering extensions to a less restricted class of input data is an interesting future direction. The primary roadblock of a direct generalization from 2D to 3D is not with the ECT computation itself, but in sampling directions on $\mathbb{S}^2$ in a regular grid pattern such that a 3D tensor can be constructed to be used as input to a CNN.

In Ch. 5 we apply the ECT-CNN framework to study the shape of leaves, comparing across two distinct leaf-shape datasets. Although the ECT-CNN model itself does not yield the best performance when compared to the traditional use of CNNs on the shape image representations, we do see that the overall best performing model for both datasets uses the ECT in some way. For the leaf graph dataset, the best performing model is the TDA combined with traditional method, using the two copies of the ECT image stacked with one copy of the shape image representation to get a 3-channel image. Its classification accuracy is very closely followed by the ECT-CNN alone. For the larger leaf outline dataset, we similarly see the best performance using both the ECT image and the outline image passed to the CNN for classification. Overall, comparison of these classification results show that we can boost performance of the traditional CNN by including topological measures of shape, specifically using the ECT. Also, the relative high performance of the ECT-CNN for this task verifies its utility for the classification of ECT data over traditionally used methods such as SVM models.

In additional to the applications presented here, future work may include further analysis of leaf shape datasets, including the unused data from [56]. These datasets contain leaf outlines collected from 75 different locations around the world, representing 141 different plant families and would provide a more diverse dataset to test the effectiveness of the ECT-CNN. Other applications to real data could include analyzing protein structure using a 3D ECT-CNN, representing protein backbones as embedded simplicial complexes $K \in \mathbb{R}^3$. While the ECT computation time and CNN model training times for the leaf shape datasets are not prohibitive, applying this method to 3D datasets will require work to parallelize the ECT computation code. The use of GPUs to speed up the training of CNN models may also be necessary for application to larger datasets.

As the field of topological deep learning grows, so too does the need for model architectures

designed to leverage the apparent topological information of their input data in a meaningful way. Future work in this space can use TDA tools to both understand logistics of existing models, and to develop new architectures. Both of which avenues have the potential to contribute to ongoing deep learning efforts towards model interpretability and explainability: understanding the behavior of a model's inner mechanics and how a model reaches a conclusion from given inputs.

BIBLIOGRAPHY

[1] Henry Adams, Tegan Emerson, Michael Kirby, Rachel Neville, Chris Peterson, Patrick Shipman, Sofya Chepushtanova, Eric Hanson, Francis Motta, and Lori Ziegelmeier. "Persistence Images: A Stable Vector Representation of Persistent Homology". In: *Journal of Machine Learning Research* 18.8 (2017), pp. 1–35.

[2] Erik J Amézquita, Michelle Y Quigley, Tim Ophelders, Jacob B Landis, Daniel Koenig, Elizabeth Munch, and Daniel H Chitwood. "Measuring hidden phenotype: quantifying the shape of barley seeds using the Euler characteristic transform". In: *in silico Plants* 4.1 (Dec. 2021), diab033.

[3] Davide Bacciu and Luigi Di Sotto. "A Non-negative Factorization Approach to Node Pooling in Graph Convolutional Neural Networks". In: *AI*IA 2019 – Advances in Artificial Intelligence*. Ed. by Mario Alviano, Gianluigi Greco, and Francesco Scarcello. Cham: Springer International Publishing, 2019, pp. 294–306.

[4] Danielle Barnes, Luis Polanco, and Jose A. Perea. "A Comparative Study of Machine Learning Methods for Persistence Diagrams". In: *Frontiers in Artificial Intelligence* 4 (2021).

[5] Yuliy Baryshnikov and Robert Ghrist. "Euler integration over definable functions". In: *Proceedings of the National Academy of Sciences* 107 (2009), pp. 9525–9530.

[6] Leo Betthauser. "Topological Reconstruction of Grayscale Images". In: *A Dissertation Presented to the Graduate School of the University of Florida in Partial Fufillment of the Requirements for the Degree of Doctor of Philosophy*. University of Florida. 2018.

[7] Filippo Maria Bianchi, Daniele Grattarola, and Cesare Alippi. "Spectral clustering with graph neural networks for graph pooling". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 874–883.

[8] Valerio Biscione and Jeffrey S. Bowers. "Convolutional neural networks are not invariant to translation, but they can learn to be". In: *Journal of Machine Learning Research (JMLR)* 22.1 (Jan. 2021).

[9] Cristian Bodnar, Cătălina Cangea, and Pietro Liò. "Deep graph mapper: Seeing graphs through the neural lens". In: *Frontiers in big Data* 4 (2021).

[10] Cristian Bodnar, Fabrizio Frasca, Nina Otter, Yuguang Wang, Pietro Lio, Guido F Montufar, and Michael Bronstein. "Weisfeiler and Lehman go cellular: CW networks". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 2625–2640.

[11] Cristian Bodnar, Fabrizio Frasca, Yuguang Wang, Nina Otter, Guido F Montufar, Pietro Lio, and Michael Bronstein. "Weisfeiler and Lehman go topological: Message passing

simplicial networks". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 1026–1037.

[12] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. "Spectral Networks and Locally Connected Networks on Graphs". In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2014.

[13] Peter Bubenik. "Statistical topological data analysis using persistence landscapes". In: *Journal of Machine Learning Research (JMLR)* 16.1 (Jan. 2015), pp. 77–102.

[14] Eric Bunch, Qian You, Glenn Fung, and Vikas Singh. "Simplicial 2-Complex Convolutional Neural Networks". In: *TDA & Beyond*. 2020.

[15] Cătălina Cangea, Petar Veličković, Nikola Jovanović, Thomas Kipf, and Pietro Liò. "Towards sparse hierarchical graph classifiers". In: *Workshop on Relational Representation Learning (R2L) at NIPS 2018 arXiv:1811.01287* (2018).

[16] Gunnar Carlsson and Rickard Brüel Gabrielsson. "Topological approaches to deep learning". In: *Topological Data Analysis: The Abel Symposium 2018*. Springer. 2020, pp. 119–146.

[17] Corrie J Carstens, Kathy J Horadam, et al. "Persistent homology of collaboration networks". In: *Mathematical problems in engineering* 2013 (2013).

[18] Frédéric Chazal, Brittany Terese Fasy, Fabrizio Lecci, Alessandro Rinaldo, and Larry Wasserman. "Stochastic Convergence of Persistence Landscapes and Silhouettes". In: *Proceedings of the Thirtieth Annual Symposium on Computational Geometry*. SOCG'14. Kyoto, Japan: Association for Computing Machinery, 2014, pp. 474–483.

[19] Ilya Chevyrev, Vidit Nanda, and Harald Oberhauser. "Persistence Paths and Signature Features in Topological Data Analysis". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42.1 (2020), pp. 192–202.

[20] Daniel H. Chitwood and Wagner C. Otoni. "Morphometric analysis of Passiflora leaves: the relationship between landmarks of the vasculature and elliptical Fourier descriptors of the blade". In: *GigaScience* 6.1 (Jan. 2017), giw008.

[21] Domenico Mattia Cinque, Claudio Battiloro, and Paolo Di Lorenzo. "Pooling Strategies for Simplicial Convolutional Networks". In: *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2023, pp. 1–5.

[22] Taco S. Cohen, Mario Geiger, Jonas Köhler, and Max Welling. "Spherical CNNs". In: *International Conference on Learning Representations*. 2018.

[23] David Cohen-Steiner, Herbert Edelsbrunner, and John Harer. "Stability of Persistence Diagrams". In: *Discrete & Computational Geometry* 37.1 (Jan. 2007), pp. 103–120.

[24] Lorin Crawford, Anthea Monod, Andrew X. Chen, Sayan Mukherjee, and Raúl Rabadán. "Predicting Clinical Outcomes in Glioblastoma: An Application of Topological and Functional Data Analysis". In: *Journal of the American Statistical Association* 115.531 (2020), pp. 1139–1150.

[25] Y. Le Cun, B. Boser, J. S. Denker, R. E. Howard, W. Habbard, L. D. Jackel, and D. Henderson. "Handwritten digit recognition with a back-propagation network". In: *Advances in Neural Information Processing Systems 2*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1990, pp. 396–404.

[26] Justin Curry, Sayan Mukherjee, and Katharine Turner. "How Many Directions Determine a Shape and other Sufficiency Results for Two Topological Transforms". In: *Transactions of the American Mathematical Society, Series B* 9 (Oct. 2022), pp. 1006–1043.

[27] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. "Convolutional neural networks on graphs with fast localized spectral filtering". In: *Proceedings of the 30th International Conference on Neural Information Processing Systems*. NIPS'16. Barcelona, Spain: Curran Associates Inc., 2016, pp. 3844–3852.

[28] Jozef Dodziuk. "Finite-difference approach to the Hodge theory of harmonic forms". In: *American Journal of Mathematics* 98.1 (1976), pp. 79–104.

[29] Stefania Ebli, Michaël Defferrard, and Gard Spreemann. *Simplicial Neural Networks*. arXiv: 2010.03633 [cs.LG]. 2020.

[30] Edelsbrunner, Letscher, and Zomorodian. "Topological persistence and simplification". In: *Discrete & computational geometry* 28 (2002), pp. 511–533.

[31] Brittany Terese Fasy, Samuel Micka, David L. Millman, Anna Schenfisch, and Lucia Williams. *Challenges in Reconstructing Shapes from Euler Characteristic Curves*. 28th Annual Fall Workshop on Computational Geometry, arXiv: 1811.11337 [cs.CG]. 2018.

[32] Brittany Terese Fasy, Samuel Micka, David L. Millman, Anna Schenfisch, and Lucia Williams. *A Faithful Discretization of the Verbose Persistent Homology Transform*. arXiv: 1912.12759 [cs.CG]. 2024.

[33] Hongyang Gao and Shuiwang Ji. "Graph u-nets". In: *International Conference on Machine Learning (ICML)*. PMLR. 2019, pp. 2083–2092.

[34] Chad Giusti, Robert Ghrist, and Danielle Bassett. "Two's company, three (or more) is a simplex: Algebraic-topological tools for understanding higher-order structure in neural data". In: *Journal of Computational Neuroscience* 41 (Jan. 2016).

[35] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[36] John Gower and G.B. Dijksterhuis. "Procrustes Problems". In: *Procrustes Problems, Oxford Statistical Science Series* Vol. 30 (Jan. 2005).

[37] Daniele Grattarola, Daniele Zambon, Filippo Maria Bianchi, and Cesare Alippi. "Understanding Pooling in Graph Neural Networks". In: *IEEE Transactions on Neural Networks and Learning Systems* 35.2 (2024), pp. 2708–2718.

[38] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. "Exploring Network Structure, Dynamics, and Function using NetworkX". in: *Proceedings of the 7th Python in Science Conference*. Ed. by Gaël Varoquaux, Travis Vaught, and Jarrod Millman. Pasadena, CA USA, 2008, pp. 11–15.

[39] Mustafa Hajij, Ghada Zamzmi, Theodore Papamarkou, Nina Miolane, Aldo Guzmán-Sáenz, Karthikeyan Natesan Ramamurthy, Tolga Birdal, Tamal K. Dey, Soham Mukherjee, Shreyas N. Samaga, Neal Livesay, Robin Walters, Paul Rosen, and Michael T. Schaub. *Topological Deep Learning: Going Beyond Graph Data*. arXiv:2206.00606 [cs.LG]. 2023.

[40] Yena Han, Gemma Roig, Gad Geiger, and Tomaso Poggio. "Scale and translation-invariance for novel objects in human vision". In: *Scientific reports* 10.1 (2020), p. 1411.

[41] Matthew Hirn. "Convolutional Neural Networks". Lecture Notes, Spring 2020 CMSE 890-002. 2020.

[42] Christoph Hofer, Roland Kwitt, Marc Niethammer, and Andreas Uhl. "Deep learning with topological signatures". In: *Advances in neural information processing systems* 30 (2017).

[43] Mattie Ji, Kun Meng, and Kexin Ding. *Euler Characteristics and Homotopy Types of Definable Sublevel Sets, with Applications to Topological Data Analysis*. arXiv:2309.03142 [math.AT]. 2023.

[44] Q. Jiang, S. Kurtek, and T. Needham. "The Weighted Euler Curve Transform for Shape and Image Analysis". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. Los Alamitos, CA, USA: IEEE Computer Society, June 2020, pp. 3685–3694.

[45] Alexandros D Keros, Vidit Nanda, and Kartic Subr. "Dist2Cycle: A Simplicial Neural Network for Homology Localization". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 36.7 (June 2022), pp. 7133–7142.

[46] Firas A Khasawneh and Elizabeth Munch. "Utilizing topological data analysis for studying signals of time-delay systems". In: *Time Delay Systems: Theory, Numerics, Applications, and Experiments* (2017), pp. 93–106.

[47] Jinpyo Kim, Wooekun Jung, Hyungmo Kim, and Jaejin Lee. *CyCNN: A Rotation Invariant CNN using Polar Mapping and Cylindrical Convolution Layers*. arXiv:2007.10588 [cs.CV]. 2020.

[48] Thomas N. Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks". In: *International Conference on Learning Representations*. 2017.

[49] Boris Knyazev, Graham W Taylor, and Mohamed Amer. "Understanding attention and generalization in graph neural networks". In: *Advances in neural information processing systems* 32 (2019).

[50] Violeta Kovacev-Nikolic, Peter Bubenik, Dragan Nikolić, and Giseon Heo. "Using persistent homology and dynamical distances to analyze protein binding." eng. In: *Stat Appl Genet Mol Biol* 15.1 (Mar. 2016), pp. 19–38.

[51] V.A Kovalevsky. "Finite topology as applied to image analysis". In: *Computer Vision, Graphics, and Image Processing* 46.2 (1989), pp. 141–161.

[52] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012).

[53] Henry Kvinge, Tegan Emerson, Grayson Jorgenson, Scott Vasquez, Timothy Doster, and Jesse Lew. "In What Ways Are Deep Neural Networks Invariant and How Should We Measure This?" In: *Advances in Neural Information Processing Systems*. Ed. by Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho. 2022.

[54] L.J. Latecki, R. Lakamper, and T. Eckhardt. "Shape descriptors for non-rigid shapes with a single closed contour". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000 (Cat. No.PR00662)*. Vol. 1. 2000, 424–429 vol.1.

[55] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. "Self-attention graph pooling". In: *International conference on machine learning*. PMLR. 2019, pp. 3734–3743.

[56] Mao Li, Hong An, Ruthie Angelovici, Clement Bagaza, Albert Batushansky, Lynn Clark, Viktoriya Coneva, Michael J. Donoghue, Erika Edwards, Diego Fajardo, Hui Fang, Margaret H. Frank, Timothy Gallaher, Sarah Gebken, Theresa Hill, Shelley Jansky, Baljinder Kaur, Phillip C. Klahs, Laura L. Klein, Vasu Kuraparthy, Jason Londo, Zoë Migicovsky, Allison Miller, Rebekah Mohn, Sean Myles, Wagner C. Otoni, J. C. Pires, Edmond Rieffer, Sam Schmerler, Elizabeth Spriggs, Christopher N. Topp, Allen Van Deynze, Kuang Zhang, Linglong Zhu, Braden M. Zink, and Daniel H. Chitwood. "Topological Data Analysis as a Morphometric Method: Using Persistent Homology to Demarcate a Leaf Morphospace". In: *Frontiers in Plant Science* 9 (2018).

[57] Lek-Heng Lim. "Hodge Laplacians on graphs". In: *SIAM Review* 62.3 (2020), pp. 685–

715.

[58] Louis-David Lord, Paul Expert, Henrique M. Fernandes, Giovanni Petri, Tim J. Van Hartevelt, Francesco Vaccarino, Gustavo Deco, Federico Turkheimer, and Morten L. Kringelbach. "Insights into Brain Architectures from the Homological Scaffolds of Functional Connectivity Networks". In: *Frontiers in Systems Neuroscience* 10 (2016).

[59] Enxhell Luzhnica, Ben Day, and Pietro Lio'. *Clique pooling for graph classification*. arXiv:1904.00374 [cs.LG]. 2019.

[60] Slobodan Maletić, Yi Zhao, and Milan Rajković. "Persistent topological features of dynamical systems". In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 26.5 (May 2016).

[61] Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. "Invariant and Equivariant Graph Networks". In: *International Conference on Learning Representations*. 2019.

[62] Lewis Marsh and David Beers. *Stability and Inference of the Euler Characteristic Transform*. arXiv:2303.13200 [math.ST]. 2023.

[63] Lewis Marsh, Felix Y. Zhou, Xiao Qin, Xin Lu, Helen M. Byrne, and Heather A. Harrington. *Detecting Temporal shape changes with the Euler Characteristic Transform*. arXiv:2212.10883 [q-bio.QM]. 2022.

[64] Sarah McGuire, Elizabeth Munch, and Matthew Hirn. *NervePool: A Simplicial Pooling Layer*. arXiv:2305.06315 [cs.CG]. 2023.

[65] Kun Meng, Jinyu Wang, Lorin Crawford, and Ani Eloyan. *Randomness of Shapes and Statistical Inference on Shapes via the Smooth Euler Characteristic Transform*. arXiv:2204.12699 [stat.ME]. 2024.

[66] Elizabeth Munch. *An Invitation to the Euler Characteristic Transform*. arXiv:2310.10395 [cs.CG]. 2023.

[67] James R Munkres. *Elements of algebraic topology*. CRC press, 1984.

[68] Emmanuel Noutahi, Dominique Beaini, Julien Horwood, Sébastien Giguère, and Prudencio Tossou. *Towards Interpretable Sparse Graph Representation Learning with Laplacian Pooling*. arXiv:1905.11577 [cs.LG]. 2020.

[69] Taejin Paik. *Invariant Representations of Embedded Simplicial Complexes*. arXiv:2302.13565 [cs.LG]. 2023.

[70] Siddharth Pal, Terrence J Moore, Ram Ramanathan, and Ananthram Swami. "Comparative topological signatures of growing collaboration networks". In: *Complex Networks VIII:*

Proceedings of the 8th Conference on Complex Networks CompleNet 2017 8. Springer. 2017, pp. 201–209.

[71] Theodore Papamarkou, Tolga Birdal, Michael Bronstein, Gunnar Carlsson, Justin Curry, Yue Gao, Mustafa Hajij, Roland Kwitt, Pietro Liò, Paolo Di Lorenzo, Vasileios Maroulas, Nina Miolane, Farzana Nasrin, Karthikeyan Natesan Ramamurthy, Bastian Rieck, Simone Scardapane, Michael T. Schaub, Petar Veličković, Bei Wang, Yusu Wang, Guo-Wei Wei, and Ghada Zamzmi. *Position Paper: Challenges and Opportunities in Topological Deep Learning*. arXiv:2402.08871 [cs.LG]. 2024.

[72] Mathilde Papillon, Sophia Sanborn, Mustafa Hajij, and Nina Miolane. *Architectures of Topological Deep Learning: A Survey of Message-Passing Topological Neural Networks*. arXiv:2304.10031 [cs.LG]. 2024.

[73] Jose A. Perea, Anastasia Deckard, Steve B. Haase, and John Harer. "SW1PerS: Sliding windows and 1-persistence scoring; discovering periodicity in gene expression time series data". In: *BMC Bioinformatics* 16.1 (Aug. 2015), p. 257.

[74] Jose A. Perea, Elizabeth Munch, and Firas A. Khasawneh. "Approximating Continuous Functions on Persistence Diagrams Using Template Functions". In: *Foundations of Computational Mathematics* 23.4 (2023), pp. 1215–1272.

[75] Georges Reeb. "Sur les points singuliers d'une forme de Pfaff complèment intégrable ou d'une fonction numérique". In: *Comptes Rendus de L'Académie ses Séances* 222 (1946), pp. 847–849.

[76] T. Mitchell Roddenberry, Nicholas Glaze, and Santiago Segarra. "Principled Simplicial Neural Networks for Trajectory Prediction". In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, July 2021, pp. 9020–9029.

[77] T. Mitchell Roddenberry, Michael T. Schaub, and Mustafa Hajij. "Signal Processing On Cell Complexes". In: *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2022, pp. 8852–8856.

[78] Ernst Röell and Basitan Rieck. "Differentiable Euler Characteristic Transforms for Shape Classification". In: *The Twelfth International Conference on Learning Representations*. 2024.

[79] Stefania Sardellitti, Sergio Barbarossa, and Lucia Testa. "Topological Signal Processing over Cell Complexes". In: *2021 55th Asilomar Conference on Signals, Systems, and Computers*. 2021, pp. 1558–1562.

[80] Michael T. Schaub, Yu Zhu, Jean-Baptiste Seby, T. Mitchell Roddenberry, and Santiago Segarra. "Signal processing on higher-order networks: Livin' on the edge... and beyond".

In: *Signal Processing* 187 (2021), p. 108149.

[81] T. Sikora. "The MPEG-7 visual standard for content description-an overview". In: *IEEE Transactions on Circuits and Systems for Video Technology* 11.6 (2001), pp. 696–702.

[82] Gurjeet Singh, Facundo Mémoli, and Gunnar E Carlsson. "Topological methods for the analysis of high dimensional data sets and 3d object recognition." In: *PBG@ Eurographics* 2 (2007).

[83] Ann Sizemore, Chad Giusti, and Danielle S. Bassett. "Classification of weighted networks through mesoscale homological features". In: *Journal of Complex Networks* 5.2 (Aug. 2016), pp. 245–273.

[84] Katharine Turner, Sayan Mukherjee, and Doug M. Boyer. "Persistent homology transform for modeling shapes and surfaces". In: *Information and Inference: A Journal of the IMA* 3.4 (2014), pp. 310–344.

[85] Sarah Tymochko, Elizabeth Munch, Jason Dunion, Kristen Corbosiero, and Ryan Torn. "Using persistent homology to quantify a diurnal cycle in hurricanes". In: *Pattern Recognition Letters* 133 (2020), pp. 137–143.

[86] Sarah Tymochko, Elizabeth Munch, and Firas A. Khasawneh. "Using Zigzag Persistent Homology to Detect Hopf Bifurcations in Dynamical Systems". In: *Algorithms* 13.11 (2020).

[87] Kalyan Varma Nadimpalli, Amit Chattopadhyay, and Bastian Rieck. "Euler Characteristic Transform Based Topological Loss for Reconstructing 3D Images from Single 2D Slices". In: *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2023, pp. 571–579.

[88] Bruce Wang, Timothy Sudijono, Henry Kirveslahti, Tingran Gao, Douglas M. Boyer, Sayan Mukherjee, and Lorin Crawford. "A statistical pipeline for identifying physical features that differentiate classes of 3D shapes". In: *The Annals of Applied Statistics* 15.2 (2021), pp. 638–661.

[89] André Weil. "Sur les théoremes de de Rham". In: *Comment. Math. Helv* 26.1 (1952), pp. 119–145.

[90] Boris Weisfeiler and Andrei Leman. "The reduction of a graph to canonical form and the algebra which appears therein". In: *NTI, Series* 2.9 (1968), pp. 12–16.

[91] Nathan Willey. *ECT of Leaves*. https://github.com/willeyna/ECT_of_leaves. 2023.

[92] Kelin Xia and Guo-Wei Wei. "Persistent homology analysis of protein structure, flexibility, and folding". In: *International journal for numerical methods in biomedical engineering*

30.8 (2014), pp. 814–844.

[93] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. "How Powerful are Graph Neural Networks?" In: *International Conference on Learning Representations*. 2019.

[94] Maosheng Yang and Elvin Isufi. *Convolutional Learning on Simplicial Complexes*. arXiv:2301.11163 [cs.LG]. 2023.

[95] Maosheng Yang, Elvin Isufi, and Geert Leus. "Simplicial Convolutional Neural Networks". In: *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2022, pp. 8847–8851.

[96] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. "Hierarchical graph representation learning with differentiable pooling". In: *Advances in neural information processing systems* 31 (2018).

[97] Jaejun Yoo, Eun Young Kim, Yong Min Ahn, and Jong Chul Ye. "Topological persistence vineyard for dynamic functional brain connectivity during resting and gaming stages". In: *Journal of Neuroscience Methods* 267 (2016), pp. 1–13.

[98] Hao Yuan and Shuiwang Ji. "StructPool: Structured Graph Pooling via Conditional Random Fields". In: *International Conference on Learning Representations*. 2020.

[99] Ali Zia, Abdelwahed Khamis, James Nichols, Usman Bashir Tayab, Zeeshan Hayder, Vivien Rolland, Eric Stone, and Lars Petersson. "Topological deep learning: a review of an emerging paradigm". In: *Artificial Intelligence Review* 57.4 (Feb. 2024), p. 77.

[100] Afra Zomorodian and Gunnar Carlsson. "Computing persistent homology". In: *Proceedings of the twentieth annual symposium on Computational geometry*. 2004, pp. 347–356.