University of São Paulo
Institute of Mathematics, Statistics and Computer Science
Bachelor of Computer Science

An Introduction to Geometric Deep Learning on Sets, Graphs, and Grids

Gabriel Jacob Perin

Final Essay

mac 499 — Capstone Project

Supervisor: Prof.ª Dr.ª Nina S. T. Hirata

São Paulo
2025

The purpose of abstracting is not to be vague, but to create a new semantic level in which one can be absolutely precise.
— Edsger W. Dijkstra

Acknowledgements

First, I thank my advisor, Professor Nina S. T. Hirata, for her guidance, patience, and support throughout my undergraduate studies. She played a fundamental role in introducing me to Machine Learning and in shaping my academic path.

I am deeply grateful to the many mentors and collaborators I met through USP, UT Austin, and IBM Research. From each of you, I learned something meaningful. I also thank the members of the L.E.A.R.N. group for the stimulating discussions and shared curiosity that kept me excited about learning.

To my friends from BCC: thank you for the study sessions, the shared coffees, the drinks, and the valuable lesson that challenging problems are far more enjoyable when faced together.

To my lifelong friends from Pelotas–RS: even with distance, your presence has never felt far. Thank you for grounding me and for giving me space to pause when I needed it.

Finally, my deepest gratitude goes to my parents. Your unwavering support, patience, and belief in me made this journey possible. I am endlessly thankful.

To all of you — from the bottom of my heart — thank you.

Resumo

Gabriel Jacob Perin. **Uma Introdução ao Aprendizado Profundo Geométrico em Conjuntos, Grafos e Grades**. Monografia (Bacharelado). Instituto de Matemática, Estatística e Ciência da Computação, Universidade de São Paulo, São Paulo, 2025.

O Aprendizado Profundo tornou-se um paradigma dominante na inteligência artificial moderna, alcançando resultados expressivos em diversos domínios. No entanto, grande parte de seu desenvolvimento permanece empírica, em vez de fundamentada teoricamente. O Aprendizado Profundo Geométrico (ou *Geometric Deep Learning*, GDL) surge como um arcabouço conceitual que permite compreender e projetar arquiteturas neurais por meio de simetrias. Este trabalho busca tornar o GDL acessível em nível de graduação ao simplificar seus fundamentos teóricos e ao focar em três domínios geométricos centrais: Conjuntos, Grafos e Grades. Em vez de apresentar uma revisão exaustiva da literatura, o texto enfatiza intuição, ferramentas matemáticas essenciais e exemplos claros que conectam estrutura geométrica ao projeto de redes neurais. O leitor é introduzido aos conceitos fundamentais de aprendizagem de máquina, ao papel dos vieses geométricos e ao *GDL blueprint* antes de explorar como arquiteturas canônicas, como *DeepSets*, *Graph Neural Networks* e *Convolutional Neural Networks*, emergem naturalmente desses princípios. Por fim, são apresentados experimentos comparando modelos que preservam simetrias com aproximadores universais, destacando o impacto dos vieses geométricos no comportamento e desempenho dos modelos.

Palavras-chave: Aprendizado de Máquina. Aprendizado Profundo. Aprendizado Profundo Geométrico.

Abstract

Gabriel Jacob Perin. **An Introduction to Geometric Deep Learning on Sets, Graphs, and Grids**. Capstone Project Report (Bachelor). Institute of Mathematics, Statistics and Computer Science, University of São Paulo, São Paulo, 2025.

Deep Learning has become a dominant paradigm in modern artificial intelligence, achieving impressive results across many domains, yet much of its development remains empirical rather than theoretically grounded. Geometric Deep Learning (GDL) offers a principled framework to understand and design neural architectures through symmetries. This work aims to make GDL accessible at the undergraduate level by simplifying its theoretical foundations and focusing on three core geometric domains: Sets, Graphs, and Grids. Rather than presenting an exhaustive survey, the work emphasizes intuition, essential mathematical tools, and clear examples that connect geometric structure to neural network design. The reader is introduced to the foundational concepts of Machine Learning, the role of geometric priors, and the GDL blueprint before exploring how canonical architectures such as DeepSets, Graph Neural Networks, and Convolutional Neural Networks naturally emerge from these principles. Finally, the thesis presents experiments comparing symmetry-aware models with universal approximators, highlighting the impact of geometric inductive biases on model behavior and performance.

Keywords: Machine Learning. Deep Learning. Geometric Deep Learning.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

In recent years, Deep Learning has become one of the most influential and widely adopted paradigms in artificial intelligence. Its success spans numerous domains, including computer vision, natural language processing, robotics, and scientific modeling. As models grow larger and are increasingly deployed in mission-critical applications, there is a growing need to better understand why certain architectures work, how they generalize, and how they can be designed more systematically. Despite its remarkable empirical performance, Deep Learning has largely evolved through experimentation rather than theory, often relying on heuristics, architectural trial-and-error, and massive computational resources. This empirical foundation has led to an ecosystem where many architectures appear disparate, ad hoc, or domain-specific, without an obvious unifying principle.

Geometric Deep Learning (GDL) emerged as an attempt to bring structure and theoretical grounding to this landscape (Michael M Bronstein *et al.*, 2021). Rather than treating neural networks as black boxes, GDL provides a principled framework for designing architectures by analyzing the underlying geometric structure of data (Michael M Bronstein *et al.*, 2021). Through concepts such as symmetry, invariance, and equivariance, GDL categorizes existing models and guides the creation of new ones in a way that aligns model behavior with the domain it operates on (Michael M Bronstein *et al.*, 2021). These principles are increasingly relevant in areas such as molecular modeling and physical simulation, domains where respecting structure is central to performance, robustness, and interpretability (Michael M Bronstein *et al.*, 2021).

However, fully approaching GDL can be challenging. The field sits at the intersection of mathematics, theoretical computer science, and Machine Learning, involving concepts such as group theory, topology, and representation theory, areas not typically included in a standard undergraduate computer science curriculum (Borde and M. Bronstein, 2025). The goal of this thesis is to bridge that gap. By restricting the scope of GDL to three core geometric domains, namely Sets, Graphs, and Grids, we simplify the theoretical landscape while retaining the essential ideas. Rather than providing a comprehensive survey of the field, our emphasis is on intuition, exemplification, and key theoretical results that describe how symmetry informs architectural design.

This monograph is intended to be accessible to undergraduate students with introduc-

tory knowledge of Machine Learning and Deep Learning. It is largely self-contained and minimizes the need for advanced mathematical prerequisites. By the end of this work, the reader should understand why symmetry matters in learning systems, how to formally analyze invariances and equivariances, and how widely used models such as DeepSets, Graph Neural Networks, and Convolutional Neural Networks arise naturally from geometric considerations (Michael M Bronstein *et al.*, 2021; Zaheer *et al.*, 2017; Xu *et al.*, 2018).

The first chapter, Fundamentals, introduces the background required for the remainder of the work. We begin with a brief overview of Machine Learning, followed by the principles and formalization of Supervised Learning. We conclude by discussing the curse of dimensionality and why high-dimensional learning requires structure and constraints.

The next chapter, Geometric Priors, introduces the foundations of GDL. We define the notion of a signal over a geometric domain, explore the role of symmetries and scale separation, and examine how these concepts help mitigate the curse of dimensionality. Finally, we present the GDL Blueprint: a general scheme for constructing neural network architectures grounded in geometric principles.

The following three chapters instantiate the blueprint in concrete domains. We begin with Sets, the simplest geometric objects. By studying permutation symmetry, we show how it naturally leads to the DeepSets architecture (Zaheer *et al.*, 2017). Next, we examine Graphs, where data elements are connected by relational structure. We again study permutation symmetry in this context and show how it gives rise to Graph Neural Networks (GNNs) (Michael M Bronstein *et al.*, 2021). Remarkably, we also observe that the widely used Transformer architecture can be understood as a special case of a GNN (Vaswani *et al.*, 2017; Joshi, 2020). We conclude with Grids, a structured subclass of graphs commonly used to represent images. Here, translation symmetry leads to Convolutional Neural Networks (CNNs) as a natural consequence of the GDL framework (LeCun, Boser, *et al.*, 1989; Michael M Bronstein *et al.*, 2021).

Finally, the GDL in Action chapter transitions from theory to practice. We implement the architectures studied throughout the thesis and compare them with universal approximators that do not encode geometric priors. Through experiments, we empirically examine how the presence or absence of geometric inductive biases influences predictive behavior.

Chapter 2

Fundamentals

This chapter introduces foundational concepts in Machine Learning. In the first section, we provide a brief overview of the field. The second section defines the core principles of Supervised Learning. Finally, the third section discusses the curse of dimensionality and highlights the difficulties of approximating arbitrary functions in high-dimensional spaces. The subsequent chapters demonstrate how the **Geometric Deep Learning** framework informs and inspires the design of neural network architectures aimed at overcoming these challenges.

2.1 Learning from Data

The human mind is remarkably adept at solving complex problems. People can recognize faces in a crowd, understand spoken language, and predict physical events, often without relying on *formal definitions* or *rigid rules*. We perform such tasks by learning from experience, generalizing from the examples we observe in everyday life.

Machine Learning (ML) is a field of computer science that enables computers to tackle complex tasks without being explicitly programmed with task-specific rules. Instead of hand-crafting algorithms for each problem, ML provides techniques that *leverage data* to automatically discover effective empirical solutions. By analyzing large datasets, these methods can learn patterns, make predictions, and adapt to new information, much like humans do when learning from examples.

This premise encompasses a wide range of approaches, giving rise to multiple machine learning paradigms, each defined by the nature of the data and the learning objective. Among these, perhaps the most widely studied—and the one explored in this work—is **Supervised Learning** (SL). In Supervised Learning, the dataset includes explicit information about the desired output for each input. For example, consider building a classification system that distinguishes between images of cats and dogs. In this setting, Supervised Learning can be applied if each image in the training set is labeled as either a cat or a dog.

In contrast, **Unsupervised Learning** (UL) assumes that the dataset contains only input data, with no associated output labels. Instead of learning from known outcomes,

UL algorithms attempt to discover patterns or structures in the data. A common approach is clustering, where the algorithm groups similar inputs together under the assumption that items within the same group share some underlying characteristic.

A third major paradigm is **Reinforcement Learning** (RL). Like UL, it operates without explicit input-output pairs. However, instead of trying to discover structure, RL algorithms interact with an environment where certain actions yield rewards. The data includes feedback in the form of reward signals, indicating how good a particular action was in a given situation. The learning objective is to discover a policy, *i. e.* a mapping from states (inputs) to actions (outputs), that maximizes the reward.

Across all these paradigms, advances in computational power, particularly the widespread availability of GPUs, have been a key enabler of the success of **Deep Learning** (DL). Deep learning refers to the use of deep neural networks, which are neural architectures composed of many layers, capable of modeling highly complex and flexible functions. These models are typically trained using gradient-based optimization methods, which adjust the model parameters to minimize prediction error. Deep learning has achieved remarkable success across a wide range of applications, and is widely regarded as one of the main driving forces behind the recent surge in interest and progress in artificial intelligence.

One of the key drivers of this progress has been the development of new Deep Learning architectures that are easier to train and better suited to specific types of data. For example, the introduction of Convolutional Neural Networks (CNNs) by LeCun, Boser, *et al.* (1989), and future enhancements in architectures such as LeNet by LeCun, Bottou, *et al.* (2002), enabled deep learning models to achieve unprecedented performance on previously intractable tasks. A landmark achievement was the success of AlexNet by Krizhevsky *et al.* (2012) in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) Russakovsky *et al.* (2015), which marked a turning point for computer vision and Deep Learning more broadly.

In this context, architectural research in Deep Learning has largely been driven by empirical experimentation and inspiration from other scientific fields. While many architectures have been developed through trial and error, there have also been efforts to establish unified theoretical frameworks from which various architectures could be systematically derived. Notable examples include Topological Deep Learning Hajij *et al.* (2022) and Categorical Deep Learning Gavranović *et al.* (2024), which aim to formalize the design of neural networks using concepts from topology and category theory, respectively. This work focuses on one of the most basic and influential of these unifying efforts: **Geometric Deep Learning** (GDL).

Geometric Deep Learning is a framework that characterizes Deep Learning architectures based on the symmetries they are designed to preserve. By formalizing how neural networks respect these symmetries in data, GDL provides a unifying perspective that explains the success of many modern architectures (e.g., CNNs, Transformers) and enables their generalization to more complex domains, such as graphs, manifolds, and point clouds.

2.2 The Principles of Supervised Learning

We now introduce the main components of Supervised Learning. The contents of this chapter closely follow Michael M Bronstein *et al.* (2021), Abu-Mostafa *et al.* (2012) and Michael M. Bronstein (2021).

Supervised Learning in its simplest formalization assumes access to a *dataset* $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ of samples drawn *i. i. d.* from an underlying data distribution P defined over $\mathcal{X} \times \mathcal{Y}$, where $\mathcal{X}$ is the input space and $\mathcal{Y}$ is the output (labels) space. We may further assume that there exists an unknown function $f^* : \mathcal{X} \to \mathcal{Y}$ such that $f^*(x_i) = y_i$ for all $i = 1, ..., N$.

Given a set $\mathcal{F}$ of functions that map $\mathcal{X}$ to $\mathcal{Y}$ (often referred to as the *Hypothesis Set*), which is organized in terms of a *complexity measure* $c : \mathcal{F} \to \mathbb{R}_+$, the learning problem is to find $f \in \mathcal{F}$ that *better approximates* f^*.

To solve this problem, it is necessary to properly define this notion of approximation. For this, assume access to a point-wise loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$ and define the *Population Loss* $L_P : \mathcal{F} \to \mathbb{R}_+$ such that

$$L_P(f) := \mathbb{E}_{(x,y)\sim P}[\ell\ (f(x), y)],$$

and an *Empirical Loss* $L_\mathcal{D} : \mathcal{F} \to \mathbb{R}_+$ such that

$$L_\mathcal{D}(f) := \frac{1}{N} \sum_{(x_i,y_i)\in\mathcal{D}} \ell\ (f(x_i), y_i).$$

Intuitively, $L_\mathcal{D}$ measures how well f approximates f^* with respect to the samples available in the dataset, while L_P measures how well this approximation is performed regarding the whole data distribution. In an ideal world, we would be able to choose f that minimizes L_P and get a best possible approximation for f^*. However, clearly, L_P is not accessible in practice. Thus, the best one can do is to minimize $L_\mathcal{D}$, in the hopes of finding a function that also minimizes L_P. This leads to one of the core principles behind SL: **Empirical Risk Minimization**.

The idea of Empirical Risk Minimization is to choose an hypothesis $\hat{f}$ that minimizes the Empirical Loss $L_\mathcal{D}$, often taking the complexity measure into account. There are many formulation that seek to achieve this goal, such as the *Penalized form*

$$\hat{f} = \arg\min_{f\in\mathcal{F}} L_\mathcal{D}(f) + \lambda c(f),$$

where $\lambda \in \mathbb{R}$ is called the *regularization coefficient*, or the *Restricted form*

$$\hat{f} = \arg\min_{f\in\mathcal{F}_\rho} L_\mathcal{D}(f),$$

where $\mathcal{F}_\rho := \{f \in \mathcal{F} \text{ s. t. } c(f) \leq \rho\}$ and $\rho \in \mathbb{R}$ is a complexity threshold.

Naturally, although performing well on the examples in $\mathcal{D}$, $\hat{f}$ may end up with high populational error. This phenomenon is called *overfitting*, and can be caused by different

reasons. When overfitting does not occur (*i. e.* $\hat{f}$ has low population and empirical loss), it is said that $\hat{f}$ *generalizes well*.

To illustrate these ideas, consider solving a regression problem from $\mathcal{X} = \mathbb{R}^d$ to $\mathcal{Y} = \mathbb{R}$ with a *Shallow Neural Network*. In this setting, our hypothesis class is

$$\mathcal{F} = \{f_{A,b,w}(x) = w^\top \theta(Ax + b) \mid A \in \mathbb{R}^{m \times d}, b, w \in \mathbb{R}^m\},$$

where $\theta : \mathbb{R}^m \longrightarrow \mathbb{R}^m$ is a point-wise non-linear function. Possible complexity measures for this set are the L_1-norm,

$$c(f_{A,b,w}) = \sum_{i \in [m]} \sum_{j \in [d]} |A_{ij}| + \sum_{i \in [m]} |b_i| + \sum_{i \in [m]} |w_i|,$$

or the L_2-norm,

$$c(f_{A,b,w}) = \sum_{i \in [m]} \sum_{j \in [d]} A_{ij}^2 + \sum_{i \in [m]} b_i^2 + \sum_{i \in [m]} w_i^2,$$

where $[n]$ denotes the set of integers ranging from 1 to n, i. e. $[n] = \{1, ..., n\}$. Finally, possible loss functions are the *absolute error* $\ell\ (\hat{y}, y) = |\hat{y} - y|$ and the *squared error* $\ell\ (\hat{y}, y) = (\hat{y} - y)^2$. The components of the learning process are summarized in Figure 2.1.

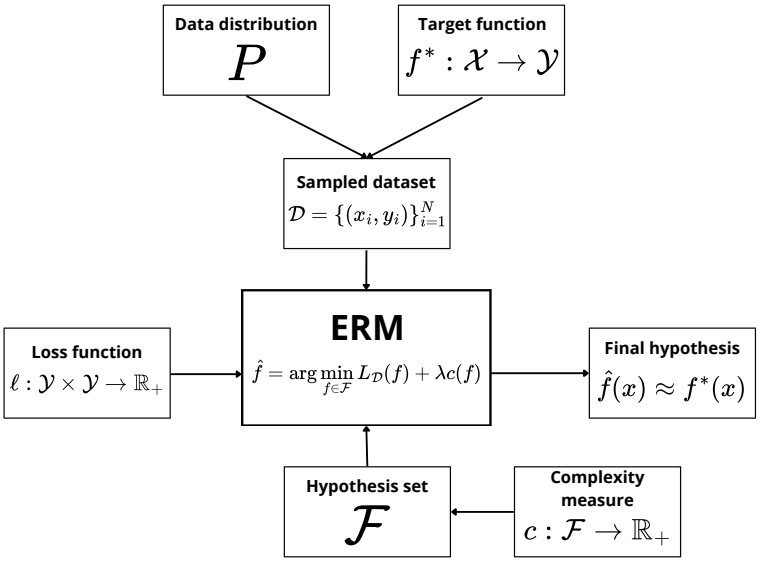

Figure 2.1: *Components of the Supervised Learning formalization.*

2.3 The Curse of Dimensionality

A natural question that arises is *how many* samples are required for the ERM to yield a good approximation of f^*. Although this answer may depend on many factors, it is known that this number does not scale well with the number of dimensions of $\mathcal{X}$. To grasp an

understanding of this fact, consider the expected distance of two points X, Y sampled independently in a unit cube of dimension d. It is known that $\mathbb{E}[\|X - Y\|] \geq \frac{\sqrt{d}}{3}$, for all $d \in \mathbb{N}$ (Weisstein, 2025). Therefore, the expected distance grows at least as fast as $\sqrt{d}$.

In other words, high-dimensional datasets are at a risk of being very sparse and thus, leading to poor generalization. Figure 2.2 illustrates why: when data is dense, the learning algorithm receives enough evidence to constrain the true underlying function. The left plot shows that many observations force any reasonable interpolating function to resemble the true sine wave. In contrast, sparse datasets impose weak constraints: many distinct functions are consistent with the few observed points. As shown in the right plot, the same small set of samples can be interpolated by multiple drastically different functions, all achieving zero training error.

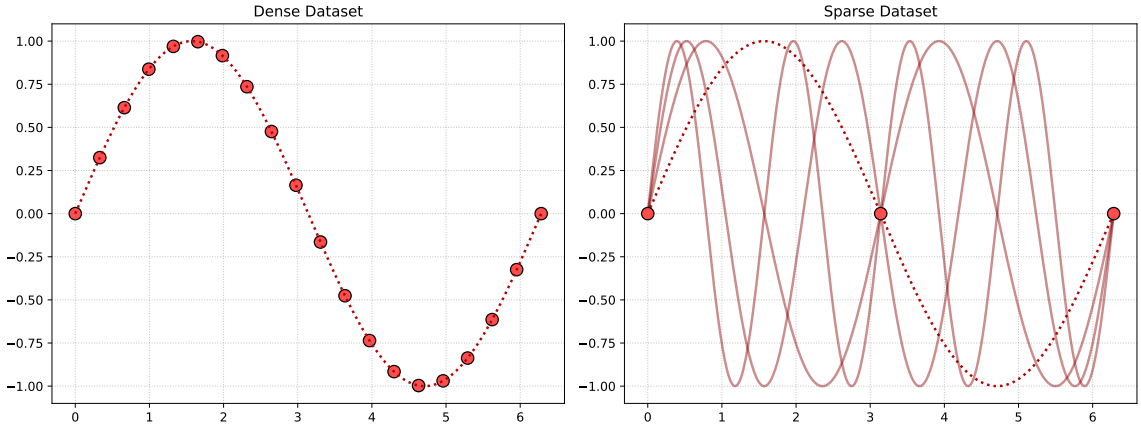

Figure 2.2: *When many samples are available (left), only one sine function can match the data, resulting in reliable inference. However, with very few samples (right), many different functions fit the observations equally well, making the true underlying function impossible to recover from the data alone.*

This phenomenon is commonly called *The Curse of Dimensionality*, and is a well studied phenomenon in the field of ML. There are many attempts to try to circumvent this problem, such as dimensionality reduction (for example, Principal Component Analysis (Jolliffe and Cadima, 2016)). However, such techniques assume data can be described as a combination of lower dimensional structures, which is not true in many real world applications.

In DL, an alternative approach for circumventing this problem is to use *inductive biases*, *i. e.* to encode domain knowledge in the Hypothesis Set, in a way that this knowledge is not required to be learned from the data. This allows achieving better generalization, with fewer samples. Geometric Deep Learning proposes to exploit the geometric properties of the data to restrict the Hypothesis Set according to the *inductive biases* observed in the domain application.

In the next chapter, the main geometric ideas will be introduced, and a *blueprint* for designing such Hypothesis Sets will be presented.

Chapter 3

Geometric Priors

The content of this chapter closely follows Michael M Bronstein *et al.* (2021), Borde and M. Bronstein (2025) and Michael M. Bronstein (2021). A *prior* is any assumption about the application domain that is built directly into the model architecture, ensuring that the model does not need to learn this property solely from data. Within the context of Geometric Deep Learning (GDL), two fundamental types of priors are particularly important: symmetries and scale separation.

Intuitively, symmetries are transformations that leave an object unchanged. In GDL, architectures are designed to respect such symmetries, thereby constraining the hypothesis space to functions consistent with them. Scale separation, in contrast, embodies the principle that phenomenon occurring at different scales can be modeled independently. By incorporating this assumption, models can simplify learning and effectively reduce the dimensionality of the problem.

In this chapter, we will examine each of these priors in detail and, at the end, show how they come together in a general blueprint for embedding inductive biases into neural network architectures.

3.1 Signals

We start by formally defining a data point. Usually, $x \in \mathcal{X}$ is thought of as a vector or a matrix. However, this view does not fully capture the geometric properties of the data. Instead, we adopt the notion of a *signal*.

Definition 3.1 (Signal)**.** *Let Ω be a finite set and C a vector space of dimension d (whose dimensions are called* channels*). A* ***signal*** *on Ω is a function*

$$x : \Omega \longrightarrow C.$$

For signals that share the same domain Ω, the following operations are naturally defined by leveraging the vector space structure of C:

1. *Addition:* $(x + y)(u) = x(u) + y(u), \quad \forall\, u \in \Omega,$

2. *Scalar multiplication:* $(\alpha x)(u) = \alpha\, x(u), \quad \forall\, u \in \Omega,\ \alpha \in \mathbb{R},$

3. *Inner product:* $\langle x, y \rangle = \sum_{u \in \Omega} \langle x(u), y(u) \rangle$.

We note that in Michael M Bronstein *et al.*, 2021, the domain Ω is treated as an arbitrary set, and the inner product is expressed as an integral to accommodate continuous spaces. In our setting, however, Ω is assumed to be finite (sufficient for the domains relevant to this work) and therefore discrete. Under this assumption, the integral naturally reduces to a finite summation.

The core motivation for this definition is to decouple the *geometric structure* of the data (represented by Ω) from its *features* (represented by $\mathcal{C}$). Although Ω is not required to be anything more than a finite set, interesting properties emerge when it is endowed with additional structure.

As an example, taking $\Omega = [n] \times [n]$ and $\mathcal{C} = \mathbb{R}^3$ yields $n \times n$ RGB images. It is also important to distinguish between collections of signals where Ω is fixed and those where Ω may vary. In the latter case, each signal can live on a different domain. For instance, molecules can be modeled as a collection of signals on the vertices of graphs, where each molecule corresponds to a signal defined on its own underlying graph. Figure 3.1 illustrates this distinction.

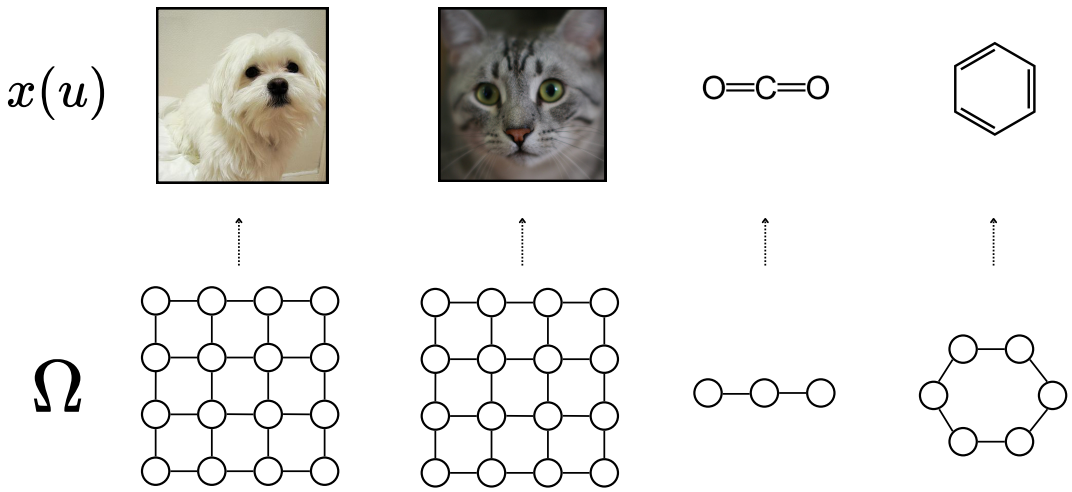

Figure 3.1: *Illustration of signals defined on different domains. The first two examples are cat and dog images represented on a regular 2-D grid, thus sharing the same underlying domain. In contrast, the last two examples correspond to carbon dioxide and benzene molecules, where the signals are defined over distinct graph structures, and therefore do not share the same domain. Cat and dog images are obtained from ImageNet (Deng et al., 2009).*

3.2 Symmetry

Intuitively, a symmetry is a transformation that preserves the essential information of an object — for example, reflecting an image of a dog still yields an image recognizable as the same dog. When working with signals defined over a domain Ω, we are particularly interested in transformations that preserve the structure of that domain.

The notion of symmetry is inherently task-dependent. For instance, while reflection may be a valid symmetry when classifying animals, it would not apply to road sign recognition, where a mirrored image could alter the sign's meaning. To study such transformations in a principled way, we employ the language of groups, which provide the formal framework for describing symmetries.

Definition 3.2 (Group)**.** *A* group *is an ordered pair $(\mathcal{G}, \circ)$, where $\mathcal{G}$ is a set and $\circ : \mathcal{G}\times\mathcal{G} \longrightarrow \mathcal{G}$ is a binary operation satisfying the following axioms:*

1. *Associativity: For all $a, b, c \in \mathcal{G}$, $(a\circ b)\circ c = a\circ(b\circ c)$.*
2. *Identity element: There exists an element $e \in \mathcal{G}$ such that for all $a \in \mathcal{G}$,$e\circ a = a\circ e = a$.*
3. *Inverse element: For each $a \in \mathcal{G}$, there exists an element $a^{-1} \in \mathcal{G}$ such that $a\circ a^{-1} = a^{-1}\circ a = e$.*

In GDL, rather than considering groups as abstract entities, we turn our attention to groups whose elements represent transformations in the data domain. For this, we need a mathematical formalization for the interaction between these transformations and the data. This formalization is given by group actions.

Definition 3.3 (Group Action)**.** *A (left) group action of a group $(\mathcal{G}, \circ)$ on a set X is a mapping $\cdot : \mathcal{G} \times X \longrightarrow X$, satisfying the following axioms:*

1. *Identity: the identity element $e \in \mathcal{G}$ acts as the identity transformation on X,*

$$e \cdot x = x, \quad \forall x \in X.$$

2. *Compatibility: for all $g, h \in \mathcal{G}$ and $x \in X$, the action satisfies:*

$$(g\circ h) \cdot x = g \cdot (h \cdot x).$$

Another useful concept is the one of orbits.

Definition 3.4 (Orbit)**.** *Let $(\mathcal{G}, \circ)$ be a group that acts on the set X with left group action $\cdot$. The orbit of an element $x \in X$ is defined as*

$$Orb\,(x) = \{g \cdot x \,:\, g \in \mathcal{G}\}$$

A natural next step is to verify how these transformations on Ω affect the signal that lives on it. Suppose we have a group $(\mathcal{G}, \circ)$ acting on Ω. We can lift its action to the set

of signals $\mathcal{S} = \{x : \Omega \longrightarrow C\}$ by defining

$$(g \cdot x)(u) = x(g^{-1} \cdot u),$$

for all $g \in G$, $u \in \Omega$ and $x \in \mathcal{S}$.

A consequence of this definition is that the group $(\mathcal{G}, \circ)$ acts *linearly* on $\mathcal{S}$ (the reader may easily varify that by applying the definitions). This is a very desirable property, as linear group actions on vector spaces can be represented as matrices, where the group action is simply matrix multiplication. This concept will be further explored in the next chapters.

To better visualize these concepts, consider the following example, where the domain can be thought of as cyclic graph of 4 nodes, and the group acting on it are rotations.

Example 3.1. *Let $\Omega = \{0, 1, 2, 3\}$. Consider the group $\mathcal{G} = \{0, 1, 2, 3\}$, where the group operation is addition modulo 4.*

$$g \circ h = g + h \bmod 4, \quad \text{for } g, h \in \mathcal{G}.$$

And the inverse is given by

$$g^{-1} = -g \bmod 4.$$

Now define a signal $x : \Omega \longrightarrow \mathbb{R}$, for instance

$$x(0) = a, \quad x(1) = b, \quad x(2) = c, \quad x(3) = d.$$

The action of $\mathcal{G}$ on the domain Ω can be defined also as addition modulo 4:

$$g \circ u = g + u \bmod 4 \quad \text{for } g \in \mathcal{G}, u \in \Omega,$$

and can be lifted to an action on signals:

$$(g \cdot x)(u) = x(g^{-1} \cdot u) = x(u - g \bmod 4), \qquad g \in \mathcal{G},\ u \in \Omega.$$

For example, when $g = 1$

$$(g \cdot x)(u) = x(u - 1 \bmod 4),$$

so that

$$\begin{aligned}
(g \cdot x)(0) &= x(0 - 1 \bmod 4) = x(3) = d \\
(g \cdot x)(1) &= x(1 - 1 \bmod 4) = x(0) = a \\
(g \cdot x)(2) &= x(2 - 1 \bmod 4) = x(1) = b \\
(g \cdot x)(3) &= x(3 - 1 \bmod 4) = x(2) = c.
\end{aligned}$$

Figure 3.2 illustrates how the group action affects the signal. Although the transformation produces a new signal, the cyclic order of the letters a, b, c, d remains unchanged. This demonstrates the idea of a symmetry: the underlying structure is preserved even though the signal itself is transformed.

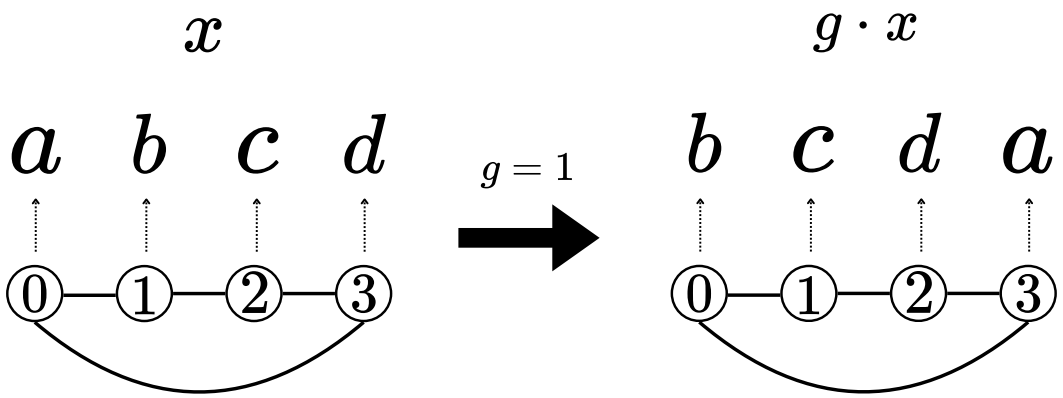

Figure 3.2: *Illustration of a rotation acting on a signal defined over a cyclic graph of 4 nodes.*

Now that we have the mathematical tools to describe the symmetries of the data, we are ready to define two of the principal concepts of Geometric Deep Learning: *invariance* and *equivariance*.

Definition 3.5 (Invariance)**.** *Let $(\mathcal{G}, \circ)$ be a group that acts on the set X with left group action $\cdot$ and Y a set. A function $f : X \to Y$ is said to be $\mathcal{G}$-invariant if*

$$f(g \cdot x) = f(x) \quad \forall g \in \mathcal{G}, x \in X.$$

Definition 3.6 (Equivariance)**.** *Let $(\mathcal{G}, \circ)$ be a group acting on the sets X and Y. A function $f : X \to Y$ is said to be $\mathcal{G}$-equivariant if*

$$f(g \cdot x) = g \cdot f(x) \quad \forall g \in \mathcal{G},\ x \in X,$$

where $\cdot$ denotes the corresponding group action on each set.

Thus, while an invariant function maps all elements of an orbit to the same value, an equivariant function ensures that inputs are transformed in a predictable and structured way under the action of a group.

These two notions form the main tools for incorporating symmetry priors into neural networks. When groups act on the domain Ω, and this action is extended to signals defined on Ω, the resulting transformations often correspond to changes that leave the underlying phenomenon unchanged. By constraining a network to be equivariant or invariant with respect to such transformations, we embed prior knowledge about the domain directly into the model, reducing the burden of having to learn these symmetries purely from data.

In chapters 4 and 5, we will see how the permutation group interacts with sets and graphs. After that, in chapter 6, we will study how the translation group interacts with grids.

3.3 Scale Separation

Although restricting the hypothesis space to respect the symmetries present in the data can help mitigate the curse of dimensionality, it may not be sufficient on its own. The scale separation prior formalizes the idea that phenomenon operating at different scales can often be modeled independently. Moreover, certain tasks may rely primarily on information at a specific scale, making it crucial to consider scale explicitly.

For example, in image classification, textures coexist with shapes and objects, often playing distinct roles depending on the task (see Figure 3.3). Similarly, in graphs, local connectivity patterns (e.g., whether a node is part of a triangle) coexist with global structures (e.g., the connected component to which the node belongs).

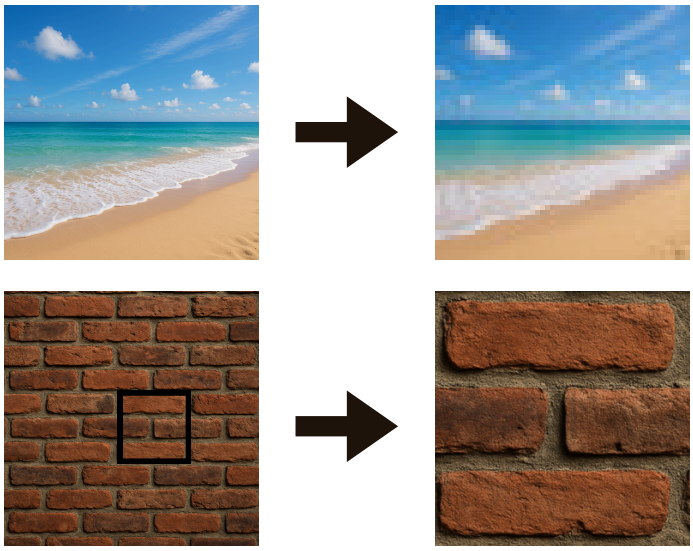

Figure 3.3: *Illustration of the role of scale in visual recognition. In the first example, the scene remains identifiable even after the resolution is significantly reduced—recognition is dominated by global structure. In contrast, in the second example, a small cropped region is sufficient to recognize the brick texture, indicating that recognition is driven by local structure.*

This observation can be leveraged to reduce the effective dimensionality of our problem, by *coarsening* the domain on which the signal is defined. Formally, this can be defined as following:

Definition 3.7 (Coarsening Operator)**.** *Let* $S = \{x : \Omega \to C\}$ *and* $S' = \{x' : \Omega' \to C\}$ *be sets of signals, with* $|\Omega'| \leq |\Omega|$*. A Coarsening Operator* P *is a function* $P : S \to S'$*.*

The intuition in this definition is that a signal x is mapped to a coarser signal x', which restricts the signal to a specific scale. This process is illustrated in figure 3.4.

Of course, knowing in advance how to perform coarsening is often not feasible, as it is highly task-dependent. For instance, in Figure 3.3, cropping the first image could remove essential information required to recognize the scene. For this reason, coarsening is

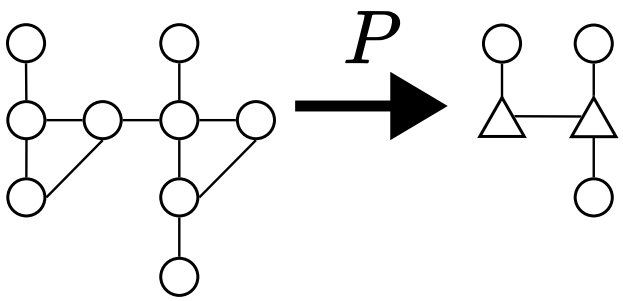

Figure 3.4: *Illustration of a possible graph coarsening process. In this example, each triangle is collapsed into a single node, effectively reducing the number of nodes in the graph and therefore decreasing the dimensionality of the domain.*

typically handled within the neural network architecture itself. Common examples include pooling operations in Convolutional Neural Networks, a topic we will revisit later.

3.4 GDL Blueprint

Now we are ready to combine both geometric priors into the Geometric Deep Learning Blueprint.

Definition 3.8 (Equivariant Block)**.** *Let $\mathcal{S} = \{x : \Omega \to C\}$, $\mathcal{S}' = \{x' : \Omega' \to C'\}$ and $\mathcal{S}'_c = \{x'_c : \Omega'_c \to C'\}$ be sets of signals, with $|\Omega'_c| \leq |\Omega'|$. Let $(\mathcal{G}, \circ)$ be a group acting on $\Omega, \Omega', \Omega'_c$ through $\cdot$. We define the following building blocks:*

1. *Linear $\mathcal{G}$-equivariant layer $E : \mathcal{S} \to \mathcal{S}'$, satisfying $E(g \cdot x) = g \cdot E(x)$ for all $g \in G$ and $x \in \mathcal{S}$.*
2. *Nonlinearity $\theta : \mathcal{S}' \to \mathcal{S}'$, obtained applying a non-linear function $\vartheta : C' \to C'$ coordinate-wise: $\theta(x')(u') = \vartheta(x'(u'))$, for $x' \in \mathcal{S}'$ and $u' \in \Omega'$.*
3. *Coarsening Operator $P : \mathcal{S}' \to \mathcal{S}'_c$.*

A $\mathcal{G}$-Equivariant Block $B : \mathcal{S} \to \mathcal{S}'_c$ is a composition

$$B(x) = P(\theta(E(x))), \quad x \in \mathcal{S}.$$

This definition specifies a function that not only preserves equivariance but can also incorporate coarsening of the domain. By defining the right group, operating on specific

domains and stacking multiple equivariant blocks, one can derive a wide range of architectures as special cases of the general blueprint. In practice, however, the choice of nonlinearities and coarsening operators must be made carefully, since not all of them preserve equivariance.

A natural question arises: why impose linearity on the equivariant block E? Linearity provides a powerful characterization tool. For instance, all linear equivariant maps with respect to translations turn out to be convolutions, a fact that we will revisit in the next chapters. Moreover, this constraint aligns with many established architectures, such as convolutions in CNNs and message passing in GNNs. On the other hand, a composition of linear functions remains linear, which would severely limit expressivity. To overcome this, the blueprint allows the interleaving of coordinate-wise nonlinearities (e.g., ReLU or MLPs applied independently at each site), which expand representational capacity.

It is worth noting that not all architectures adhere strictly to this formulation. Some omit the coarsening operator altogether, while others relax the requirement that E be linear, thereby sidestepping the need for explicit nonlinearities. Concrete examples of these variations will be discussed in subsequent chapters.

While stacking equivariant blocks suffices for tasks where the output itself must transform equivariantly with the input (e.g., semantic segmentation or node classification, where labels are structured as signals), invariance (as required, for example, in image classification and graph classification) is not obtained directly from the equivariant blocks. For this, we need an invariant layer, also referred to as global pooling.

Definition 3.9 (Invariant Layer)**.** *Let $\mathcal{S} = \{x : \Omega \to C\}$ be a set of signals, and $\mathcal{Y}$ the set of labels. Let $(\mathcal{G}, \circ)$ be a group that acts on Ω. A $\mathcal{G}$-Invariant Layer is a function $I : \mathcal{S} \to \mathcal{Y}$ that satisfies*

$$I(g \cdot x) = I(x) \quad \text{for all } x \in S, g \in \mathcal{G}.$$

This layer maps the transformed signal to the final prediction. By appending an invariant layer to the end of a sequence of equivariant blocks, we obtain an invariant neural network.

Many well-known architectures can be understood through this framework by selecting an appropriate symmetry group and geometric domain. Table 3.1 highlights representative examples.

In the following chapters, we will explore a few of these architectures. We will examine sets and graphs under permutation groups, leading to DeepSets and GNNs; show how Transformers can be interpreted as a special case of GNNs; and finally, derive CNNs from grids equipped with the translation group.

Architecture	**Domain Ω**	**Symmetry Group $\mathcal{G}$**
CNN	2D Grid	Translation
Spherical CNN	Sphere	Rotation
Intrinsic / Mesh CNN	Manifold	Isometry / Gauge symmetry
GNN	Graph	Permutation
DeepSets	Set	Permutation
Transformer	Complete Graph	Permutation
LSTM	1D Grid	Time warping

Table 3.1: *Examples of neural architectures, their input domains, and corresponding symmetry groups.*

Chapter 4

Learning on Sets

The content of this chapter closely follows Michael M Bronstein *et al.* (2021) and Bronstein, Michael M. (2022a). Sets can be understood as unordered collections of distinct objects, where repetition plays no role. We begin our exposition of the GDL blueprint with sets, as they represent the simplest mathematical structure with minimal geometric assumptions. Moreover, since a set can be viewed as a graph without edges, many of the ideas developed here naturally extend to graphs. This chapter proceeds by first defining signals on sets, then examining the group of transformations acting on the domain, and finally deriving the *DeepSets* architecture.

4.1 Setup

We start by defining the domain Ω of the signals, which are simply finite sets without any additional structure. Assume that this set is represented by natural numbers, therefore $\Omega = [d]$. Consider that the codomain C of the signal x is $\mathbb{R}^k$.

One natural way of representing a signal $x : [d] \to \mathbb{R}^k$ is to build a *feature matrix* $X \in \mathbb{R}^{d \times k}$, where the ith row of X denotes $x(i)$,

$$X = \begin{bmatrix} \text{———} & x(1) & \text{———} \\ \text{———} & x(2) & \text{———} \\ & \vdots & \\ \text{———} & x(d) & \text{———} \end{bmatrix}.$$

However, this representation creates an implicit node ordering. If we want to use X as input to a neural network f, we would require it to have the same output for all *permutations* of the rows of X, encompassing the idea that orders do not matter for sets. This idea can be formalized by the permutation group.

4.2 Permutation Symmetry

We start by defining what is a permutation.

Definition 4.1 (Permutation)**.** *Let Ω be a finite set. A function $\varphi : \Omega \longrightarrow \Omega$ is said to be a permutation if it is a bijection.*

One may verify that the set of all permutations of a finite set Ω under function composition form a group $(\mathcal{G}, \circ)$, where the identity element is the identity function and inverse elements are inverse functions. This group acts on Ω by simple function application $g \cdot u = g(u)$, $g \in \mathcal{G}$ and $u \in \Omega$.

More interstingly is how this group acts on the signal x. Recall that a group acting on Ω can be lifted to act (linearly) on $\mathcal{S}$ through

$$(g \cdot x)(u) = x(g^{-1} \cdot u).$$

To understand how this manifests in our setup, lets work through an example.

Example 4.1. *Let $\Omega = \{1, 2, 3\}$ and $x : \Omega \longrightarrow \mathbb{R}$, $x(u) = u + 3$.*

Our feature matrix can be written as

$$X = \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix}.$$

Let $g : \Omega \longrightarrow \Omega$ be a permutation such that $g(1) = 1$, $g(2) = 3$ and $g(3) = 2$. Therefore, its inverse is $g^{-1}(1) = 1$, $g^{-1}(2) = 3$ and $g^{-1}(3) = 2$. Analyzing how this transformation affects the signal yields:

$$\begin{aligned}(g \cdot x)(1) &= x(g^{-1} \cdot 1) = x(g^{-1}(1)) = x(1) = 4,\\ (g \cdot x)(2) &= x(g^{-1} \cdot 2) = x(g^{-1}(2)) = x(3) = 6,\\ (g \cdot x)(3) &= x(g^{-1} \cdot 3) = x(g^{-1}(3)) = x(2) = 5.\end{aligned}$$

The transformed feature matrix can be written as

$$X = \begin{bmatrix} 4 \\ 6 \\ 5 \end{bmatrix}.$$

Therefore, although permutations do not affect the underlying domain, they permute the signal according to the permutation rule. In the example, each coordinate is mapped to the next, and the last is mapped back to the first. At a first glance it may seem as we are overcomplicating the notation of permutations, but we highlight that the power of the GDL framework is on describing a wide variety of transformations on different data. More complicated transformations will be analyzed in the next chapter.

A more linear algebra friendly way of representing permutations is through *permutation matrices*.

Definition 4.2. *Let $I \in \mathbb{R}^{d\times d}$ be the identity matrix. Let $\varphi : [d] \longrightarrow [d]$ be a permutation. The permutation matrix of φ, denoted by P, is defined as*

$$P_i = I_{\varphi^{-1}(i)}.$$

In other words, if one considers a signal that maps the integer i to the canonical vector e_i (i. e. $e_i \in \mathbb{R}^d$ such that the i-th entry is equal to one and the others are equal to 0), and the identity matrix to be its feature matrix, obtaining the permutation matrix is the same as allowing the permutaion to act on such signal and retrieving the transformed feature matrix. This representation is useful because permuting the rows of an abribrary feature matrix X can be done by simply multiplying the matrix in the left by the appropriate permutation matrix (i. e. $(PX)_i = X_{\varphi^{-1}(i)}$).

We are now ready to express the permutation invariance and equivariance for neural networks operating on feature matrices.

Definition 4.3 (Permutation Invariance)**.** *A function $f : \mathbb{R}^{d\times k} \longrightarrow \mathcal{Y}$, where $\mathcal{Y}$ is any set, is said to be permutation invariant if, for all permutation matrices $P \in \mathbb{R}^{d\times d}$ and all $X \in \mathbb{R}^{d\times k}$,*

$$f(PX) = f(X).$$

Equivalently,

Definition 4.4 (Permutation Equivariance)**.** *A function $f : \mathbb{R}^{d\times k} \longrightarrow \mathbb{R}^{d\times k'}$ is said to be permutation equivariant if, for all permutation matrices $P \in \mathbb{R}^{d\times d}$ and all $X \in \mathbb{R}^{d\times k}$,*

$$f(PX) = Pf(X).$$

These are the two concepts used to restrict the architectures with the geometric constraints for operating on sets. Next, we will see examples of how this properties can be accomplished.

4.3 DeepSets

We will now show how these constraints can be enforced in a neural network architecture, exemplifying how the *DeepSets* (Zaheer *et al.*, 2017) fit the framework.

We will start by designing a permutation equivariant block. We will perform two small changes to the general blueprint (Definition 3.8). First, we won't be applying any coarsening operator. Second, we will not require the equivariant transformation to be linear.

The key insight for achieving permutation equivariance is through weight sharing. The DeepSets architecture accomplish that by applying the same function on each row of the feature matrix. We will go in details in this architecture, showing how to apply this concepts with multi-layer perceptrons (MLPs). To start, we must first define the MLP. This will be accomplished using affine layers.

Definition 4.5 (Affine Layer). *Let $A \in \mathbb{R}^{k' \times k}$ and $b \in \mathbb{R}^{k'}$. An affine layer function $f_{A,b} : \mathbb{R}^k \longrightarrow \mathbb{R}^{k'}$ is a function*

$$f_{A,b}(x) = Ax + b.$$

We say that A and b are the learnable parameters.

With this, we can define a Multi-layer Perceptron.

Definition 4.6 (Multi-layer Perceptron). *Let $k_0, ..., k_l \in \mathbb{N}$ and let $\mathcal{W} = \{(A_i, b_i)\}_{i=1}^{l}$ be a familly of ordered pairs of matrices such that $A_i \in \mathbb{R}^{k_i \times k_{i-1}}$ and $b_i \in \mathbb{R}^{k_i}$, with corresponding affine layers f_{A_i,b_i}. Let $\theta_i : \mathbb{R}^{k_i} \longrightarrow \mathbb{R}^{k_i}$, for $i \in [l-1]$ be a point-wise non-linearity (such as ReLU or sigmoid). A Multi-layer Perceptron is a function $MLP_{\mathcal{W}} : \mathbb{R}^{k_0} \longrightarrow \mathbb{R}^{k_l}$ obtained by composing Affine Layers and non-linearities:*

$$MLP_{\mathcal{W}} = f_{A_l,b_l} \circ \theta_{l-1} \circ f_{A_{l-1},b_{l-1}} \circ \ldots \circ \theta_2 \circ f_{A_2,b_2} \circ \theta_1 \circ f_{A_1,b_1}.$$

We say that $\mathcal{W}$ are the learnable parameters.

For classification tasks, usually the softmax function is appended after the last affine layer.

Regarding expressivity, the Universal Approximation Theorem states that a MLP with sufficient parameters and a suitable non-linear activation can approximate any continuous function (under mild assumptions) arbitrarily well. In other words, given enough hidden units, an MLP is expressive enough to come arbitrarily close to any target continuous mapping (Cybenko, 1989; Hornik, 1991).

The DeepSets accomplish a permutation equivariant representation by applying the same MLP on every row of the feature matrix. To match the language of the GDL blueprint, one can consider the equivariant block to be one affine layer (applied row-wise) and a non-linearity applied in every row of the input matrix. Each one of these equivariant blocks would match one pair of affine layer and nonlinearity from the MLP, where the last equivariant block does not have the nonlinearities. However, to simplify notation, we will consider simply applying the MLP on every row.

Let $\text{MLP}_{\mathcal{W}} : \mathbb{R}^k \longrightarrow \mathbb{R}^{k'}$ be a multi-layer perceptron and $X \in \mathbb{R}^{d \times k}$ be a feature matrix. A permutation equivariant representation (according to Definition 4.4) H can be obtained by computing $H_i = \text{MLP}_{\mathcal{W}}(X_i)$ for all $i \in [d]$, that is,

$$H = \begin{bmatrix} \text{\_\_\_\_} & \text{MLP}_{\mathcal{W}}(X_1) & \text{\_\_\_\_} \\ \text{\_\_\_\_} & \text{MLP}_{\mathcal{W}}(X_2) & \text{\_\_\_\_} \\ & \vdots & \\ \text{\_\_\_\_} & \text{MLP}_{\mathcal{W}}(X_d) & \text{\_\_\_\_}. \end{bmatrix}$$

The reader may easily verify that this indeed is permutation equivariant. All that is left to complete the DeepSets architecture is to build an invariant layer. This invariant layer aggregates the representations H_i and maps them to the final prediction. This aggregation is performed trough a permutation-invariant operator (e. g. sum, mean, standard deviation, or max), and the final prediction is obtained by feeding the aggregated representation

into another MLP. This can be formalized as following:

Definition 4.7 (DeepSets)**.** *Let $MLP_{\mathcal{W}} : \mathbb{R}^k \to \mathbb{R}^{k'}$ and $MLP_{\mathcal{W}'} : \mathbb{R}^{k'} \to \mathcal{Y}$ be MLPs. Let $\oplus$ denote any permutation-invariant operator. A DeepSet is a function $DS_{\mathcal{W},\mathcal{W}'} : \mathbb{R}^{d\times k} \to \mathcal{Y}$, defined by*

$$DS_{\mathcal{W},\mathcal{W}'}(X) = MLP_{\mathcal{W}'}\left(\bigoplus_{i\in[d]} MLP_{\mathcal{W}}(X_i)\right)$$

It is straightforward to verify that the construction above yields a permutation-invariant function, so we omit the proof. Instead, we now turn to a more interesting and fundamental aspect of the DeepSets framework: its expressive power. In particular, we will show that any function defined on sets can be represented using a similar architectural form.

Theorem 4.1. *Let χ be a discrete set and let 2^χ denote its power set (i. e. the set of all subsets of χ). Let $f : 2^\chi \to \mathcal{Y}$ be any function. Then there exist functions ϕ and ρ such that, for all $S \in 2^\chi$,*

$$f(S) = \rho\left(\sum_{s\in S} \phi(s)\right).$$

Proof. We prove the statement by explicit construction. Since χ is discrete, fix an arbitrary enumeration $\psi : \chi \to \mathbb{N}$. Define

$$\phi(s) = 2^{\psi(s)} \qquad \text{for all } s \in \chi.$$

Consider any subset $S \in 2^\chi$. The value $\sum_{s\in S} \phi(s)$ corresponds to the binary integer whose k-th bit is 1 if and only if $\psi^{-1}(k) \in S$. Therefore, this sum uniquely encodes the subset S. More formally, if $S, L \in 2^\chi$ and $S \neq L$, then

$$\sum_{s\in S} \phi(s) \neq \sum_{l\in L} \phi(l).$$

Thus, the mapping

$$S \longmapsto \sum_{s\in S} \phi(s)$$

is injective.

We now define ρ on the image of this map by assigning

$$\rho\left(\sum_{s\in S} \phi(s)\right) = f(S), \qquad \text{for all } S \in 2^\chi.$$

Since the encoding is injective, this definition is consistent and ρ is well defined. Therefore, for every $S \in 2^\chi$,

$$f(S) = \rho\left(\sum_{s\in S} \phi(s)\right),$$

completing the proof. □

Notably, Zaheer *et al.*, 2017 extend the result to the case where χ is not necessarily discrete. Since the general proof introduces additional technical considerations, we do not include it here.

Chapter 5

Learning on Graphs

The content of this chapter closely follows Michael M Bronstein *et al.* (2021), Bronstein, Michael M. (2022a) and Bronstein, Michael M. (2022b). Graphs can be understood as sets augmented with connections between the elements. Graphs are everywhere, and are used to model a vast collections of phenomenon, from molecules, to social networks and maps.

In this chapter, first, we will define signals on graphs, then we will examine how the permutation group acts on these objects. Next, we will apply the GDL blueprint to derive networks espacialized on graph data: *Graph Neural Networks* (GNNs). Finally, we will see how to fit the popular *Transformer* architecture as as special case of the GNN.

5.1 Setup

There are multiple ways of defining a graph. Here, we consider that a graph G is an ordered pair (V, E) of two finite sets. The set V is said to be the set of *vertices* (or *nodes*) of the graph G. The set E contains sets of unordered pairs of elemnts of V. We say that two vertices $u, v \in V$ are adjacent if $\{u, v\} \in E$. We call $\{u, v\}$ (usually denotes simply by uv or vu, since order does not matter) an edge of G and, similarly, we call E the set of edges of G.

Edges can be represented as adjacency matrices.

Definition 5.1 (Adjacency matrix)**.** *Let $G = (V, E)$ be a graph. The adjacency matrix $A \in \{0, 1\}^{V \times V}$ of G is defined as*

$$A_{ij} = \begin{cases} 1, & \text{if } ij \in E, \\ 0, & \text{otherwise.} \end{cases}$$

It is immediate of this definitions that A is symmetric. It is important to highlight that *digraphs* (i. e. directed graphs) can be obtained by changing the definition of E to contain ordered pairs instead of sets. In this case, there will be a distinction between edges uv and vu, in a way that the adjacency matrix will not be symmetric in all cases.

In any case, let us proceed with the given definition of graphs. To better understand it, consider the following example.

Example 5.1. *Consider the graph whose vertices form a cycle, with one additional edge connecting two non-adjacent vertices (a* chord*). Its vertex and edge sets are:*

$$V = \{v_1, v_2, v_3, v_4, v_5\},$$

$$E = \{\{v_1, v_2\}, \{v_2, v_3\}, \{v_3, v_4\}, \{v_4, v_5\}, \{v_5, v_1\}, \{v_1, v_3\}\}.$$

A possilbe graphical representation is given in Figure 5.1.

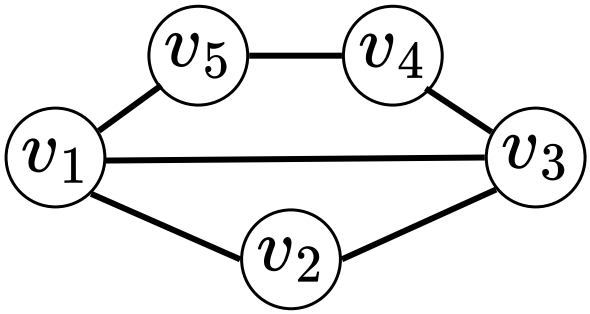

Figure 5.1: *An example of a graph.*

Although we can also have features for the edges and for the whole graph, let us consider now graphs with features only in the vertices. That is, the signal domain is $\Omega = V$. For simplicity, let us assume that V is represented by the natural numbers, i. e. $\Omega = V = [d]$. Therefore, the feature matrix $X \in \mathbb{R}^{d \times k}$ of the signal $x : [d] \longrightarrow \mathbb{R}^k$ can be defined exactly as in the case of sets, i. e. $X_i = x(i)$, for all $i \in [d]$.

Therefore, for a graph G, combining the adjacency matrix A with the feature matrix X provides a representation of both the signal and the structure of the graph, serving as input to a neural network. However, just like in the previous case with sets, this representation assumes an implicit ordering of the nodes. Therefore, we must once again rely on invariance and equivariance to the permutation group to enforce the prior that the ordering of the nodes do not matter. The key difference here is that we must also permute the adjacency matrix accordingly, to preserve the structure of the graph and only relabel its nodes. When applying a permutation matrix P, this amounts to computing $PAP^\top$.

Therefore, permutation invariance and equivariance for graphs can be defined as following:

Definition 5.2 (Permutation Invariance)**.** *A function $f : \mathbb{R}^{d\times k} \times \mathbb{R}^{d\times d} \to \mathcal{Y}$, where $\mathcal{Y}$ is any set, is said to be permutation invariant if, for all permutation matrices $P \in \mathbb{R}^{d\times d}$, all $X \in \mathbb{R}^{d\times k}$, and all $A \in \mathbb{R}^{d\times d}$,*

$$f(PX, PAP^\top) = f(X, A).$$

Equivalently,

Definition 5.3 (Permutation Equivariance)**.** *A function $f : \mathbb{R}^{d\times k} \times \mathbb{R}^{d\times d} \to \mathbb{R}^{d\times k'}$ is said to be permutation equivariant if, for all permutation matrices $P \in \mathbb{R}^{d\times d}$, all $X \in \mathbb{R}^{d\times k}$, and all $A \in \mathbb{R}^{d\times d}$,*

$$f(PX, PAP^\top) = Pf(X).$$

In this context, a useful tool for designing equivariant and invariant neural networks is the idea of neighbourhoods of vertices:

Definition 5.4 (Neighbourhood)**.** *Let $G = (V, E)$ be a graph with adjacency matrix $A \in \mathbb{R}^{d\times d}$. The neighbourhood of $v \in V$, denoted by $N_A(v)$, is defined as*

$$N_A(v) := \{u \in V : A_{uv} = 1\}.$$

Similarly, for $v \in V$, we can define $X_{N(v)} \in \mathbb{R}^{N(v)\times k}$ to denote the feature matrix restricted to the neighbourhood of v (i. e. a matrix that contains the features of the nodes in the neighbourhood of v).

Next, we will see how to explore this concept to design a permutation equivariant layer for graphs.

5.2 Graph Neural Networks

A key idea that can be explored to create permutation equivariant layers for graphs is the idea of *locality*. Each node updates its representation using the information from its neighbors. Formally, this yields the following definiton:

Definition 5.5 (GNN Layer)**.** *Let $G = (V, E)$ be a graph with corresponding adjacency matrix $A \in \mathbb{R}^{d\times d}$ and feature matrix $X \in \mathbb{R}^{d\times k}$. Define $\mathcal{M}_d = \bigcup_{i\in\{0\}\cup[d]} \mathbb{R}^{i\times d}$ to be the set of all matrices with up to d rows. Let $\sigma : \mathbb{R}^k \times \mathcal{M}_d \to \mathbb{R}^{k'}$ be a permutation invariant function w. r. t. the second argument (i. e. $\sigma(h, PH) = \sigma(h, H)$ for all permutation matrices P). A GNN layer $f : \mathbb{R}^{d\times k} \times \mathbb{R}^{d\times d} \to \mathbb{R}^{d\times k'}$ is a function defined by*

$$f(X, A) = \begin{bmatrix} \text{—} & \sigma(X_1, X_{N_A(1)}) & \text{—} \\ \text{—} & \sigma(X_2, X_{N_A(2)}) & \text{—} \\ & \vdots & \\ \text{—} & \sigma(X_d, X_{N_A(d)}) & \text{—} \end{bmatrix}.$$

The function σ is usually called a diffusion, propagation or message passing. Next, we show that GNN layers are indeed permutation invariant. Before, we will prove a result that will help with the proof.

Lemma 5.1. *Let $f : \mathbb{R}^{d\times k} \longrightarrow \mathbb{R}^{d\times k'}$ be a permutation invariant function and $C \subseteq [d]$. Let $P \in \mathbb{R}^{d\times d}$ be a permutation matrix associated with perutation $\varphi : [d] \longrightarrow [d]$. Then, for all $X \in \mathbb{R}^{d\times k}$,*

$$f((PX)_C) = f(X_{\varphi^{-1}(C)}),$$

where $\varphi^{-1}(C)$ denotes the image of φ^{-1} over C, i. e. $\varphi^{-1}(C) = \{\varphi^{-1}(c) : c \in C\}$.

Proof. Let $c_1, c_2, ..., c_l \in C \subseteq [d]$, such that $c_1 < c_2 < ... < c_l$. Note that,

$$(PX)_C = \begin{bmatrix} (PX)_{c_1} \\ (PX)_{c_2} \\ \vdots \\ (PX)_{c_l} \end{bmatrix} = \begin{bmatrix} X_{\varphi^{-1}(c_1)} \\ X_{\varphi^{-1}(c_2)} \\ \vdots \\ X_{\varphi^{-1}(c_l)} \end{bmatrix},$$

which is clearly a permutation of the rows of $X_{\varphi^{-1}(C)}$. Since f is permutation invariant, we have $f((PX)_C) = f(X_{\varphi^{-1}(C)})$. □

We are now ready to state and prove GNN permutation equivariance.

Theorem 5.1. *Let $f : \mathbb{R}^{d\times k} \times \mathbb{R}^{d\times d} \longrightarrow \mathbb{R}^{d\times k'}$ be a GNN layer with corresponding function $\sigma : \mathbb{R}^k \times \mathcal{M}_d \longrightarrow \mathbb{R}^{k'}$. Then,*

$$f(PX, PAP^\top) = Pf(X, A),$$

for all permutation matrices $P \in \mathbb{R}^{d\times d}$, all $X \in \mathbb{R}^{d\times k}$ and all $A \in \mathbb{R}^{d\times d}$.

Proof. Let $P \in \mathbb{R}^{d\times d}$ be a permutation matrix associated with permutation $\varphi : [d] \longrightarrow [d]$. Let $G = (V, E)$ be a graph with feature matrix $X \in \mathbb{R}^{d\times k}$ and adjacency matrix $A \in \mathbb{R}^{d\times d}$. Let $f : \mathbb{R}^{d\times k} \times \mathbb{R}^{d\times d} \longrightarrow \mathbb{R}^{d\times k'}$ be a GNN layer with associated function $\sigma : \mathbb{R}^k \times \mathcal{M}_d \longrightarrow \mathbb{R}^{k'}$. Then, for $i \in [d]$,

$$\begin{aligned} f(PX, PAP^\top)_i &= \sigma((PX)_i, (PX)_{N_{PAP^\top}(i)}) \\ &= \sigma(X_{\varphi^{-1}(i)}, (PX)_{N_{PAP^\top}(i)}) \\ &= \sigma(X_{\varphi^{-1}(i)}, X_{\varphi^{-1}(N_{PAP^\top}(i))}), \end{aligned}$$

where the last equality holds because of Lemma 5.1. Now, note that

$$\begin{aligned} \varphi^{-1}(N_{PAP^\top}(i)) &= \varphi^{-1}(\{j \in [d] : (PAP^\top)_{ij} = 1\}) \\ &= \varphi^{-1}(\{j \in [d] : A_{\varphi^{-1}(i)\varphi^{-1}(j)} = 1\}) \\ &= \varphi^{-1}(\{\varphi(j) \in [d] : A_{\varphi^{-1}(i)j} = 1\}) \\ &= \{j \in [d] : A_{\varphi^{-1}(i)j} = 1\} \\ &= N_A(\varphi^{-1}(i)). \end{aligned}$$

Therefore, we have

$$\begin{aligned} f(PX, PAP^\top)_i &= \sigma(X_{\varphi^{-1}(i)}, X_{\varphi^{-1}(N_{PAP^\top}(i))}) \\ &= \sigma(X_{\varphi^{-1}(i)}, X_{N_A(\varphi^{-1}(i))}) \\ &= f(X, A)_{\varphi^{-1}(i)} \\ &= (Pf(X, A))_i. \end{aligned}$$

Since $f(PX, PAP^\top)_i = (Pf(X, A))_i$ holds for all $i \in [d]$, we have

$$f(PX, PAP^\top) = Pf(X, A).$$

□

Now that we have a framework for GNN layers, it remains to specify how to implement σ. Notably, most of GNN layers fall under 3 categories: *convolutional*, *attentional*, and *message-passing*.

Let $\varrho : \mathbb{R}^k \longrightarrow \mathbb{R}^{k'}$ and $\vartheta : \mathbb{R}^k \times \mathbb{R}^{k'} \longrightarrow \mathbb{R}^{k''}$ be two learnable functions (e. g. MLPs). A convolutional GNN layer can be generically expressed as

$$\sigma(X_i, X_{N_A(i)}) = \vartheta\Big(X_i, \bigoplus_{j \in N_A(i)} c_{ij}\varrho(X_j)\Big),$$

where $\oplus$ denotes any permutation invariant aggregator (e. g. sum), and c_{ij} are constant scalars (i. e. not learnable). Likewise, an attentional GNN can be expressed as

$$\sigma(X_i, X_{N_A(i)}) = \vartheta\Big(X_i, \bigoplus_{j \in N_A(i)} a(X_i, X_j)\varrho(X_j)\Big),$$

where $a : \mathbb{R}^k \times \mathbb{R}^k \longrightarrow \mathbb{R}$ is a learnable function, called the *self-attention mechanism*, which is usually normalized using softmax across all neighbors.

Finally, the most general form of GNN is the message-passing GNN, which defines σ by

$$\sigma(X_i, X_{N_A(i)}) = \vartheta\Big(X_i, \bigoplus_{j \in N_A(i)} m(X_i, X_j)\Big),$$

where $m : \mathbb{R}^k \times \mathbb{R}^k \longrightarrow \mathbb{R}^{k'}$ is a learnable *message* function.

One important thing to notice is that these approaches are sorted in order of increasing generality. Attentional GNNs may represent convolutional GNNs where the attention mechanism is simply a look-up table (i. e. $a(X_i, X_j) = c_{ij}$) and message-passing GNNs can implement attentional GNNs by propagating only the senders node features (i. e. $m(X_i, X_j) = a(X_i, X_j)\varrho(X_j)$).

GNN layers are usually interleaved with non-linearities, but commonly not with a coarsening operator. Once the representations have been updated for multiple layers, they can be used to perform vertex level tasks, pairwise (invariantly) aggregated for edge

level tasks or invariantly aggregated for the whole graph for graph level tasks, following the GDL blueprint.

Regarding expressivity, it is easy to note that not all permutation equivariant operations can be represented as message passing (e. g. global averaging). In fact, it is known that message-passing GNNs are at most as expressive as the Weisfeiler–Lehman test (Xu *et al.*, 2018). One type of message-passing GNN that matches WL-test expressivity is the GIN (Xu *et al.*, 2018).

Naturally, the ideas presented can be extended to edge and graph-level features. However, we will keep the discussion up to node level featuers, as it suffices to exemplify the GDL framework.

5.3 Transformers as GNNs

The NLP field has been profoundly transformed by the introduction of the Transformer architecture (Vaswani *et al.*, 2017). Contrary to the common view that Transformers are inherently sequence-based models, one can derive their core computational primitive—the self-attention mechanism—from the perspective of attentional Graph Neural Networks (GNNs). In this section, we present the fundamental components of the Transformer architecture and outline how they naturally arise from, and relate to, GNN formulations. Our exposition follows the interpretations developed in Joshi, 2020; Michael M Bronstein *et al.*, 2021.

We start by explaining the transformer. We will use standard NLP notation and later link to GDL notation.

Consider a sentence $(s_1, \dots, s_k)$, where each s_i denotes a token (i.e., a subword unit). Each token s_i is associated with an initial embedding $h_i^{(0)} \in \mathbb{R}^d$, which encodes both its semantic content and its position within the sequence (via positional encodings).

A Transformer layer updates the representation of every token by aggregating information from all other tokens in the sentence. Which tokens should influence the update of a given representation is not predetermined; instead, it is learned during training through the self-attention mechanism. Formally,

$$h_i^{l+1} = \sum_{j \in [k]} w_{ij}(V h_j^l), \text{ where } w_{ij} = \frac{e^{(Qh_i)^\top K h_j}}{\sum_{p \in [k]} e^{(Qh_i)^\top K h_p}}$$

where Q, K, and V are learnable projection matrices. In practice, Transformer layers include additional components. For instance, self-attention is computed multiple times using distinct sets of (Q, K, V) matrices; the resulting attention heads are then concatenated and projected back to the model dimension. Subsequently, a LayerNorm operation and a position-wise MLP are applied before the updated token representation is passed to the next layer. For simplicity, we restrict our discussion to a single self-attention head.

Next, we will show that one can describe the self-attention mechanism as a GNN layer. For this, consider the set of all tokens in the sentence as a complete graph. That is, the

vertices are $S = s_1, ..., s_k$ and every vertex is connected to every vertex. Define the signal that maps tokens to embeddings $x(s_i) = h_i^0, i \in [k]$ and consider applying the attentional GNN layer to update the representation of the i-th token of the sentence. That is,

$$h_i^1 = \vartheta\left(h_i^0, \bigoplus_{j\in[k]} a(h_i^0, h_j^0)\varrho(h_j^0)\right)$$

Note that the invariant operator ranges from 1 to k because we are considering a complete graph. Define the functions $\vartheta(a, b) = b$, the aggregation operator $\bigoplus = \sum$, the attention weight function $a(h_i^{(0)}, h_j^{(0)}) = w_{ij}$, and the value transformation $\varrho(h_j^{(0)}) = Vh_j^{(0)}$. Under these choices, the resulting update rule coincides with the self-attention mechanism.

At first glance, this appears to contradict the common view of Transformers as purely sequence-based models. Interpreting Transformers as GNNs makes explicit the need for positional encodings: without them, all tokens would be indistinguishable under permutation of the input.

Chapter 6

Learning on Grids

The content of this chapter closely follows Michael M Bronstein *et al.* (2021) and Bronstein, Michael M. (2022c). Grids can be viewed as a particular class of graphs whose adjacency structure follows a regular, grid-like pattern. Typical examples include images and videos, where each pixel interacts with its spatial neighbors.

This chapter begins by formalizing grids and the signals defined on them. We then examine the translation group acting on these domains. Finally, we show how convolution induces translation equivariance and how convolutional neural networks (CNNs) leverage this property to achieve remarkable performance.

6.1 Setup

We start by defining the domain Ω of the signals, i. e., the grids. In this section, we will work with 2D grids, since they are used to represent images. Therefore, a $d \times d$ grid can be defined as a set of ordered pairs, ranging from 0 to $d-1$. That is, $\Omega = \{0, ..., d-1\} \times \{0, ..., d-1\}$. Regarding the structure of the grid, we consider that node $(i, j) \in \Omega$ is adjacent to 4 other nodes: $(i+1 \bmod d, j), (i, j+1 \bmod d), (i-1 \bmod d, j), (i, j-1 \bmod d)$. Note that this represents grids that are adjacent across borders. We do this for mathematical simplicity.

As for the signal, let us consider that the image C of the signal x are the real numbers $\mathbb{R}$ (i. e. $C = \mathbb{R}$). This is sufficient to encode grayscale images and simplifies mathematics. Notably, one could consider a vector space of more dimensions (yielding, for example, images of more channels). However, this adds the complexity of manipulating tensors (i. e. higher dimensionar matrices) instead of matrices.

A natural way of representing this is through a matrix $X \in \mathbb{R}^{\Omega}$ s. t. $X_{ij} = x(i, j)$, where each entry of the matrix corresponds to the value of the signal in the corresponding node $X_{ij} = x(i, j)$.

6.2 Translation Symmetry

To derive convolutional neural networks, we are interested in the translation group. A translation can be understood as a special case of permutation.

Definition 6.1 (Translation)**.** *Let* $\Omega = \{0, ..., d-1\} \times \{0, ..., d-1\}$*. Let* $m, n \in \{0, ..., d-1\}$*. A function* $\varphi_{m,n} : \Omega \longrightarrow \Omega$ *is said to be a translation if it has the form*

$$\varphi_{m,n}(i, j) = (i + m \bmod d, j + n \bmod d).$$

Similarly to permutations, one may verify that the set of all translations of Ω under function composition form a group. The identity element is the identity function, and inverse elements are inverse functions (i. e. $\phi_{m,n}^{-1} = \phi_{-m,-n}$). This group acts on Ω by simple function application.

Recall that a group acting on Ω can be lifted to act (linearly) on $\mathcal{S}$. In this case, this amounts to

$$(\phi_{m,n} \cdot x)(i, j) = x(\phi_{m,n}^{-1} \cdot (i, j)) = x(i - m \bmod d, j - n \bmod d).$$

A more linear algebra friendly way of representing translations is through shift operators. They can be obtained by using two permutation matrices. Consider a matrix X that represents the signal x. Suppose that we want to obtain the matrix X' that represents the signal $\varphi_{m,n} \cdot x$.

Define the permutation $\varpi_m : \{0, ..., d-1\} \longrightarrow \{0, ..., d-1\}$ that maps i to $i + m \bmod d$. Likewise, define ϖ_n and let S^m and S^n be their respective permutation matrices. X' can be obtained by computing

$$X' = S^m X (S^n)^\top$$

We are now ready to define invariance and equivariance for the translation group.

Definition 6.2 (Translation Invariance)**.** *A function* $f : \mathbb{R}^\Omega \longrightarrow \mathcal{Y}$*, where* $\mathcal{Y}$ *is any set, is said to be translation invariant if, for all* $m, n \in \{0, ..., d-1\}$*, all* $X \in \mathbb{R}^\Omega$*,*

$$f(S^m X (S^n)^\top) = f(X).$$

Equivalently,

Definition 6.3 (Translation Equivariance)**.** *A function* $f : \mathbb{R}^\Omega \longrightarrow \mathbb{R}^\Omega$ *is said to be translation equivariant if, for all* $m, n \in \{0, ..., d-1\}$*, all* $X \in \mathbb{R}^\Omega$*,*

$$f(S^m X (S^n)^\top) = S^m f(X) (S^n)^\top.$$

Next, we will se how convolutions lie at the heart of developing translation equivariant architectures.

6.3 Convolution

Now, we will explain what is the convolution operator. Furthermore, we will prove that convolutions are both linear and translation equivariant. More than that, we will prove that every translation equivariant linear function can be written as a convolution, showing expressivity.

Definition 6.4 (Convolution)**.** *Let $x : \Omega \longrightarrow \mathbb{R}$ be the input signal and $w : \{0, ..., k-1\} \times \{0, ..., k-1\} \longrightarrow \mathbb{R}$ be the convolution kernel. The convolution operator $*$ is defined as:*

$$(x * w)(i, j) = \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} w(m, n)\, x(i - m, j - n),$$

where $x * w : \Omega \longrightarrow \mathbb{R}$.

This can be interpreted as sliding the kernel w over the signal x and computing inner products.

Notably, $x(i - m, j - n)$ may fall ourside the domain $\{0, ..., d-1\} \times \{0, ..., d-1\}$. In that case, a *padding rule* is necessary. Usually, people set values outside the domain to be 0. However, this breaks equivariance. Here, we will consider circular padding. That is,

$$x(i - m, j - n) = x(i - m \bmod d, j - n \bmod d).$$

Next, we prove that convolution is linear and translation equivariant

Theorem 6.1 (Convolution Linearity)**.** *Let $x, y : \{0, ..., d-1\} \times \{0, ..., d-1\} \longrightarrow \mathbb{R}$ be input signals and $w : \{0, ..., k-1\} \times \{0, ..., k-1\} \longrightarrow \mathbb{R}$ be a convolution kernel. Let $\alpha \in \mathbb{R}$ Then,*

$$(x * w) + (y * w) = (x + y) * w$$

and

$$\alpha(x * w) = (\alpha x) * w.$$

Proof. Let $i, j \in \{0, ..., d-1\}$. First, note that

$$\begin{aligned}
((x + y) * w)(i, j) &= \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} w(m, n)\, (x + y)(i - m, j - n) \\
&= \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} w(m, n)\, (x(i - m, j - n) + y(i - m, j - n)) \\
&= \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} w(m, n)\, x(i - m, j - n) + w(m, n)\, y(i - m, j - n) \\
&= \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} w(m, n)\, x(i - m, j - n) + \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} w(m, n)\, y(i - m, j - n) \\
&= ((x * w) + (y * w))(i, j)
\end{aligned}$$

Therefore, $(x * w) + (y * w) = (x + y) * w$. Likewise,

$$\begin{aligned}
\alpha(x * w)(i,j) &= \alpha \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} w(m,n)\, x(i-m, j-n) \\
&= \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} w(m,n)\, \alpha x(i-m, j-n) \\
&= \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} w(m,n)\, (\alpha x)(i-m, j-n) \\
&= ((\alpha x) * w)(i,j)
\end{aligned}$$

Therefore, $\alpha(x * w) = (\alpha x) * w$. □

Theorem 6.2 (Convolution Translation Equivariance)**.** *Let $x : \{0, ..., d-1\} \times \{0, ..., d-1\} \longrightarrow \mathbb{R}$ be an input signal and $w : \{0, ..., k-1\} \times \{0, ..., k-1\} \longrightarrow \mathbb{R}$ be a convolution kernel. Let $m, n \in \{0, ..., d-1\}$ and let $\varphi_{m,n}$ be a translation. Then,*

$$(\varphi_{m,n} \cdot x) * w = \varphi_{m,n} \cdot (x * w).$$

Proof. Let $i, j \in \{0, ..., d-1\}$. Note that,

$$\begin{aligned}
(\varphi_{m,n} \cdot (x * w))(i,j) &= (x * w)(i - m \bmod d, j - n \bmod d) \\
&= \sum_{u=0}^{k-1} \sum_{v=0}^{k-1} w(u,v) x((i - m \bmod d) - u, (j - n \bmod d) - v) \\
&= \sum_{u=0}^{k-1} \sum_{v=0}^{k-1} w(u,v) x(i - m - u \bmod d, j - n - v \bmod d) \\
&= \sum_{u=0}^{k-1} \sum_{v=0}^{k-1} w(u,v) (\varphi_{m,n} \cdot x)(i-u, j-v) \\
&= ((\varphi_{m,n} \cdot x) * w)(i,j),
\end{aligned}$$

where the third equality holds because of circular padding. Therefore, $(\varphi_{m,n} \cdot x) * w = \varphi_{m,n} \cdot (x * w)$. □

Lastly, we prove that all linear translation equivariant functions can be written as convolutions.

Theorem 6.3. *Let $\mathcal{S} = \{x : \{0, ..., d-1\} \times \{0, ..., d-1\} \longrightarrow \mathbb{R}\}$. Let $f : \mathcal{S} \longrightarrow \mathcal{S}$ be linear and translation equivariant. Then, there exists $w : \{0, ..., k-1\} \times \{0, ..., k-1\} \longrightarrow \mathbb{R}$ such that*

$$f(x) = x * w$$

for all $x \in \mathcal{S}$.

Proof. Let $x \in \mathcal{S}$. Let $b \in \mathcal{S}$ such that $b(0,0) = 1$ and $b(i,j) = 0$ for all $(i,j) \neq (0,0)$. Note

that

$$x = \sum_{u=0}^{d-1} \sum_{v=0}^{d-1} x(u, v)(\varphi_{u,v} \cdot b).$$

Because of linearity, we have

$$f(x) = \sum_{u=0}^{d-1} \sum_{v=0}^{d-1} x(u, v) f(\varphi_{u,v} \cdot b).$$

And because of translation equivariance, we have

$$\begin{aligned} f(x) &= \sum_{u=0}^{d-1} \sum_{v=0}^{d-1} x(u, v) f(\varphi_{u,v} \cdot b) \\ &= \sum_{u=0}^{d-1} \sum_{v=0}^{d-1} x(u, v) \varphi_{u,v} \cdot f(b) \end{aligned}$$

Let $w = f(b)$. Note that

$$\begin{aligned} f(x)(i, j) &= \sum_{u=0}^{d-1} \sum_{v=0}^{d-1} x(u, v)(\varphi_{u,v} \cdot w)(i, j) \\ &= \sum_{u=0}^{d-1} \sum_{v=0}^{d-1} x(u, v) w(i - u \bmod d, j - v \bmod d) \end{aligned}$$

Now define $m = i - u \bmod d$ and $n = j - v \bmod d$. Applying a change of variable yields

$$\begin{aligned} f(x)(i, j) &= \sum_{m=0}^{d-1} \sum_{n=0}^{d-1} x(i - m \bmod d, j - n \bmod d) w(m, n) \\ &= (x * w)(i, j). \end{aligned}$$

Where the last equality holds because of circular padding. Therefore, $f(x) = x * w$. □

Now, we will see how Convolutional neural networks explore the idea of translation equivariance.

6.4 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) were first introduced by LeCun, Boser, *et al.*, 1989. Their defining component is the **convolutional layer**, which applies a convolution with a learned kernel to extract local, translation-equivariant features from grid-structured data.

In general terms, a standard CNN layer is the composition of three operations:

1. **Convolution**, which, under appropriate boundary conditions, ensures translation equivariance;
2. **A pointwise non-linearity**, such as ReLU;
3. **A coarsening operator**, typically a pooling layer (e.g. max or average pooling).

Although convolution itself is exactly translation-equivariant, some practical design choices in CNNs break this symmetry. As discussed in the previous section, *zero-padding* introduces boundary artifacts that violate equivariance at the image edges. Likewise, *max pooling* breaks perfect equivariance, yielding only approximate translation equivariance.

After a sequence of convolutional and pooling layers, it is common to apply a multi-layer perceptron (MLP) to map the learned feature maps to task-specific outputs. Importantly, this does *not* yield translation invariance. To obtain invariance, one must first apply a spatial aggregation operator, such as global average pooling or summation, before passing the result through an MLP. This idea follows directly from the invariant layer of the GDL blueprint.

Finally, translations are not the only symmetries present in visual data. For instance, rotating an image of a dog still yields an image recognizable as the same dog. This motivates extending convolution to be equivariant to larger groups, such as rotations and reflections. This line of work was initiated by Cohen and Welling with Group-Equivariant CNNs (G-CNNs) T. Cohen and Welling, 2016. Further developments include steerable CNNs (Weiler and Cesa, 2019) and spherical CNNs (T. S. Cohen *et al.*, 2018). A detailed discussion of these architectures is beyond the scope of this chapter.

Chapter 7

GDL in Action

This chapter presents a series of controlled experiments designed to illustrate, in practice, the concepts developed throughout this work. While previous chapters focused on formalizing the Geometric Deep Learning perspective—its motivation, theoretical foundations, and architectural principles—the results presented here demonstrate how these ideas translate into concrete implementations. Rather than aiming for state-of-the-art performance or extensive hyperparameter tuning, the primary objective is to compare reasonable baselines: a universal approximator model and architectures informed by GDL principles. Through these comparisons, we investigate how geometric inductive biases influence model behavior and performance. Additionally, this chapter serves as an opportunity to transition from high-level conceptual discussion to practical implementation details, highlighting considerations that arise when turning theory into code.

The experiments are organized progressively. We begin with sets in a point cloud classification setting, followed by graphs applied to molecular property prediction, and conclude with grids in an image classification task.

All code used to run the experiments is publicly available at this url.

7.1 Sets

We begin by examining the Geometric Deep Learning setting in the context of sets, focusing on a point cloud classification task. The corresponding subsection is structured in two parts. The first presents the methodological framework, including architectural choices, data preprocessing, and relevant hyperparameter configurations. The second part provides the experimental results, followed by ablation studies and a discussion of the outcomes.

7.1.1 Method

We begin by describing the dataset used for point cloud classification. ModelNet10 (Wu *et al.*, 2015) is a widely used benchmark dataset for 3D object recognition. It contains 4,899 CAD models across 10 object categories, including chairs, sofas, tables, and bathtubs. Exact classes and object count are provided in Table 7.1. The dataset provides consistent

train–test splits and serves as a standard evaluation setting for neural networks operating on unordered 3D point clouds. ModelNet10 is frequently used to test architectures that incorporate permutation invariance or geometric inductive biases, making it a suitable choice for evaluating Set-based learning methods in the context of Geometric Deep Learning. From each CAD model, we sample 1,024 points, which constitute the input.

Class	Train	Validation	Total
bathtub	106	50	156
bed	515	100	615
chair	889	100	989
desk	200	86	286
dresser	200	86	286
monitor	465	100	565
nightstand	200	86	286
sofa	680	100	780
table	392	100	492
toilet	344	100	444
Total	**3991**	**908**	**4899**

Table 7.1: *Number of samples per class in the ModelNet10 dataset. Source: Dale, 2024*

For the permutation-invariant model, we use DeepSets (Zaheer *et al.*, 2017). The first MLP, which learns equivariant representations, has three layers. The input size is 3 (corresponding to the 3D coordinates), and the remaining layers use a dimensionality of 256. The resulting 256-dimensional representation for each point is aggregated using an average operator and then fed to a second MLP with two layers. This second MLP has an input and hidden size of 256, and its final layer outputs dimension 10. ReLU activations are used throughout, except in the final layer of the second MLP, which uses a softmax for classification.

We compare DeepSets against a three-layer MLP baseline. We flatten the points to form the input, resulting in a feature dimension of 3,072. The model uses a hidden dimension of 4,096 and an output dimension of 10. As before, ReLU activations are used, with softmax applied at the final layer.

We apply batch normalization (Ioffe and Szegedy, 2015) between the layers of all MLPs to improve convergence.

For training, we use a batch size of 16, a learning rate of 0.001 for 100 epochs, the Adam optimizer (Kingma and Ba, 2015), and a logarithmic learning rate scheduler, which multiplies the learning rate by 0.95 after each epoch. Minimal hyperparameter tuning was performed, as the goal of these experiments is to illustrate the effects of invariance and equivariance rather than to achieve optimal performance. As fot the loss, we use Cross Entropy.

We note that the DeepSet model has 202,506 trainable parameters, while the MLP has 29,425,674 parameters. We attempted to train an MLP with fewer parameters, but results were very poor (close to random guessing). Therefore, we chose to perform the

comparison using the larger MLP. This further illustrates how encoding appropriate geometric inductive biases can improve parameter efficiency.

7.1.2 Results and Ablation

We begin by comparing the training and validation performance of each model. Figure 7.1 shows the training and validation metrics for both approaches.

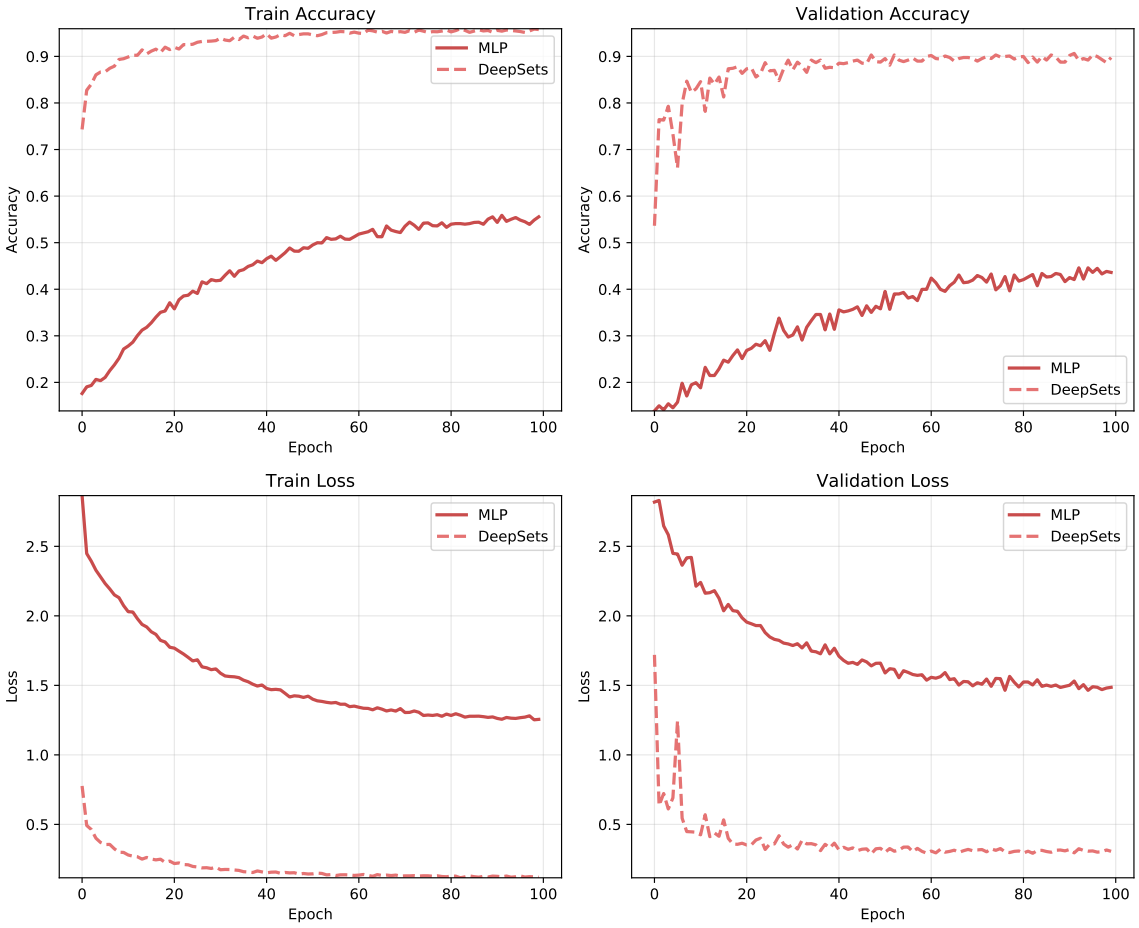

Figure 7.1: *Training and validation loss and accuracy for DeepSets and the MLP on ModelNet10. DeepSets achieve substantially higher performance while using far fewer parameters.*

We observe that DeepSets significantly outperform the MLP. In terms of validation accuracy, the MLP reaches a maximum value of 44.6% and converges to 43.61%, while DeepSets reach a maximum value of 90.64% and finish at 89.76%, despite using only 0.69% of the parameters. This highlights how geometric inductive biases can simultaneously improve performance and parameter efficiency (through weight sharing).

To further investigate the role of permutation invariance in this task, we select four examples from the validation set and randomly permute the point order before feeding them to the networks. We repeat this process 500 times and measure how the predictions change. Figure 7.2 presents the results.

From these results, we observe that DeepSets predictions remain consistent under

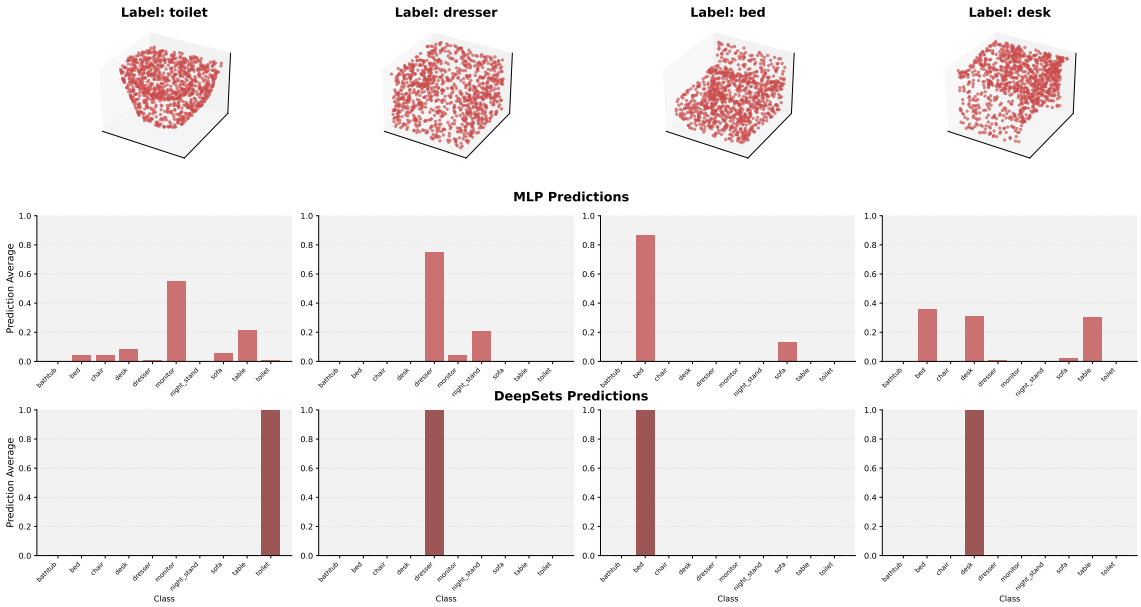

Figure 7.2: *Average class predictions across random point permutations for the MLP and DeepSets. Permutations significantly affect the MLP's predictions but not those of DeepSets, as expected.*

input permutations, as expected from a permutation-invariant architecture. In contrast, the MLP is noticeably affected by point reordering, demonstrating its sensitivity to input arrangement.

7.2 Graphs

Next, we explore GDL in the context of Graphs, using a molecule classification task. This section is, as the one before, structured in two parts. The first presents the methodological framework and the second part provides the experimental results, followed by ablation studies and a discussion of the outcomes.

7.2.1 Method

For the molecular property prediction task, we use the MUTAG dataset (Debnath *et al.*, 1991). MUTAG is a widely used benchmark in graph learning and cheminformatics, containing 188 chemical compounds. Each molecule is represented as a graph, where nodes correspond to atoms and edges represent chemical bonds. Node features encode atom types, while edge attributes indicate different bond categories—although in our experiments we only use node features. The task is a binary classification problem: predicting whether a compound is mutagenic on a specific strain of Salmonella typhimurium.

Table 7.2 reports the train–test split and class distribution used in our experiments. As there is no official split, we use random ones.

For the GNN model, we use a Graph Isomorphism Network (GIN) (Xu *et al.*, 2018). A GIN is a neural architecture designed to match the discriminative power of the We-

Category	Train	Validation	Total
Mutagenic	104	21	125
Non-mutagenic	46	17	63
Total Samples	150	38	188

Table 7.2: *Train–test split and class distribution for the MUTAG dataset.*

isfeiler–Lehman (WL) graph isomorphism test, a classical algorithm used to determine whether two graphs are structurally identical. The update rule of a layer is defined as follows:

$$\mathbf{h}_v = \mathrm{MLP}\Big((1+\epsilon)\,\mathbf{h}_v \;+\; \sum_{u\in\mathcal{N}(v)} \mathbf{h}_u\Big),$$

where $\mathbf{h}_v$ denotes the representation of node v and ϵ can be either learned or fixed.

In our experiments, we use three GIN layers. In all of them, ϵ is set to 0 and we use 2-layer MLPs. The first MLP has an input dimension of 7 (corresponding to the one-hot encoding of atom types), and all remaining dimensions are set to 64. The final node representations are aggregated via summation and fed into a single-layer MLP with output size 2, followed by a softmax function.

For the MLP baseline, we construct the adjacency and feature matrices for each molecule. These matrices are padded to match the maximum number of atoms in the dataset (28). We then flatten and concatenate both matrices and feed the result into a 3-layer MLP. This model has an input size of 980, a hidden dimension of 64, and an output dimension of 2. The loss function used is again cross entropy. Again, batch normalization (Ioffe and Szegedy, 2015) is used in all MLPs.

The MLP baseline contains a total of 67,330 trainable parameters, whereas the GNN model contains 21,826 parameters.

For training, we use a batch size of 32. We follow the same setup as before and use a learning rate of 0.001 for 100 epochs, the Adam optimizer (Kingma and Ba, 2015), and a logarithmic learning rate scheduler, which multiplies the learning rate by 0.95 after each epoch.

7.2.2 Results and Ablation

Again, we begin by comparing the training and validation performance of each model. Figure 7.3 shows the training and testing metrics for both approaches.

Unlike the previous experiment, the performance of both methods is very close. On the validation set, both models reach the same maximum accuracy of 89.47%, although they converge to slightly different final values. The MLP finishes at 86.84%, whereas the GNN converges to 89.47%.

To better illustrate the effect of permutation invariance, we perform an analogous ablation study: we select four validation examples and randomly permute the order of

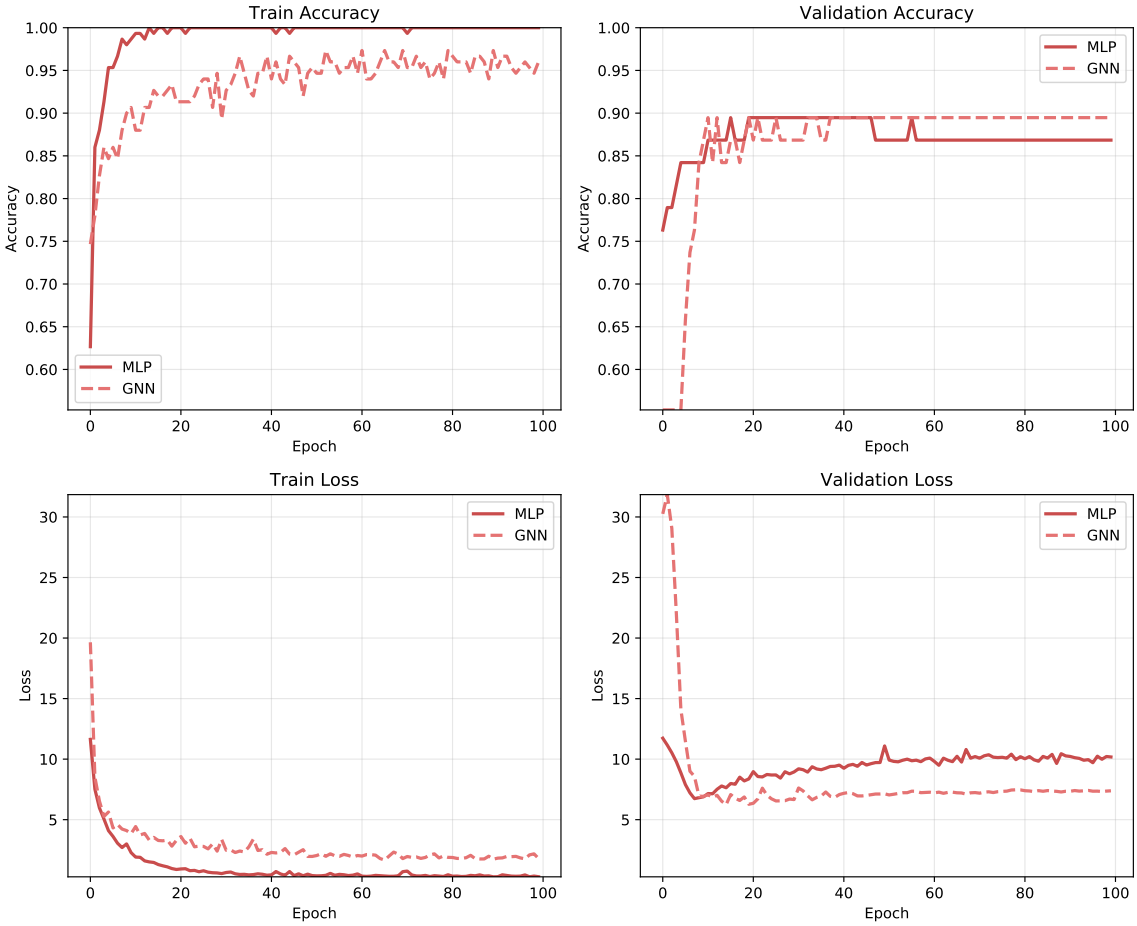

Figure 7.3: *Training and validation loss and accuracy for the GNN and the MLP on MUTAG. Although the MLP shows slight overfitting, the overall performance remains very similar.*

their nodes before feeding them to both networks. This process is repeated 500 times, and we examine the average predictions. Figure 7.4 summarizes the results.

Overall, although both models achieve comparable accuracy, the ablation highlights a key difference: the GNN remains robust to input permutations, as guaranteed by its permutation-invariant architecture, whereas the MLP is noticeably affected by node reordering, demonstrating its sensitivity to the input structure.

7.3 Grids

Finally, we explore GDL in the context of Grids, performing an image classification task. This section is, structured in the same way as the previous ones. The first presents the methodological framework and the second part provides the experimental results, followed by ablation studies and a discussion of the outcomes.

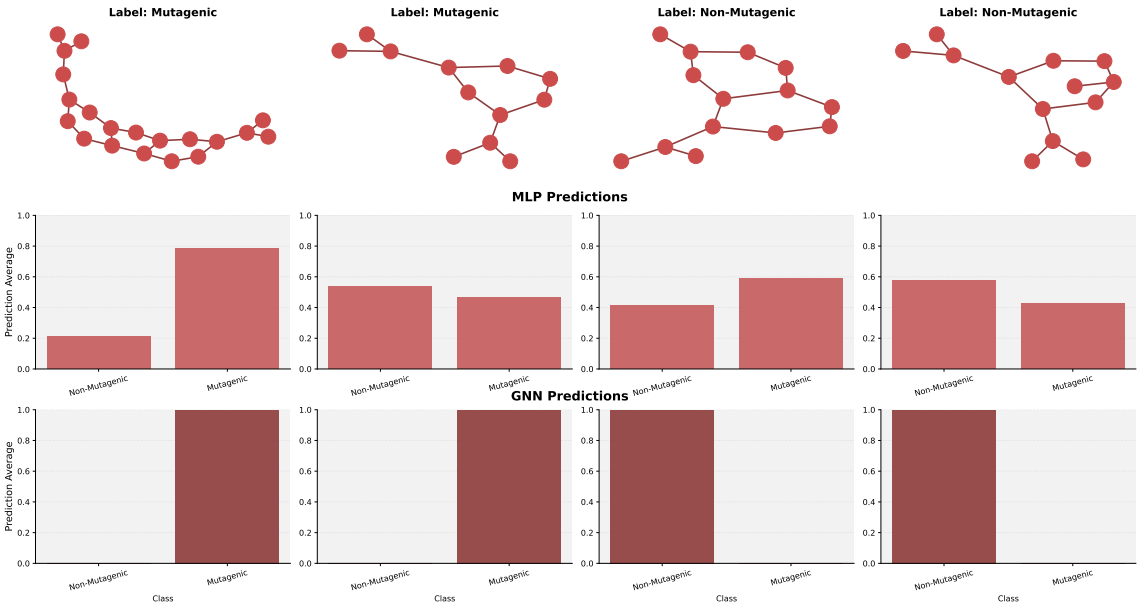

Figure 7.4: *Average class predictions across random node permutations for the MLP and GNN. As expected, permutations significantly affect the MLP's predictions but not those of the GNN.*

7.3.1 Method

For the image classification task, we use the MNIST dataset (LeCun, Bottou, *et al.*, 2002). The MNIST dataset is one of the most widely used benchmarks in Machine Learning and image-processing for handwritten digit classification. It consists of 28 × 28 pixel grayscale images of the digits 0 through 9, each labelled with the corresponding digit. Because the dataset is balanced across the ten classes and relatively small, it serves as a convenient starting point for benchmarking classification architectures. Table 7.3 presents class counts over train and validation.

Class	Train Count	Validation Count
0	5923	980
1	6742	1135
2	5958	1032
3	6131	1010
4	5842	982
5	5421	892
6	5918	958
7	6265	1028
8	5851	974
9	5949	1009
Total	**60000**	**10000**

Table 7.3: *Number of samples per class in the MNIST dataset across the training and testing splits.*

For the CNN model, we use four convolutional layers with ReLU activations and batch

normalization. The convolutional layers do not include bias terms, and padding is set to circular mode to improve equivariance. The first two layers use 32 channels, and the last two use 64 channels. Between the second and third layers, we apply a BlurPool coarsening operator, i.e., we convolve with a blur filter and downscale the feature maps to half their original size (from 28×28 to 14×14). Although coarsening operations usually break perfect equivariance, this strategy is known to help mitigate the issue (Zhang, 2019). After the final convolutional layer, we average each channel over the spatial dimensions to obtain (approximately) invariant representations. The resulting 64-dimensional vector is fed into a linear layer with input dimension 64 and output dimension 10, which is then followed by a softmax to produce class probabilities.

For the MLP baseline, we flatten the images into 784-dimensional vectors. Each vector is then fed into a 3-layer MLP with hidden dimension 64, ReLU activations, and a final softmax. The MLP has 55,306 parameters, while the CNN has 16,794 parameters.

For training, we use a batch size of 32. We set the learning rate to 0.001 for 20 epochs, use the Adam optimizer (Kingma and Ba, 2015), and apply a logarithmic learning rate scheduler that multiplies the learning rate by 0.95 after each epoch.

7.3.2 Results and Ablation

Figure 7.5 shows the training and testing metrics for both approaches.

In terms of validation accuracy, the MLP achieves a maximum value of 98.21% and converges to 97.98%, while the CNN reaches a maximum of 99.34% and converges to 98.91%. Although the difference is relatively small, the CNN outperforms the MLP while using fewer parameters.

To better illustrate the effect of translation invariance, we perform an analogous ablation study as before: we select four validation examples and randomly translate the pixels before feeding them to both networks. This process is repeated 500 times, and we examine the average predictions. Figure 7.6 summarizes the results.

Overall, although both models achieve comparable accuracy, the ablation highlights a key distinction: the CNN remains far more robust to input translations (although not perfectly invariant due to the coarsening step), whereas the MLP is noticeably affected by such transformations, demonstrating its sensitivity to the input structure.

7.4 Discussion

Across the three experimental domains—sets, graphs, and grids—the results consistently demonstrate the value of incorporating geometric inductive biases into model design. In each setting, symmetry-aware architectures achieved competitive or superior performance in the validation set, while requiring significantly fewer parameters than their unconstrained MLP baselines, therefore showcasing better generalization capability. Beyond accuracy, the ablation studies revealed an even clearer distinction: models respecting the symmetry of the input domain behaved predictably under permissible transformations, whereas the baselines were highly sensitive to them.

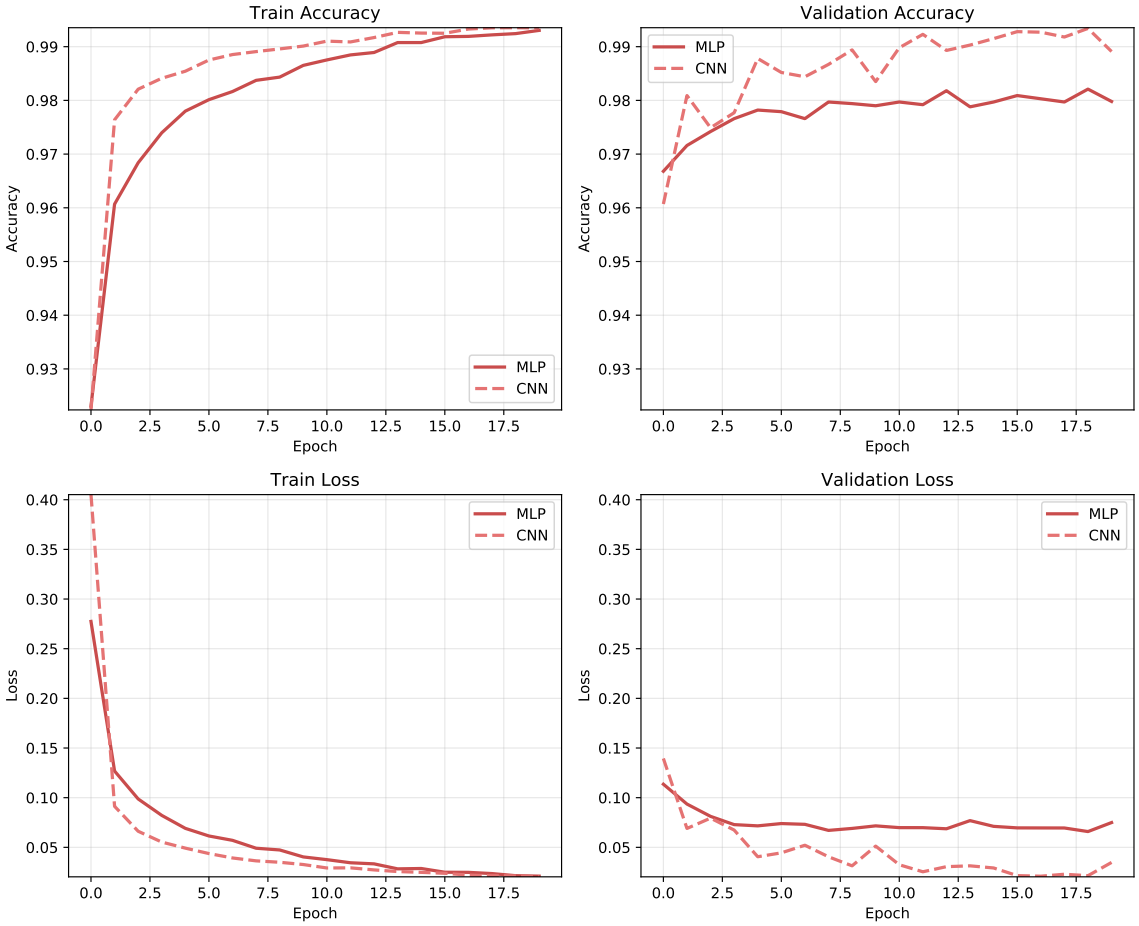

Figure 7.5: *Training and validation loss and accuracy for the CNN and the MLP on MNIST. The CNN slightly outperforms the MLP.*

DeepSets showed strong advantages in the set-based experiment, both in accuracy and invariance to point permutations. In the graph task, the performance gap between GIN and the MLP was smaller, yet the GIN remained stable under node reordering, confirming its alignment with the relational structure of the data. Finally, in the grid experiment, the CNN demonstrated improved robustness to translations relative to the MLP, reflecting the approximate equivariance built into convolutional architectures.

Overall, these results empirically validate the theoretical framework developed in the previous chapters, demonstrating that the principles of symmetry, equivariance, and invariant representation are not only mathematically grounded but also practically beneficial across diverse learning settings.

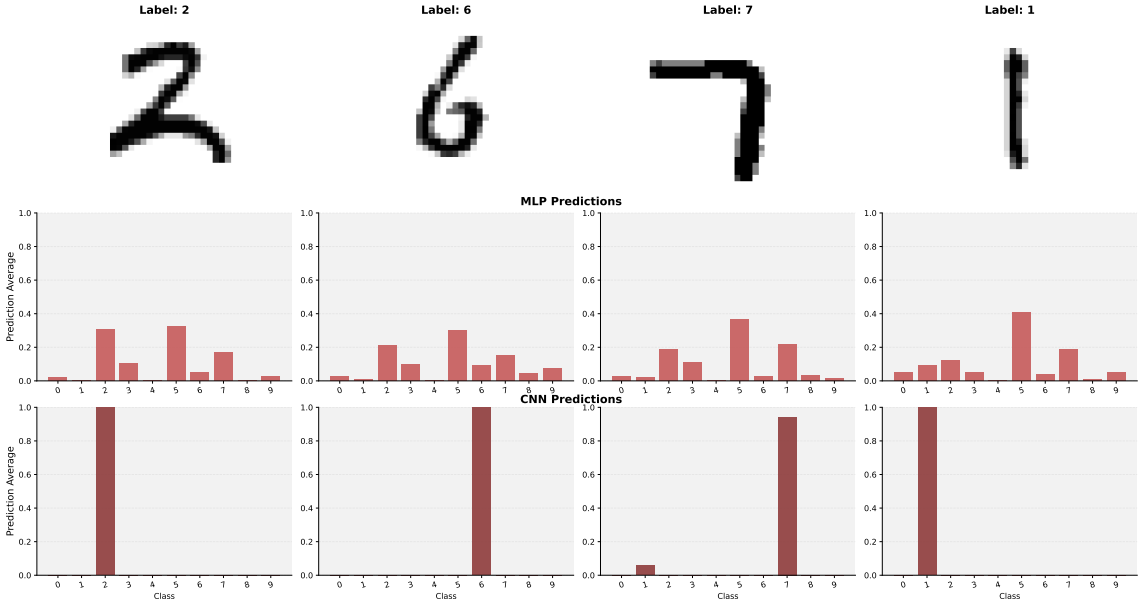

Figure 7.6: *Average class predictions across random translations for the MLP and CNN. As expected, translations significantly affect the MLP's predictions but not those of the CNN.*

Chapter 8

Conclusion

This work explored Geometric Deep Learning as a principled framework for understanding and constructing neural network architectures through symmetry. By focusing on three fundamental geometric domains—Sets, Graphs, and Grades—we showed how invariance and equivariance naturally guide architecture design and help address challenges such as sample inefficiency and the curse of dimensionality. Rather than presenting GDL in its full mathematical generality, the thesis prioritized intuition, foundational concepts, and simplified formal tools, making the topic accessible to readers at the undergraduate level.

Throughout the chapters, we demonstrated how canonical models such as DeepSets, Graph Neural Networks, and Convolutional Neural Networks emerge directly from geometric priors embedded in the data domain. Lastly, the experimental results reinforced the theoretical foundations developed in earlier sections. While this work does not attempt to exhaustively cover the rapidly growing field of GDL, it provides a structured entry point and a conceptual blueprint for further study.

References

[Abu-Mostafa *et al.* 2012] Yaser S. Abu-Mostafa, Malik Magdon-Ismail, and Hsuan-Tien Lin. *Learning From Data*. AMLBook, 2012 (cit. on p. 5).

[Borde and M. Bronstein 2025] Haitz Sáez de Ocáriz Borde and Michael Bronstein. "Mathematical foundations of geometric deep learning". *arXiv preprint arXiv:2508.02723* (2025) (cit. on pp. 1, 8).

[Michael M Bronstein *et al.* 2021] Michael M Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. "Geometric deep learning: grids, groups, graphs, geodesics, and gauges". *arXiv preprint arXiv:2104.13478* (2021) (cit. on pp. 1, 2, 5, 8, 9, 17, 23, 28, 30).

[Michael M. Bronstein 2021] Michael M. Bronstein. *Geometric Deep Learning – Lecture 2*. YouTube. Accessed: 2025-11-24. 2021. URL: https://www.youtube.com/watch?v=4RmpSvQ2LL0 (cit. on pp. 5, 8).

[Bronstein, Michael M. 2022a] Bronstein, Michael M. *Geometric Deep Learning – Lecture 5*. https://www.youtube.com/watch?v=J2bLt3-SSpg. Online; accessed 2025-12-11. 2022 (cit. on pp. 17, 23).

[Bronstein, Michael M. 2022b] Bronstein, Michael M. *Geometric Deep Learning – Lecture 6*. https://www.youtube.com/watch?v=HvQw7Zq1jtU. Online; accessed 2025-12-11. 2022 (cit. on p. 23).

[Bronstein, Michael M. 2022c] Bronstein, Michael M. *Geometric Deep Learning – Lecture 6*. https://www.youtube.com/watch?v=Mj7UoabhWYQ&list=PLn2-dEmQeTfSLXW8yXP4q_Ii58wFdxb3C&index=7. Online; accessed 2025-12-11. 2022 (cit. on p. 30).

[T. Cohen and Welling 2016] Taco Cohen and Max Welling. "Group equivariant convolutional networks". In: *International conference on machine learning*. PMLR. 2016, pp. 2990–2999 (cit. on p. 35).

[T. S. Cohen *et al.* 2018] Taco S Cohen, Mario Geiger, Jonas Köhler, and Max Welling. "Spherical CNNs". In: *International Conference on Learning Representations*. 2018 (cit. on p. 35).

[Cybenko 1989] George Cybenko. "Approximation by superpositions of a sigmoidal function". *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314 (cit. on p. 20).

[Dale 2024] DJA van Dale. "Utilising 3d gaussian splatting for pointnet object classification" (2024) (cit. on p. 37).

[Debnath *et al.* 1991] Asim Kumar Debnath, Rosa L Lopez de Compadre, Gargi Debnath, Alan J Shusterman, and Corwin Hansch. "Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity". *Journal of medicinal chemistry* 34.2 (1991), pp. 786–797 (cit. on p. 39).

[Deng *et al.* 2009] Jia Deng *et al.* "Imagenet: a large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255 (cit. on p. 9).

[Gavranović *et al.* 2024] Bruno Gavranović *et al.* "Position: categorical deep learning is an algebraic theory of all architectures". *arXiv preprint arXiv:2402.15332* (2024) (cit. on p. 4).

[Hajij *et al.* 2022] Mustafa Hajij *et al.* "Topological deep learning: going beyond graph data". *arXiv preprint arXiv:2206.00606* (2022) (cit. on p. 4).

[Hornik 1991] Kurt Hornik. "Approximation capabilities of multilayer feedforward networks". *Neural networks* 4.2 (1991), pp. 251–257 (cit. on p. 20).

[Ioffe and Szegedy 2015] Sergey Ioffe and Christian Szegedy. "Batch normalization: accelerating deep network training by reducing internal covariate shift". In: *International conference on machine learning*. pmlr. 2015, pp. 448–456 (cit. on pp. 37, 40).

[Jolliffe and Cadima 2016] Ian T Jolliffe and Jorge Cadima. "Principal component analysis: a review and recent developments". *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 374.2065 (2016), p. 20150202 (cit. on p. 7).

[Joshi 2020] Chaitanya Joshi. "Transformers are graph neural networks". *The Gradient* (2020) (cit. on pp. 2, 28).

[Kingma and Ba 2015] Diederik P. Kingma and Jimmy Ba. "Adam: a method for stochastic optimization". In: *International Conference on Learning Representations (ICLR)*. 2015 (cit. on pp. 37, 40, 43).

[Krizhevsky *et al.* 2012] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". *Advances in neural information processing systems* 25 (2012) (cit. on p. 4).

[LeCun, Boser, *et al.* 1989] Yann LeCun, Bernhard Boser, *et al.* "Backpropagation applied to handwritten zip code recognition". *Neural computation* 1.4 (1989), pp. 541–551 (cit. on pp. 2, 4, 34).

[LeCun, Bottou, *et al.* 2002] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition". *Proceedings of the IEEE* 86.11 (2002), pp. 2278–2324 (cit. on pp. 4, 42).

[Russakovsky *et al.* 2015] Olga Russakovsky *et al.* "Imagenet large scale visual recognition challenge". *International journal of computer vision* 115.3 (2015), pp. 211–252 (cit. on p. 4).

[Vaswani *et al.* 2017] Ashish Vaswani *et al.* "Attention is all you need". *Advances in neural information processing systems* 30 (2017) (cit. on pp. 2, 28).

[Weiler and Cesa 2019] Maurice Weiler and Gabriele Cesa. "General E(2)-equivariant steerable CNNs". In: *Advances in Neural Information Processing Systems*. Vol. 32. 2019 (cit. on p. 35).

[Weisstein 2025] Eric W. Weisstein. *Hypercube Line Picking. MathWorld – A Wolfram Resource*. Accessed: 2025-11-24. 2025. URL: https://mathworld.wolfram.com/HypercubeLinePicking.html (cit. on p. 7).

[Wu *et al.* 2015] Zhirong Wu *et al.* "3d shapenets: a deep representation for volumetric shapes". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 1912–1920 (cit. on p. 36).

[Xu *et al.* 2018] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. "How powerful are graph neural networks?" *arXiv preprint arXiv:1810.00826* (2018) (cit. on pp. 2, 28, 39).

[Zaheer *et al.* 2017] Manzil Zaheer *et al.* "Deep sets". *Advances in neural information processing systems* 30 (2017) (cit. on pp. 2, 19, 22, 37).

[Zhang 2019] Richard Zhang. "Making convolutional networks shift-invariant again". In: *International conference on machine learning*. PMLR. 2019, pp. 7324–7334 (cit. on p. 43).