RIPSNET: A GENERAL ARCHITECTURE FOR FAST AND ROBUST ESTIMATION OF THE PERSISTENT HOMOLOGY OF POINT CLOUDS

Felix Hensel*, Marc Glisse & Frédéric Chazal
Université Paris-Saclay, CNRS, Inria, Laboratoire de Mathématiques d’Orsay, France

Thibault de Surrel* & Mathieu Carrière*
Université Côte d’Azur, Inria, France

Théo Lacombe
LIGM, Laboratoire Informatique Gaspard Monge
Univ Gustave Eiffel, CNRS, LIGM, F-77454 Marne-la-Vallée, France

Hiroaki Kurihara
Fujitsu Ltd., Kanagawa, Japan

Yuichi Ike
Graduate School of Information Science and Technology
The University of Tokyo, Japan

ABSTRACT

The use of topological descriptors in modern machine learning applications, such as persistence diagrams (PDs) arising from Topological Data Analysis (TDA), has shown great potential in various domains. However, their practical use in applications is often hindered by two major limitations: the computational complexity required to compute such descriptors exactly, and their sensitivity to even low-level proportions of outliers. In this work, we propose to bypass these two burdens in a data-driven setting by entrusting the estimation of (vectorization of) PDs built on top of point clouds to a neural network architecture that we call RipsNet. Once trained on a given data set, RipsNet can estimate topological descriptors on test data very efficiently with generalization capacity. Furthermore, we prove that RipsNet is robust to input perturbations in terms of the 1-Wasserstein distance, a major improvement over the standard computation of PDs that only enjoys Hausdorff stability, yielding RipsNet to substantially outperform exactly-computed PDs in noisy settings. We showcase the use of RipsNet on both synthetic and real-world data. Our implementation will be made freely and publicly available as part of the open-source library Gudhi.

1 INTRODUCTION

The knowledge of topological features (such as connected components, loops, and higher dimensional cycles) that are present in data sets provides a better understanding of their structural properties at multiple scales, and can be leveraged to improve statistical inference and prediction. Topological Data Analysis (TDA) is the branch of data science that aims to detect and encode such topological features, in the form of persistence diagrams (PD). A PD is a (multi-)set of points $D$ in $\mathbb{R}^2$, in which each point $p \in D$ corresponds to a topological feature of the data whose size is encoded by its coordinates. PDs are descriptors of a general nature and allow flexibility in their computation. As such, they have been successfully applied to many different areas of data science, including, e.g., material science (Buchet et al., 2018), genomic data (Camara, 2017), and 3D-shapes (Li et al., 2014). In the present work, we focus on PDs stemming from point cloud data, referred to as Rips PDs, which find natural use in shape analysis (Chazal et al., 2009; Gamble & Heo, 2010) but also in other domains such as time series analysis (Perea & Harer, 2015; Pereira & de Mello, 2015; Umeda, 2017).

*These authors contributed equally to the work.

[https://gudhi.inria.fr/](https://gudhi.inria.fr/)
or in the study of the behavior of deep neural networks \cite{guss2018,naitzat2020,birdal2021}.

A drawback of Rips PDs computed on large point clouds is their computational cost. Furthermore, even though these topological descriptors enjoy stability properties with respect to the input point cloud in the Hausdorff metric \cite{chazal2014}, they are fairly sensitive to perturbations: moving a single point in an arbitrarily large point cloud can alter the resulting Rips PD substantially.

In addition, the lack of linear structure (such as addition and scalar multiplication) of the space of PDs hinders the use of PDs in standard machine learning pipelines, which are typically developed to handle inputs belonging to a finite dimensional vector space. This burden motivated the development of vectorization methods, which allow to map PDs into a vector space while preserving their structure and interpretability. Vectorization methods can be divided into two classes, finite-dimensional embeddings \cite{bubenik2015,adams2017,carrierre2015,chazal2015,kaliskik2018}, turning PDs into elements of Euclidean space \( \mathbb{R}^d \), and kernels \cite{carrierre2017,kusano2016,le2018, reiminghaus2015}, that implicitly map PDs to elements of infinite-dimensional Hilbert spaces.

In this work, we propose to overcome the previous limitations of Rips PDs, by learning their finite-dimensional embeddings directly from the input point cloud data sets with neural network architectures. This approach allows not only for a much faster computation time, but also for increased robustness of the topological descriptors.

**Contributions.** More specifically, our contributions in this work are summarized as follows.

- We introduce RipsNet, a DeepSets-like architecture capable of learning finite-dimensional embeddings of Rips PDs built on top of point clouds.
- We study the robustness properties of RipsNet. In particular, we prove that for a given point cloud \( X \), perturbing a proportion \( \lambda \in (0, 1) \) of its points can only change the output of RipsNet by \( O(\lambda) \), while the exact persistence diagram may change by a fixed positive quantity even in the regime \( \lambda \to 0 \).
- We experimentally showcase how RipsNet can be trained to produce fast, accurate, and useful estimations of topological descriptors. In particular, we observe that using RipsNet outputs instead of exact PDs yields better performances for classification tasks based on topological properties.

**Related work.** Our RipsNet architecture is directly based on DeepSets \cite{zaheer2017}, a particular case of equivariant neural network \cite{cohen2021} designed to handle point clouds as inputs. Namely, DeepSets essentially consists of processing a point cloud \( X = \{ x_1, \ldots, x_n \} \subset \mathbb{R}^d \) via

\[
X \mapsto \phi_2(\text{op}(\{ \phi_1(x_i) \}_{i=1}^n)),
\]

where \( \text{op} \) is a permutation invariant operator on sets (such as sum, mean, maximum, etc.) and \( \phi_1 : \mathbb{R}^d \to \mathbb{R}^{d'} \) and \( \phi_2 : \mathbb{R}^{d'} \to \mathbb{R}^{d''} \) are parametrized maps (typically encoded by neural networks) optimized in the training phase. Eq. 1 makes the output of DeepSets architectures invariant to permutations, a property of Rips PDs that we want to reproduce in RipsNet. Note also that DeepSets have also been used recently in TDA for defining the PersLay layer \cite{carrierre2020} for PDs whose is to learn how to compute the best PD embedding w.r.t. a given data science task. Even though PersLay and RipsNet have similar architectures, their goals are different, as RipsNet does not aim at learning the best PD embedding, but at approximating the PD computation itself.

There exist a few works attempting to compute or estimate (vectorizations of) PDs through the use of neural networks. In \cite{som2020}, the authors propose a convolutional neural network (CNN) architecture to estimate persistence images (see below) computed on 2D-images. Similarly, in \cite{montufar2020}, the authors provide an experimental overview of specific PD features (such as, e.g., their tropical coordinates \cite{kalisnik2019}) that can be learned using a CNN, when PDs are computed on top of 2D-images. On the other hand, RipsNet is designed to handle the (arguably harder) situation where input data are point clouds of arbitrary cardinality instead of 2D-images (i.e., vectors). Very recently, \cite{yan2022} addressed a similar question in the context of graphs. Fairly similar to our approach, the recent work \cite{zhou2021} also aims at learning to compute topological descriptors on top of point clouds via a neural network. However, note that
our methodology is quite different: while our approach based on DeepSets architectures allows to process point clouds directly, the approach proposed in [Zhou et al. 2021] requires the user to equip the point clouds with graph structures (that depend on hyper-parameters mimicking Rips filtrations) and is suited to 3D point clouds only. Furthermore, a key difference between our approach and the aforementioned works is that we provide a theoretical study of our model that provides insights on its behavior, particularly in terms of robustness to noise (which shows in our experimental results, see Section 4.2 and Table I), while the other works are mostly experimental.

Supplementary material. Proofs of theoretical results, as well as additional experiments on UCR-timeseries data and additional persistence diagram vectorization, can be found in (de Surrel et al. 2022). Code will be incorporated in future versions of Gudhi, a version to reproduce the experiments is provided at https://github.com/hensel-f/ripsnet.

2 Background

In this section, we recall the basics of persistence theory. We refer the interested reader to (Cohen-Steiner et al. 2009) Edelsbrunner & Harer (2010) Oudot (2015) for a thorough treatment.

Persistence diagrams. Let $\mathcal{X}$ be a topological space, and $f: X \to \mathbb{R}$ a real-valued continuous function. The $\alpha$-sublevel set of $(\mathcal{X}, f)$ is defined as $X_\alpha = \{ x \in X : f(x) \leq \alpha \}$. Increasing $\alpha$ from $-\infty$ to $+\infty$ yields an increasing nested sequence of sublevel sets, called the filtration induced by $f$. It starts with the empty set and ends with the entire space $X$. Ordinary persistence keeps track of the times of appearance and disappearance of topological features (connected components, loops, cavities, etc.) in this sequence. For instance, one can store the value $\alpha_b$, called the birth time, for which a new connected component appears in $X_{\alpha_b}$. This connected component eventually merges with another one for some value $\alpha_d \geq \alpha_b$, which is stored as well and called the death time. One says that the component persists on the corresponding interval $[\alpha_b, \alpha_d]$. Similarly, we save the $[\alpha_b, \alpha_d]$ values of each loop, cavity, etc. that appears in a specific sublevel set $X_{\alpha_b}$ and disappears (gets “filled”) in $X_{\alpha_d}$. This family of intervals is called the barcode, or persistence diagram (PD), of $(\mathcal{X}, f)$, and can be represented as a multiset (i.e., elements are counted with multiplicity) of points supported on the open half-plane $\{ (\alpha_b, \alpha_d) \in \mathbb{R}^2 : \alpha_b < \alpha_d \} \subset \mathbb{R}^2$. The information of connected components, loops, and cavities is represented in PDs of dimension 0, 1, and 2, respectively.

Filtrations for point clouds. Let $X = \{ x_1, \ldots, x_N \}$ be a finite point cloud in $\mathcal{X} = \mathbb{R}^d$, and $\alpha \geq 0$. Let $d_f: X \to \mathbb{R}, v \mapsto \min_{x \in X} \| v - x \|$ denote the distance of $v \in \mathbb{R}^d$ to $X$. In this case, the sublevel set $X_\alpha$ is given by the union of $d$-dimensional closed balls of radius $\alpha$ centered at $x_i$ ($i = 1, \ldots, N$). From this filtration, different types of PDs can be built, called Čech, Rips, and alpha filtrations, which can be considered as being equivalent for the purpose of this work (in particular, RipsNet can be used seamlessly with any of these choices, see (de Surrel et al. 2022)). Due to its computational efficiency in low-dimensional settings, we use the alpha filtration in our numerical experiments.

Metrics between persistence diagrams. The space of PDs can be equipped with a parametrized metric $d_s$, $1 \leq s \leq \infty$ which is rooted in algebraic considerations and whose proper definition is not required in this work. In the particular case $s = \infty$, this metric will be referred to as the bottleneck distance between PDs. Of importance is, that the space of PDs $\mathcal{D}$ equipped with such metrics lacks linear (Hilbert; Euclidean) structure (Carriere & Bauer 2018; Bubenik & Wagner 2020).

The lack of linear structure of the metric space $(\mathcal{D}, d_s)$ prevents a faithful use of PDs in standard machine learning pipelines, for which one typically requires inputs to belong to a finite-dimensional vector space. A natural workaround is thus to seek for a vectorization of PDs (PV), that is a map $\phi: (\mathcal{D}, d_s) \to (\mathbb{R}^d, \| \cdot \|)$ for some dimension $d$. Provided the map $\phi$ satisfies suitable properties (e.g., being Lipschitz, injective, etc.), one can turn a sample of diagrams $\{ D_1, \ldots, D_n \} \subset \mathcal{D}$ into vectors $\{ \phi(D_1), \ldots, \phi(D_n) \} \subset \mathbb{R}^d$ which can subsequently be used in machine learning.

Various vectorization techniques, with success in applications, have been proposed (Carriere et al. 2015; Chazal et al. 2015; Kališnik 2018). In this work, we focus on the persistence image (Adams et al. 2017) —though the approach developed in this work adapts faithfully to any other vectorization as persistence landscapes (Bubenik 2015), as detailed in (de Surrel et al. 2022).
Persistence images (PI). Given a PD $D$, computing its persistence image essentially boils down to putting a Gaussian $g_u(z) := \frac{1}{2\pi\sigma^2} \exp\left(-\frac{\|z-u\|^2}{2\sigma^2}\right)$, with fixed variance $\sigma^2$, on each of its points $u$ and weighing it by a piecewise differentiable function $w : \mathbb{R}^2 \to \mathbb{R}_{\geq 0}$ (typically a function of the distance of $u$ to the diagonal in $\mathbb{R}^2$) and then discretizing the resulting surface on a fixed grid to obtain an image. Formally, one starts by rotating the diagram $D$ via the map $T : \mathbb{R}^2 \to \mathbb{R}^2$, $(b, d) \mapsto (b, d - b)$. The persistence surface of $D$ is defined as $\rho_D(z) := \sum_{u \in T(D)} w(u) g_u(z)$, where $w$ satisfies $w(z, 0) = 0$.

Now, given a compact subset $A \subset \mathbb{R}^2$ partitioned into domains $A = \bigcup_{i \in \mathbb{N}} P_i$—in practice a rectangular grid regularly partitioned in $(n \times n)$ pixels—we set $I(\rho_D)_P := \int_{P} \rho_D d\mathbf{z}$. The vector $(I(\rho_D)_P)_{i=1}^{n}$ is the persistence image of $D$. The transformation $\Pi : X \mapsto Dgm(X) \mapsto I(\rho_{Dgm(X)})$ defines a finite-dimensional vectorization as shown in Figure 1.

### 3 RipsNet

**Motivation and definition.** In the pipeline illustrated in Figure 1, the computation of the PD $Dgm(X)$ from an input point cloud $X$ is the most complex operation involved: it is both computationally expensive and introduces non-differentiability in the pipeline. Moreover, as detailed in (de Surrel et al., 2022) for the specific case of persistence images (PI), the output vectorization can be highly sensitive to perturbations in the input point cloud $X$: moving a single point $p_i \in X$ can arbitrarily change $\Pi(X)$ even when $n$ is large. This instability can be a major limitation when incorporating persistence vectorizations (PVs) of diagrams in practical applications.

To overcome these difficulties, we propose a way to bypass this computation by designing a neural network architecture that we call RipsNet (RN). The goal is to learn a function, denoted by RN as well, able to reproduce persistence vectorizations for a given distribution of input point clouds $X \sim P$ after training on a sample $\{X_i\}_{i=1}^{n}$ with labels being the corresponding vectorizations $\{PV(X_i)\}_{i=1}^{n}$.

As RN takes point clouds $X = \{x_1, \ldots, x_N\} \subset \mathbb{R}^d$ of potentially varying sizes as input, it is natural to expect it to be permutation invariant. An efficient way to enforce this property is to rely on a DeepSets architecture (Zaheer et al., 2017). Namely, it consists of decomposing the network into a succession of two maps $\phi_1 : \mathbb{R}^d \to \mathbb{R}^d$ and $\phi_2 : \mathbb{R}^d \to \mathbb{R}^{d'}$ and a permutation invariant operator $op$—typically the sum, the mean, or the maximum—$\text{RN} : X \mapsto \phi_2(op(\phi_1(x)_{x \in X}))$. For each $x \in X$, the map $\phi_1$ provides a representation $\phi_1(x)$; these point-wise representations are gathered into a single one via the permutation invariant operator $op$, and the map $\phi_2$ is finally applied on it.

In practice, $\phi_1$ and $\phi_2$ are themselves parameterized by neural networks; in this work, we will consider simple feed-forward fully-connected networks (see Section 4 for the architecture hyper-parameters), though more general architectures could be considered. The parameters characterizing $\phi_1$ and $\phi_2$ are tuned during the training phase, where we minimize the $L2$-loss $\sum_{i=1}^{n} ||\text{RN}(X_i) - PV(X_i)||^2$, over a set of training point clouds $\{X_i\}_i$ with corresponding pre-computed vectorizations $\{PV(X_i)\}_i$.

Once properly trained (assuming good generalization properties), when extracting topological information of a point cloud, a important advantage of using RN instead of PV lies in the computational efficiency: while the exact computation of PDs and vectorizations has cubic complexity in the number of simplices in the filtration (which can itself be exponential in the number of points in $X$), running the forward pass of a trained network is significantly faster, as showcased in Section 4. Moreover, RN also satisfies some strong robustness properties, as detailed in the following. This yields a substantial advantage over exact PVs when the data contain some perturbations such as noise, outliers, or adversarial attacks.
Wasserstein stability of RipsNet. We now show that RipsNet satisfies robustness properties. A convenient formalism to demonstrate these properties is to represent a point cloud $X = \{x_1, \ldots, x_N\}$ by a probability measure $m_X := \frac{1}{N} \sum_{i=1}^{N} \delta_{x_i}$, where $\delta_{x_i}$ denotes the Dirac mass located at $x_i \in \mathbb{R}^d$. Let $\mu(f)$ denote $\int f d\mu$ for a map $f$ and a probability measure $\mu$. Such measures can be compared by Wasserstein distances $W_p$, $p \geq 1$, which are defined for any two probability measures $\mu, \nu$ supported on a compact subset $\Omega \subset \mathbb{R}^d$ as $W_p(\mu, \nu) := (\inf_{\pi} \int \|x - y\|^p d\pi(x, y))^{\frac{1}{p}}$, where the infimum is taken over measures $\pi$, supported on $\mathbb{R}^d \times \mathbb{R}^d$, with marginals $\mu$ and $\nu$.

In this section, we set $\text{op}$ as the mean operator: $\text{op}\{y_1, \ldots, y_N\} = \frac{1}{N} \sum_{i=1}^{N} y_i$, and let $\text{RN} = \phi_2 \circ \text{op} \circ \phi_1$, where $\phi_1, \phi_2$ are Lipschitz-continuous maps with Lipschitz constants $C_1, C_2$ respectively.

Let us first investigate the pointwise stability of $\text{RN}$. If $X = \{x_1, \ldots, x_{N-1}, x_N\} \subset \Omega$ and $X' = \{x_1, \ldots, x_{N-1}, x_N'\} \subset \Omega$, then $W_1(m_X, m_{X'}) \leq \frac{1}{N} \|x_N - x_N'\|$. Therefore, moving a single point $x_N$ of $X$ to another location $x_N'$ changes the $W_1$ distance between the two measures by at most $O(1/N)$. More generally, moving a fraction $\lambda \in (0, 1)$ of the points in $X$ affects the Wasserstein distance in $O(\lambda)$. From this, we can derive a first stability result for RipsNet.

**Proposition 3.1.** For any two point clouds $X, Y$, and any $p \geq 1$, one has
\[
\|\text{RN}(X) - \text{RN}(Y)\| \leq C_1 C_2 \cdot W_1(m_X, m_Y) \leq C_1 C_2 \cdot W_p(m_X, m_Y).
\]

In particular, this result implies that moving a small proportion of points $\lambda$ in $\Omega$ in a point cloud $X$ does not affect the output of $\text{RN}$ by much. We refer to it as a “pointwise stability” result in the sense that it describes how $\text{RN}$ is affected by perturbations of a fixed point cloud $X$. Note that in contrast, Rips PDs, as well as their vectorizations, are not robust to such perturbations: moving a single point of $X$, even in the regime $\lambda \rightarrow 0$, may change the resulting PD by a fixed positive amount, preventing a similar result to hold for PVs.

The pointwise stability result of Proposition 3.1 can now be used to obtain a good theoretical understanding on how RipsNet behaves in practical learning settings. For this, we consider the following model: let $P$ be a law on some compact set $\Omega \subset \mathbb{R}^d$, fix $N \in \mathbb{N}$, and let $\text{P}$ denote $P^\otimes N$, that is, $X \sim P$ is a random point cloud $X = \{x_1, \ldots, x_N\}$ where the $x_i$’s are i.i.d. $\sim P$.

In practice, given a training sample $X_1, \ldots, X_n \sim P$, RipsNet is trained to minimize the empirical risk $\hat{\mathcal{R}}_n$ which, hopefully, yields a small theoretical risk $\mathcal{R}$, where
\[
\hat{\mathcal{R}}_n := \frac{1}{n} \sum_{i=1}^{n} \|\text{RN}(X_i) - \text{PV}(X_i)\| \quad \text{and} \quad \mathcal{R} := \int \|\text{RN}(X) - \text{PV}(X)\| d\text{P}(X).
\]

**Remark 3.2.** The question to know whether “$\hat{\mathcal{R}}_n$ small $\Rightarrow \mathcal{R}$ small” is related to the capacity of RipsNet to generalize properly. Providing a theoretical setting where such an implication should hold is out of the scope of this work, but can be checked empirically by looking at the performances of RipsNet on validation sets.

We now consider a standard contamination model ([Huber, 1964]): given a point cloud $X \sim P$, we randomly replace a fraction $\lambda = \frac{N-K}{N} \in (0, 1)$ of its points by corrupted observations distributed with respect to some law $Q$. Let $Y \sim Q^\otimes N-K := Q$ and $F(X, Y)$ denote this corrupted point cloud.

**Lemma 3.3.** Let $C(P, Q) := \mathbb{E}_{P \otimes Q}[\|x - y\|]$. Then, $\mathbb{E}_{P \otimes Q}[W_1(F(X, Y), X)] \leq \lambda C(P, Q)$. In particular, if $P, Q$ are supported on a compact set $\Omega \subset \mathbb{R}^d$ with diameter $\leq L$, the bound is $\lambda L$.

This result is similar to the well-known robustness properties of the Wasserstein distance for contamination models, as detailed in, e.g., [Staerman et al., 2021]. We can now state our main result.

**Proposition 3.4.** One has $\int \|\text{RN}(F(X, Y)) - \text{PV}(X)\| d\text{P}(X) d\text{Q}(Y) \leq \lambda C_1 C_2 \cdot C(P, Q) + \mathcal{R}$. In particular, if $P, Q$ are supported on a compact subset of $\mathbb{R}^d$ with diameter $\leq L$, one has
\[
\int \|\text{RN}(F(X, Y)) - \text{PV}(X)\| d\text{P}(X) d\text{Q}(Y) \leq O(\lambda + \mathcal{R}).
\]

Therefore, if RipsNet achieves a low theoretical test risk ($\mathcal{R}$ small) and only a small proportion $\lambda$ of points is corrupted, RipsNet will produce outputs similar, in expectation, to the persistence vectorizations $\text{PV}(X)$ of the clean point cloud.

2As the $x_i$’s are i.i.d., we may assume without loss of generality that the last $N-K$ points are replaced.
4 Numerical Experiments

In this section, we illustrate the properties of our general architecture RN presented in the previous sections. The approach we use is the following: we first train a RN architecture on a training data set $\text{Tr}_1$, comprised of point clouds $\text{PC}_1$ with their corresponding labels $\text{L}_1$ and persistence vectorizations $\text{PV}_1$ computed via alpha filtrations. Note that this training step does not require the labels $\text{L}_1$ of the point clouds since the targets are the persistence vectorizations $\text{PV}_1$. Then, we use both RN and Gudhi to compute the persistence vectorizations of three data sets: a second training data set $\text{Tr}_2 = (\text{PC}_2, \text{L}_2)$, a test data set $\text{Te} = (\text{PC}, \text{L})$, and a noisy test data set $\tilde{\text{Te}} = (\text{PC}, \text{L})$ (as per our noise model explained in Section 3). All three data sets are comprised of labeled point clouds only.

At this stage, we also measure the computation time of RN and Gudhi for generating outputs.

To obtain quantitative scores, we finally train two machine learning classifiers: one on the labeled RN PDs from $\text{Tr}_2$, and the other on the labeled Gudhi PDs from $\text{Tr}_2$, which we call $\text{Cl}_{\text{RN}}$ and $\text{Cl}_{\text{Gudhi}}$, respectively. The classifiers $\text{Cl}_{\text{RN}}$ and $\text{Cl}_{\text{Gudhi}}$ are then evaluated on the test PDs computed with RN and Gudhi, respectively, on both Te and $\tilde{\text{Te}}$. See Figure 2 for a schematic overview.

We also generate scores using alphaDTM-based filtrations (Anai et al., 2019) computed with the Python package Velour with parameters $m = 5\%$ (respectively $m = 0.75\%$ for the 3D-shape experiments), $p = 2$, in the exact same way as we did for Gudhi. We let Gudhi$^{\text{DTM}}$ and Cl$_{\text{Gudhi}}^{\text{DTM}}$ denote the corresponding model and classifier, respectively. Note that alphaDTM-based filtrations (which are modifications of the alpha filtrations in the presence of noise) usually require manual tuning, which, contrary to RN parameters, cannot be optimized during training. Hence, in our experiments, we manually tuned those parameters. Finally, note that we also added some (non-topological) baselines in each of our experiments to provide a sense of what other methods are capable of in terms of accuracy scores. However, our main purpose is to show that RN can provide a much faster and more noise-robust alternative to Gudhi: the most important comparison is between RN and both Gudhi and Gudhi$^{\text{DTM}}$.

In the following, we apply our experimental setup to two types of data. First, we focus on synthetic data generated by a simple generative model. Next, we consider 3D-shape data from the ModelNet10 data set (Wu et al., 2015). The computations were run on a computing cluster on 4 Xeon SP Gold 2.6GHz CPU cores with 8GB of RAM per core. Accuracy scores (of all methods) are provided in Table 1 and running times (of topological methods) are provided in Table 2.

4.1 Synthetic Data

Data set. Our synthetic data set consists of samplings of unions of circles in the Euclidean plane $\mathbb{R}^2$. These unions are made of either one, two, or three circles, and we use the number of circles as the labels of the point clouds. Each point cloud has $N = 600$ points, and $N - K = 200$ corrupted points, i.e., $\lambda = 1/3$, when noise is added.

We train a RipsNet architecture $\text{RN}_{\text{synth}}$ on a data set $\text{Tr}_1$ of 3300 point clouds, using 3000 point clouds for training and 300 as a validation set. The persistence diagrams $\text{PD}_1$ were computed with
alpha filtrations in dimension 1 with Gudhi, and then vectorized into normalized persistence images of resolution $50 \times 50$, leading to 2500-dimensional vectors. The image bandwidth was estimated as the 0.2-quantile of all pairwise distances between the birth-persistence transforms of the PD points, and the image weight was defined as $10 \cdot \tanh(y - x)$.

Our architecture $\text{RN}_{\text{synth}}$ is structured as follows. The permutation invariant operator is $\text{op} = \text{sum}$, $\phi_1$ is made up of three fully connected layers of 30, 20, and 10 neurons with ReLU activations, $\phi_2$ consists of three fully connected layers of 50, 100, and 200 neurons with ReLU activations, and a last layer with sigmoid activation. We used the mean squared error (MSE) loss with Adam optimizer with $\varepsilon = 5 \cdot 10^{-4}$, and early stopping after 200 epochs with less than $10^{-5}$ improvement.

Finally, we evaluate $\text{RN}_{\text{synth}}$ and Gudhi using default XGBoost classifiers $C^G_{\text{RN}_{\text{synth}}}$ and $C^G_{\text{Gudhi}}$ from Scikit-Learn trained on a data set $\mathcal{T}_2$ of 3000 point clouds and tested on a clean test set and a noisy test set $\mathcal{T}_e$ of 300 point clouds each. We compare it against two DeepSets baselines trained directly on the point clouds: $\text{DS}_1$ resp. $\text{DS}_1$ with fully connected layers of sizes (50, 30, 10, 3) resp. (50, 3). Both architectures have ReLU activations, except for the last layer, $\text{op} = \text{sum}$, default Adam optimizer, cross entropy loss, and early stopping after 200 epochs with less than $10^{-4}$ improvement.

**Results.** We show a few point clouds of $\mathcal{T}_e$ and $\mathcal{T}_e$, as well as their corresponding vectorized PDs and estimated vectorizations with $\text{RN}_{\text{synth}}$, in Figure [3].

As one can see from Figure [3] and the first four rows of Table 1, $\text{RN}_{\text{synth}}$ manages to learn features that are visually similar to the PD vectorizations generated by Gudhi, with comparable accuracy on clean data. However, features generated by $\text{RN}_{\text{synth}}$ are much more robust and thus contain complementary information to the purely topological features; even though $C^G_{\text{Gudhi}}$ and $C^G_{\text{Gudhi}_{\text{DTM}}}$ see their accuracy largely decrease when noise is added, $C^G_{\text{RN}_{\text{synth}}}$ accuracy only decreases slightly. Note that the decrease of accuracy is more moderate for $C^G_{\text{Gudhi}_{\text{DTM}}}$ since Gudhi $\text{DTM}$ is designed to be more robust to outliers. Also, one can see from Table 2 that running times are much more favorable for $\text{RN}_{\text{synth}}$, with an improvement of 2 (resp. 3) orders of magnitude over Gudhi (resp. Gudhi $\text{DTM}$).

### 3D-shape data and RipsNet generalization capacity

**Data set.** We also run experiments on Princeton’s ModelNet10 data set, comprised of 10 classes of 3D meshes. In order to obtain point clouds in $\mathbb{R}^3$, we sample 1024 points on the surfaces of the 3D meshes, which are then centered and normalized to be contained in the unit sphere. We have 2393/598 and 406/229 training and test samples at our disposal for the training of RipsNet, and for the training of neural net classifiers, respectively. The architecture of these classifiers (NN) is very simple, consisting of only two consecutive fully connected layers of 100 and 50 neurons.

For the sake of simplicity, we focus on persistence images of resolution $25 \times 25$ with weight function $(y - x)^2$ only, and consider the combination of PDs of dimension 0 and 1. The vectorization parameters were estimated as in Section 4.1. The final RipsNet architecture, using $\text{op} = \text{mean}$, was found via a 3-fold cross-validation over several models, and again optimized with Adam optimizer.

<table>
<thead>
<tr>
<th>Data</th>
<th>Gudhi (s)</th>
<th>Gudhi $\text{DTM}$ (s)</th>
<th>RN (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{P}_1$</td>
<td>09.5 ± 3.1</td>
<td>173.7 ± 13.3</td>
<td>0.4 ± 0.0</td>
</tr>
<tr>
<td>$\mathcal{P}_3$</td>
<td>118.4 ± 4.7</td>
<td>178.5 ± 8.1</td>
<td>0.2 ± 0.0</td>
</tr>
<tr>
<td>$\lambda = 2%$</td>
<td>117.8 ± 4.5</td>
<td>180.0 ± 9.2</td>
<td>0.2 ± 0.0</td>
</tr>
<tr>
<td>$\lambda = 5%$</td>
<td>117.8 ± 4.5</td>
<td>180.0 ± 9.2</td>
<td>0.2 ± 0.0</td>
</tr>
</tbody>
</table>

Table 2: Running times of topological methods.
To showcase the robustness of RN, we introduce noise fractions \( \lambda \) in \( \{0.02, 0.05, 0.1, 0.25, 0.5\} \). In addition, we compare RN to TopologyNet, a recently introduced model \((\text{Zhou et al., 2021})\) which predicts topological descriptors on 3D point clouds, but relies on building a \( k \)-NN graph on top of each point cloud (we used \( k = 16 \) in this work) and hence is more task-specific and less general than RipsNet in architectural design. TopologyNet architecture and training parameters are detailed in \((\text{Zhou et al., 2021})\). As an independent non-topological baseline, we also compared RipsNet to pointnet \((\text{Qi et al., 2017})\), a standard neural network for 3D point cloud classification.

Results. In addition to the neural net classifiers, we also train XGBoost classifiers, the results of which, as well as additional results of the neural net classifiers and running times, are reported in the last parts of Table 1 and Table 2. Due to class imbalances, the accuracy of the best possible constant classifier is 22.2%. As the sampling of the point clouds, as well as the addition of noise, are random, we repeat this process 10 times in total. Subsequently, we train the classifiers on each of these data sets, without retraining RN, and report the mean and standard deviation. In our experiments, the average running time of TopologyNet (for approximating PDs and their vectorizations) is \( 28.0 \pm 0.2 \) s. Hence, the running times of both Gudhi and TopologyNet are outperformed by RN by three and two orders of magnitude respectively, see Table 2. The accuracy of \( C_{\text{RN}} \) substantially surpasses those of \( C_{\text{Gudhi}} \) and \( C_{\text{Gudhi}, \text{DTM}} \) for all values of \( \lambda \) and remains much more robust for high levels of noise. RipsNet and TopologyNet exhibit similar behaviour: TopologyNet performs slightly better in low noise settings while RipsNet is more robust to noise. However, contrary to RipsNet, TopologyNet does not enjoy any theoretical robustness result and indeed deteriorates significantly for higher noise levels. TopologyNet is designed to handle 3D point clouds and relies on building a \( k \)-NN graph on input point clouds, and thus leverages more information, yielding better accuracy in uncorrupted settings, however yielding an increased estimation time of about a factor 140 over RipsNet. As for pointnet, for \( \lambda \geq 0.1 \), \( C_{\text{RN}} \) surpasses the pointnet baseline, whose accuracy decreases sharply for \( \lambda \geq 0.25 \), at which point \( C_{\text{RN}} \) substantially outperforms it.

Generalization capacity to ModelNet40. A final natural question to address is whether RipsNet presents generalization capacity in the following sense: is it capable of producing meaningful topological descriptors on similar—yet unseen—objects to the one it was trained on? To investigate this, we ran the following experiment: after training RN on ModelNet10, we test its performance—along with the one of TopologyNet \((\text{Zhou et al., 2021})\)—on the ModelNet40 data set; a data set that contains additional types of 3D shapes. Interestingly, both models were capable of producing good vectorizations on these unseen classes: RN reaches a MSE of \( 8 \cdot 10^{-3} \) on this new data set and TopologyNet reaches a MSE of \( 4 \cdot 10^{-3} \), these quantities being compared to the respective training MSE on ModelNet10 of \( 4.5 \cdot 10^{-3} \) and \( 1.9 \cdot 10^{-3} \). See also \((\text{de Surrel et al., 2022})\) for complementary qualitative illustrations of the output produced. While TopologyNet exhibits better quantitative results than RipsNet in this experiment, which is expected as this model takes into account more structure on the input point cloud, this comes at the price of computational efficiency: RN produces a prediction in 3ms on average, while TopologyNet requires about 780ms to do so.

5 Conclusion

Vectorization of topological features is of central importance for their practical use in machine learning applications. However, the computational complexity of the exact computation of persistence diagrams and their sensitivity to outliers and noise limit their applicability. To address these limitations, we propose RipsNet, a DeepSets-like architecture, which learns to estimate persistence vectorizations of point cloud data, and is theoretically proven to be more robust to outliers and noise than exact vectorizations. Moreover, we substantiate our theoretical findings by numerical experiments on a synthetic data set, as well as on 3D-shape data (and time series data), which exhibit significant improvements in robustness, accuracy scores, and running times.

Several questions remain open for future works. First, RN is currently limited by its data-driven nature; it requires training beforehand and the architecture should be adapted depending on data sets or tasks. In particular, generalization capacity of RN may improve by incorporating constraints on the network making it scaling-, translation- and rotation-invariant (e.g. with data augmentation), which we currently handle by manually normalizing the data and PDs. Second, RN may also be combined with PersLay \((\text{Carrière et al., 2020})\) and parametric families of filtrations, in order to avoid choosing a filtration and an embedding before learning.
REFERENCES


