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

Anonymous authors
Paper under double-blind review

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 an open-source library.

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 Cámara (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), or in the study of the behavior of deep neural networks Guss & Salakhutdinov (2018); Naitzat et al. (2020); Birdal et al. (2021).

A drawback of Rips PDs computed on large point clouds is that they are computationally expensive. Furthermore, even though these topological descriptors enjoy stability properties with respect to the input point cloud in the Hausdorff metric Chazal et al. (2014), 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 hinder 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* Bubenik (2015); Adams et al. (2017); Carrière et al. (2015); Chazal et al. (2015);

Kališnik (2018), turning PDs into elements of Euclidean space $\mathbb{R}^d$, and *kernels* Carrière et al. (2017); Kusano et al. (2016); Le & Yamada (2018); Reininghaus et al. (2015), 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. We refer to Appendix A for a description of related work.

Contributions. We introduce RipsNet, a DeepSets-like architecture capable of learning finite-dimensional embeddings of Rips PDs built on top of point clouds. We experimentally showcase how it 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. RipsNet also enjoys interesting stability properties, which are presented in Appendix D for the sake of concision.

2 Background

In TDA, persistence diagrams (PDs) are considered to be highly important descriptors of topological features (see Appendix B for further details on PDs). The space of persistence diagrams can be equipped with a parametrized metric d_s, $1 \leqslant s \leqslant \infty$ which is rooted in algebraic considerations and whose proper definition is not required in this work. Of importance is, that the space of persistence diagrams $\mathcal{D}$ 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 persistence diagrams in standard machine learning pipelines, as such techniques typically require inputs belonging to a finite-dimensional vector space. A natural workaround is thus to seek for a *vectorization* of persistence diagrams (PV), that is a map $\phi : (\mathcal{D}, d_s) \to (\mathbb{R}^d, \|\cdot\|)$ for some dimension d. Provided the map ϕ 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 a collection of vectors $\{\phi(D_1), \ldots, \phi(D_n)\} \subset \mathbb{R}^d$ which can subsequently be used to perform any machine learning task.

Various vectorization techniques, with success in applications, have been proposed Carrière et al. (2015); Chazal et al. (2015); Kališnik (2018). In this work, we focus, for the sake of concision, on two of them: the *persistence image* (PI) Adams et al. (2017) and the *persistence landscape* (PL) Bubenik (2015)—though the approach developed in this work adapts faithfully to any other vectorization. We refer to Appendix C for details on PI and PLs.

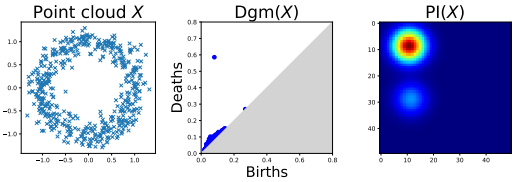

Figure 1: Pipeline to extract PIs from point clouds. Details in Appendix C

3 RipsNet

In the pipeline illustrated in Figure 1, the computation of the persistence diagram $\mathrm{Dgm}(X)$ 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 Appendix D.2 for the specific case of persistence images, 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 $\mathrm{PI}(X)$ 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 to bypass this computation by designing a neural network architecture which 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 $X \sim \mathbf{P}$ after being trained on a sample $\{X_i\}_{i=1}^n$ with labels being the corresponding vectorizations $\{\mathrm{PV}(X_i)\}_{i=1}^n$.

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 $\phi_1 \colon \mathbb{R}^d \to \mathbb{R}^{d'}$ and $\phi_2 \colon \mathbb{R}^{d'} \to \mathbb{R}^{d''}$ and a permutation invariant operator **op**—typically the sum, the mean, or the maximum

$$\mathbf{RN} \colon X \mapsto \phi_2\left(\mathbf{op}(\{\phi_1(x)\}_{x \in X})\right).$$

For each $x \in X$, the map ϕ_1 provides a representation $\phi_1(x)$; these point-wise representations are gathered via the permutation invariant operator **op**, and the map ϕ_2 is subsequently applied to compute the network output. In practice, ϕ_1 and ϕ_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 ϕ_1 and ϕ_2 are tuned during the training phase, where we minimize the $L2$-loss

$$\sum_{i=1}^{n} \|\mathbf{RN}(X_i) - \mathrm{PV}(X_i)\|^2, \tag{1}$$

over a set of training point clouds $\{X_i\}_i$ with corresponding pre-computed vectorizations $\{\mathrm{PV}(X_i)\}_i$.

Once trained properly (assuming good generalization properties), when extracting topological information of a point cloud, an important advantage of using **RN** instead of PV lies in the computational efficiency: while the exact computation of persistence diagrams and vectorizations rely on expensive combinatorial computations, running the forward pass of a trained network is significantly faster, as showcased in Section 4. As detailed in Appendix D and Figure 3, **RN** also satisfies some strong robustness properties, yielding a substantial advantage over exact PVs when the data contain some perturbations such as noise, outliers, or adversarial attacks.

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 an **RN** architecture on a training data set Tr_1, comprised of point clouds PC_1 with their corresponding labels L_1 and persistence vectorizations PV_1. Note that this training step does not require the labels L_1 of the point clouds since the targets are the persistence vectorizations PV_1. Then, we use both **RN** and `Gudhi` to compute the persistence vectorizations of three data sets: a second training data set $\mathrm{Tr}_2 = (\mathrm{PC}_2, \mathrm{L}_2)$, a test data set $\mathrm{Te} = (\mathrm{PC}, \mathrm{L})$, and a noisy test data set $\widetilde{\mathrm{Te}} = (\widetilde{\mathrm{PC}}, \widetilde{\mathrm{L}})$ (as per our noise model explained in Section D.1.2). 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 these persistence vectorizations.

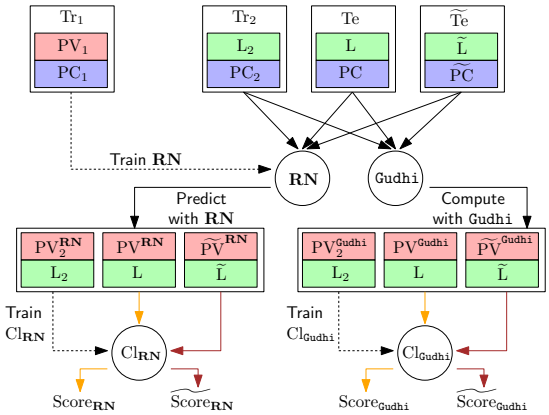

Figure 2: Scheme of our general experimental setup.

In order to obtain quantitative scores, we finally train two machine learning classifiers: one on the labeled **RN** persistence diagrams from Tr_2, and the other on the labeled `Gudhi` persistence diagrams from Tr_2, which we call $\mathrm{Cl}_{\mathbf{RN}}$ and $\mathrm{Cl}_{\texttt{Gudhi}}$, respectively. The classifiers $\mathrm{Cl}_{\mathbf{RN}}$ and $\mathrm{Cl}_{\texttt{Gudhi}}$ are then evaluated on the test persistence diagrams computed with **RN** and `Gudhi`, respectively, on both Te and $\widetilde{\mathrm{Te}}$. See Figure 2 for a schematic overview.

Finally, we also generate scores using AlphaDTM-based filtrations Anai et al. (2019) computed with `Gudhi` and the Python package `Velour` with parameters $m = 0.75\%$, $p = 2$, in the exact same

λ (%)	$\mathrm{Cl}^{\mathrm{NN}}_{\mathtt{Gudhi}}$	$\mathrm{Cl}^{\mathrm{NN}}_{\mathtt{Gudhi}^{\mathrm{DTM}}}$	$\mathrm{Cl}^{\mathrm{NN}}_{\mathbf{RN}}$	`pointnet`
0	30.4 ± 4.0	30.9 ± 2.0	53.9 ± 2.4	$\mathbf{81.6 \pm 1.1}$
2	30.3 ± 3.2	31.0 ± 2.7	53.2 ± 2.5	$\mathbf{74.5 \pm 1.6}$
5	29.9 ± 4.0	31.0 ± 2.7	55.1 ± 3.3	$\mathbf{63.4 \pm 1.6}$
10	25.2 ± 3.2	29.5 ± 3.1	$\mathbf{51.0 \pm 2.1}$	50.6 ± 1.5
15	22.9 ± 4.6	25.7 ± 3.1	$\mathbf{46.9 \pm 3.0}$	44.9 ± 1.7
25	14.4 ± 4.0	18.1 ± 2.6	$\mathbf{42.6 \pm 2.5}$	11.0 ± 0.2
50	14.0 ± 3.4	13.1 ± 1.9	$\mathbf{31.6 \pm 3.3}$	10.9 ± 0.0

Table 1: Accuracy scores of classifiers trained on `Gudhi`, $\mathtt{Gudhi}^{\mathrm{DTM}}$ and **RN** PVs on `ModelNet10` data. The highest accuracy of the three topology-based classifiers (middle) is highlighted in red, and the highest accuracy over all models in bold font.

way as we did for `Gudhi`. We let $\mathtt{Gudhi}^{\mathrm{DTM}}$ and $\mathrm{Cl}_{\mathtt{Gudhi}^{\mathrm{DTM}}}$ denote the corresponding model and classifier, respectively. Note that AlphaDTM-based filtrations usually require manual tuning, which, contrary to **RN** parameters, cannot be optimized during training. In our experiments, we manually designed those parameters so that they provide reasonably-looking persistence vectorizations. 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. However, our main purpose is to show that **RN** can provide a much faster and more noise-robust alternative to `Gudhi`, and hence the comparison we are most interested in is between **RN** and both `Gudhi` and $\mathtt{Gudhi}^{\mathrm{DTM}}$.

In the following, we apply our experimental setup to 3D-shape data from the `ModelNet10` data set Wu et al. (2015). For experimental results on a synthetic data set, as well as on time series data obtained from the `UCR` archive Dau et al. (2018), see Appendix E. The computations were run on a computing cluster on 4 Xeon SP Gold 2.6GHz CPU cores with 8GB of RAM per core.

4.1 3D-SHAPE DATA

Data set. We ran experiments on Princeton's `ModelNet10` data set, comprised of 3D-shape data of objects in 10 classes. In order to obtain point clouds in $\mathbb{R}^3$, we sample 1024 points on the surfaces of the 3D objects. They are subsequently 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 and an output layer. 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 Tables 6, 7 and Table 8 in Appendix F.

For the sake of simplicity, we focus on persistence images of resolution 25×25 with weight function $(y - x)^2$ only, and consider the combination of persistence diagrams of dimension 0 and 1. The vectorization parameters were estimated as in Section E.1 (due to computational cost, only on a random subset of all PDs). The final RipsNet architecture, using $\mathbf{op} =$ mean, was found via a 3-fold cross-validation over several models, and again optimized with Adam optimizer. As a baseline, we employ the `pointnet` model Qi et al. (2017). To showcase the robustness of **RN**, we introduce noise fractions λ in $\{0.02, 0.05, 0.1, 0.25, 0.5\}$.

Results. The accuracies of the NN classifier are compiled in Table 1 and some running times can be found in Table 3. 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 RipsNet, and report the mean and standard deviation. The vectorization running time of `Gudhi` is clearly outperformed by **RN** by three orders of magnitude (see Table 8 in Appendix E). The accuracy of $\mathrm{Cl}^{\mathrm{NN}}_{\mathbf{RN}}$ substantially surpasses those of $\mathrm{Cl}^{\mathrm{NN}}_{\mathtt{Gudhi}}$ and $\mathrm{Cl}^{\mathrm{NN}}_{\mathtt{Gudhi}^{\mathrm{DTM}}}$ for all values of λ and remains much more robust for high levels of noise. For $\lambda \geq 0.1$, $\mathrm{Cl}^{\mathrm{NN}}_{\mathbf{RN}}$ surpasses the `pointnet` baseline, whose accuracy decreases sharply for $\lambda \geq 0.25$, at which point $\mathrm{Cl}^{\mathrm{NN}}_{\mathbf{RN}}$ substantially outperforms `pointnet`.

REFERENCES

Henry Adams, Tegan Emerson, Michael Kirby, Rachel Neville, Chris Peterson, Patrick Shipman, Sofya Chepushtanova, Eric Hanson, Francis Motta, and Lori Ziegelmeier. Persistence images: a stable vector representation of persistent homology. *Journal of Machine Learning Research*, 18(8), 2017.

Hirokazu Anai, Frédéric Chazal, Marc Glisse, Yuichi Ike, Hiroya Inakoshi, Raphaël Tinarrage, and Yuhei Umeda. DTM-Based Filtrations. In *35th International Symposium on Computational Geometry (SoCG 2019)*, volume 129, pp. 58:1–58:15. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019.

Tolga Birdal, Aaron Lou, Leonidas J Guibas, and Umut Simsekli. Intrinsic dimension, persistent homology and generalization in neural networks. *Advances in Neural Information Processing Systems*, 34, 2021.

Peter Bubenik. Statistical topological data analysis using persistence landscapes. *Journal of Machine Learning Research*, 16(77):77–102, 2015.

Peter Bubenik and Alexander Wagner. Embeddings of persistence diagrams into hilbert spaces. *Journal of Applied and Computational Topology*, 4(3):339–351, 2020.

Mickaël Buchet, Yasuaki Hiraoka, and Ippei Obayashi. Persistent homology and materials informatics. In *Nanoinformatics*, pp. 75–95. 2018.

Pablo Cámara. Topological methods for genomics: present and future directions. *Current Opinion in Systems Biology*, 1:95–101, feb 2017.

Mathieu Carriere and Ulrich Bauer. On the metric distortion of embedding persistence diagrams into separable hilbert spaces. *arXiv preprint arXiv:1806.06924*, 2018.

Mathieu Carrière, Steve Oudot, and Maks Ovsjanikov. Stable topological signatures for points on 3d shapes. In *Computer Graphics Forum*, volume 34, pp. 1–12. Wiley Online Library, 2015.

Mathieu Carrière, Marco Cuturi, and Steve Oudot. Sliced Wasserstein kernel for persistence diagrams. In *International Conference on Machine Learning*, volume 70, pp. 664–673, jul 2017.

Frédéric Chazal, David Cohen-Steiner, Leonidas J Guibas, Facundo Mémoli, and Steve Y Oudot. Gromov-hausdorff stable signatures for shapes using persistence. In *Computer Graphics Forum*, volume 28, pp. 1393–1403. Wiley Online Library, 2009.

Frédéric Chazal, Vin de Silva, and Steve Oudot. Persistence stability for geometric complexes. *Geometriae Dedicata*, 173(1):193–214, 2014.

Frédéric Chazal, Brittany Terese Fasy, Fabrizio Lecci, Alessandro Rinaldo, and Larry Wasserman. Stochastic convergence of persistence landscapes and silhouettes. *Journal of Computational Geometry*, 6(2):140–161, 2015.

Taco Cohen. Equivariant convolutional networks. 2021.

David Cohen-Steiner, Herbert Edelsbrunner, and John Harer. Extending persistence using Poincaré and Lefschetz duality. *Foundations of Computational Mathematics*, 9(1):79–103, feb 2009.

Hoang Anh Dau, Eamonn Keogh, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, Yanping, Bing Hu, Nurjahan Begum, Anthony Bagnall, Abdullah Mueen, Gustavo Batista, and Hexagon-ML. The ucr time series classification archive, 2018. `https://www.cs.ucr.edu/~eamonn/time_series_data_2018/`.

Herbert Edelsbrunner and John Harer. *Computational topology: an introduction*. American Mathematical Society, 2010.

Jennifer Gamble and Giseon Heo. Exploring uses of persistent homology for statistical analysis of landmark-based shape data. *Journal of Multivariate Analysis*, 101(9):2184–2199, 2010.

William H Guss and Ruslan Salakhutdinov. On characterizing the capacity of neural networks using algebraic topology. *arXiv preprint arXiv:1802.04443*, 2018.

Sara Kališnik. Tropical coordinates on the space of persistence barcodes. *Foundations of Computational Mathematics*, pp. 1–29, jan 2018.

Sara Kališnik. Tropical coordinates on the space of persistence barcodes. *Foundations of Computational Mathematics*, 19(1):101–129, 2019.

Genki Kusano, Yasuaki Hiraoka, and Kenji Fukumizu. Persistence weighted Gaussian kernel for topological data analysis. In *International Conference on Machine Learning*, volume 48, pp. 2004–2013, jun 2016.

Tam Le and Makoto Yamada. Persistence Fisher kernel: a Riemannian manifold kernel for persistence diagrams. In *Advances in Neural Information Processing Systems*, pp. 10027–10038, 2018.

Chunyuan Li, Maks Ovsjanikov, and Frédéric Chazal. Persistence-based structural recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2003–2010, jun 2014.

Guido Montúfar, Nina Otter, and Yuguang Wang. Can neural networks learn persistent homology features? *arXiv preprint arXiv:2011.14688*, 2020.

Gregory Naitzat, Andrey Zhitnikov, and Lek-Heng Lim. Topology of deep neural networks. *J. Mach. Learn. Res.*, 21(184):1–40, 2020.

Steve Oudot. *Persistence theory: from quiver representations to data analysis*. American Mathematical Society, 2015.

Jose Perea and John Harer. Sliding windows and persistence: an application of topological methods to signal analysis. *Foundations of Computational Mathematics*, 15(3):799–838, jun 2015.

Cássio MM Pereira and Rodrigo F de Mello. Persistent homology for time series and spatial data clustering. *Expert Systems with Applications*, 42(15-16):6026–6038, 2015.

Charles R. Qi, Hao Su, Mo Kaichun, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 77–85, 2017. doi: 10.1109/CVPR.2017.16.

Jan Reininghaus, Stefan Huber, Ulrich Bauer, and Roland Kwitt. A stable multi-scale kernel for topological machine learning. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

Anirudh Som, Hongjun Choi, Karthikeyan Natesan Ramamurthy, Matthew P Buman, and Pavan Turaga. Pi-net: A deep learning approach to extract topological persistence images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pp. 834–835, 2020.

Yuhei Umeda. Time series classification via topological data analysis. *Information and Media Technologies*, 12:228–239, 2017.

Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. pp. 1912–1920, 06 2015. doi: 10.1109/CVPR.2015.7298801.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan Salakhutdinov, and Alexander Smola. Deep sets. In *Advances in Neural Information Processing Systems*, pp. 3391–3401, 2017.

Chi Zhou, Zhetong Dong, and Hongwei Lin. Learning persistent homology of 3d point clouds. *Computers & Graphics*, 2021.

A Related Work

DeepSets. Our RipsNet architecture is directly based on *DeepSets* Zaheer et al. (2017), a particular case of equivariant neural network Cohen (2021) designed to handle point clouds as inputs. Namely, DeepSets essentially consist of processing a point cloud $X = \{x_1, \ldots, x_n\} \subset \mathbb{R}^d$ via

$$X \mapsto \phi_2\left(\mathbf{op}(\{\phi_1(x_i)\}_{i=1}^n)\right), \tag{2}$$

where $\mathbf{op}$ 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. equation 2 makes the output of DeepSets architectures invariant to permutations, a property of Rips PDs that we want to reproduce in RipsNet.

Learning to estimate PDs. There exist a few works attempting to compute or estimate (vectorizations of) PDs through the use of neural networks. In Som et al. (2020), the authors propose a convolutional neural network (CNN) architecture to estimate persistence images (see Section 2) computed on 2D-images. Similarly, in Montúfar et al. (2020), the authors provide an experimental overview of specific PD features (such as, e.g., their tropical coordinates Kališnik (2019)) 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). Finally, the recent work Zhou et al. (2021) 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 a DeepSet architecture 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). 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, while the other works are mostly experimental.

B Fundamentals of Topological Data Analysis

In this section, we briefly explain some fundamentals of topological data analysis. We refer the interested reader to Cohen-Steiner et al. (2009); Edelsbrunner & Harer (2010); Oudot (2015) for a thorough treatment.

B.1 Simplicial complexes and homology groups

Let us begin by introducing the concepts of simplicial complexes and homology groups.

Definition B.1. Let V be a finite set. A subset K of the power set $P(V)$ is said to be a (finite) *simplicial complex* with vertex set V if it satisfies the following conditions.

(1) $\emptyset \notin K$;

(2) for any $v \in V$, $\{v\} \in K$;

(3) if $\sigma \in K$ and $\emptyset \neq \tau \subset \sigma$, then $\tau \in K$.

An element σ of K with the cardinality $\#\sigma = k + 1$ is called a k-simplex.

Now we introduce homology to extract topology information of simplical complexes. For $\sigma = \{v_0, \ldots, v_k\}$, we consider orderings with respect to its vertices. Two orderings of σ are said to be *equivalent* if one ordering can be obtained from the other by an even permutation. In this way, orderings consist of two equivalence classes, each of which is called an *orientation* of σ. A simplex equipped with an orientation is said to be oriented, and a simplicial complex whose simplices are all oriented is called an oriented simplicial complex. The equivalent class of the ordering $(v_{i_0}, \ldots, v_{i_k})$ is denoted by $\langle v_{i_0}, \ldots, v_{i_k} \rangle$. We use the convention that

$$\langle v_{s(0)}, \ldots, v_{s(k)} \rangle = \mathrm{sgn}(s) \langle v_0, \ldots, v_k \rangle$$

for any permutation s of $\{0, \ldots, k\}$, where $\mathrm{sgn}(s)$ denotes the signature of s. For an oriented simplicial complex K, let $C_i(K)$ be the free abelian group consists of equivalence classes of oriented

i-simplices of K, which is called the i-th *chain group* of K. We now introduce a homomorphism, which is called the *boundary operator*.

Definition B.2. Let K be an oriented simplicial complex and i be a positive integer. For a i-simplex $\sigma \in C_i(K)$, one defines the boundary operator $\partial_i \colon C_i(K) \to C_{i-1}(K)$ by

$$\partial_i(\sigma) = \sum_{j=0}^{k} (-1)^j \langle v_0 \dots v_{j-1} v_{j+1} \dots v_n \rangle.$$

Then one linearly extends the operator for the elements of $C_i(K)$. One also sets $\partial_0 = 0$.

Then we can see that $\partial_i \circ \partial_{i+1} = 0$ for any non-negative integer. This implies $\mathrm{Im}(\partial_{i+1}) \subset \mathrm{Ker}(\partial_i)$.

Definition B.3. For an oriented simplicial complex K, one defines

$$H_i(K) \coloneqq \mathrm{Ker}(\partial_i)/\,\mathrm{Im}(\partial_{i+1})$$

and calls it the i-th *homology group* of K.

We note that a simplicial complex defined in Definition B.1 is sometimes called an *abstract simplicial complex*. As explained in Definition B.1, a simplicial complex consists of a finite set and its power set. We remark that the finite set V does not need to be a subset of Euclidean space. On the other hand, we can interpret the simplicial complex from a geometric perspective. Let K be a simplicial complex and V the vertex set of K. Set $N \coloneqq \#V$ and consider the N-dimensional vector space $\mathbb{R}^N$ with standard basis $e_i \coloneqq (\underbrace{0, \dots, 0}_{i-1}, 1, 0, \dots, 0)$. With each simplex $\sigma = \{v_{i_0}, \dots, v_{i_k}\}$ of K, we associate a k-simplex

$$|\sigma| \coloneqq \{\lambda_{i_0} e_{i_0} + \dots + \lambda_{i_k} e_{i_k} \mid \lambda_{i_0} + \dots + \lambda_{i_k} = 1, \lambda_i \geq 0\}.$$

We define the *geometric realization* of K by $|K| \coloneqq \bigcup_{\sigma \in K} |\sigma| \subset \mathbb{R}^N$ with the subspace topology. A fundamental theorem is that the simplicial homology K is isomorphic to the singular homology of its geometric realization $|K|$: $H_n(K) \simeq H_n(|K|)$ for any $n \in \mathbb{Z}$. Hence, for a topological space $\mathcal{X}$, if we find a simplical complex K such that its geometric realization $|K|$ is homotopy equivalent to $\mathcal{X}$, we can compute the singular homology $H_n(\mathcal{X})$ by the combinatorial object K. Below, we will construct such a complex for a finite union of closed balls.

B.2 Filtrations and persistent homology

We introduce the notion of a filtration of a simplicial complex to consider the evolution of the topology of the simplicial complex.

Definition B.4. Let K be a simplicial complex. A family of subcomplexes $(K_\alpha)_{\alpha \in \mathbb{R}}$ of K is said to be a *filtration* of K if it satisfies

(1) $K_\alpha \subset K_{\alpha'}$ for $\alpha \leq \alpha'$;

(2) $\bigcup_{\alpha \in \mathbb{R}} K_\alpha = K$.

Čech filtration. Let X be a finite point set in $\mathcal{X} = \mathbb{R}^d$ and $\alpha \in \mathbb{R}$. To X, one can associate a function $d_X \colon \mathbb{R}^d \to \mathbb{R}, v \mapsto d(v, X) \coloneqq \min_{x \in X} \|v - x\|$. The sublevel filtration induced by d_X is frequently used to investigate the topology of the point set X. Indeed, for any $\alpha \geq 0$ the sublevel set $\mathcal{X}_\alpha = \{x \in \mathbb{R}^d : d_X(x) \leq \alpha\}$ is equal to the union of d-dimensional closed balls of radius α centered at points in X: $\bigcup_{x \in X} \overline{B}(x; \alpha)$. One can compute the homology of the sublevel set in the following combinatorial way, by constructing a simplicial complex whose geometric realization is homotopy equivalent to the sublevel set. For $\alpha \geq 0$, we define a simplicial complex $\mathcal{C}(X; \alpha)$ by

$$\{x_0, \dots, x_k\} \in \mathcal{C}(X; \alpha) :\Longleftrightarrow \bigcap_{i=0}^{k} \overline{B}(x_i; \alpha) \neq \emptyset.$$

In other words, $\mathcal{C}(X; \alpha)$ is the nerve of the family of closed sets $\{\overline{B}(x; \alpha)\}_{x \in X}$. Since each $\overline{B}(x; \alpha)$ is a convex closed set of $\mathbb{R}^d$, the geometric realization of $\mathcal{C}(X; \alpha)$ is homotopy equivalent to

$\bigcup_{x\in X}\overline{B}(x;\alpha)$ by the nerve theorem. The family $(\mathcal{C}(X;\alpha))_\alpha$ forms a filtration in the sense of Definition B.4 for $\alpha \geq 0$. If α is negative, we regard $\mathcal{C}(X;\alpha)$ as the empty set. We call this filtration the *Čech filtration*.

In the above construction of filtrations, the radii of balls increase uniformly. We can give filtrations in another way, that is, we make radii increase non-uniformly. Such a filtration is called a *weighted filtration*. Let $X \subset \mathbb{R}^d$ be a finite point set, $f\colon \mathbb{R}^d \to \mathbb{R}_{\geq 0}$ a continuous function, and $p \in [1,\infty]$. For $p < \infty$, we define a function $r_f\colon X \times \mathbb{R}_{\geq 0} \to \mathbb{R}\cup\{-\infty\}$ by

$$r_f(x,t) = \begin{cases} -\infty & \text{if } t < f(x), \\ (t^p - f(x)^p)^{\frac{1}{p}} & \text{otherwise.} \end{cases}$$

When $p = \infty$, we also define a function $r_f\colon X \times \mathbb{R}_{\geq 0} \to \mathbb{R}\cup\{-\infty\}$ by

$$r_f(x,t) = \begin{cases} -\infty & \text{if } t < f(x), \\ t & \text{otherwise.} \end{cases}$$

We replace the radius of each closed ball $\overline{B}(x;r)$ by $r_f(x,t)$. By modifying the definition of $\mathcal{C}(X;\alpha)$, we define a simplicial complex $\mathcal{C}_f(X;t)$ by

$$\{x_0,\dots,x_k\} \in \mathcal{C}_f(X;t) :\Longleftrightarrow \bigcap_{i=0}^{k}\overline{B}(x_i;r_f(x_i,t)) \neq \emptyset.$$

Then we have a filtration $\{\mathcal{C}_f(X;t)\}_t$, which is called the *weighted Čech filtration*.

Rips filtration. The Čech complex $\mathcal{C}(X;\alpha)$ exactly computes the homology of the union of closed balls $\bigcup_{x\in X}\overline{B}(x;\alpha)$, but it is computationally expensive in practice. Now we introduce another simplicial complex that is less expensive than the Čech complex.

Let X be a finite point set in $\mathbb{R}^d$. For any $\alpha \geq 0$, one can define a simplicial complex $\mathcal{R}(X;\alpha)$ whose vertex set is X by

$$\begin{aligned}\{x_0,\dots,x_k\} \in \mathcal{R}(X;\alpha) &:\Longleftrightarrow \overline{B}(x_i;\alpha)\cap\overline{B}(x_j;\alpha) \neq \emptyset \text{ for any } i,j \in \{0,\dots,k\} \\ &\Longleftrightarrow \|x_i - x_k\| \leq 2\alpha \text{ for any } i,j \in \{0,\dots,k\}.\end{aligned}$$

Otherwise, we regard $\mathcal{R}(X;\alpha)$ as the empty set. The family $(\mathcal{R}(X;\alpha))_\alpha$ forms a filtration, which we call the *Rips filtration*.

Remark that we can also construct the *weighted Rips filtration* similarly to the weighted Čech filtration.

Alpha filtration. Let X be a finite subset of $\mathbb{R}^d$ and assume that X is in a general position. For $x \in X$, set

$$V_x := \{y \in \mathbb{R}^d \mid d(x,y) \leq d(x',y), \text{for any } x' \in X\setminus\{x\}\},$$

which we call the Voronoi cell for x. With this notation, we set

$$W(x;\alpha) := B(x;\alpha)\cap V_x, \qquad F(X;\alpha) := \{W(x;\alpha)\}_{x\in X}$$

and define $\mathrm{Alpha}(X;\alpha)$ to be the nerve of $F(X;\alpha)$. Then by the nerve theorem, we find that the geometric realization of $\mathrm{Alpha}(X;\alpha)$ is homotopy equivalent to $\bigcup_{x\in X}\overline{B}(x;\alpha)$. The family $(\mathrm{Alpha}(X;\alpha))_\alpha$ forms a filtration and is called the alpha filtration. Similarly, one can construct a weighted version of the alpha filtration, called the *weighted alpha filtration*.

DTM-filtrations. To get robustness to noise and outliers, Anai et al. (2019) introduces DTM-based filtrations. Let μ be a probability measure on $\mathbb{R}^d$ and m a parameter in $[0,1)$. We define a function $\delta_{\mu,m}\colon \mathbb{R}^d \to \mathbb{R}$ by $\delta_{\mu,m}(x) = \inf\{r \geq 0, \mu(\overline{B}(x;r)) > m\}$.

Definition B.5. The distance-to-measure function (DTM for short) μ is the function $d_{\mu,m}\colon \mathbb{R}^d \to \mathbb{R}$ defined by

$$d_{\mu,m}(x) = \sqrt{\frac{1}{m}\int_0^m \delta_{\mu,t}^2(x)dt}.$$

Let X be a finite point cloud with n points and set μ_X to be a empirical measure associated with X: $\mu_X = \frac{1}{n}\sum_{x\in X}\delta_x$, where δ_x is the Dirac measure located at x. For fixed $p \in [0, +\infty]$ and $m \in [0, 1)$, the family of simplicial complexes $\mathcal{C}_{d_{\mu_X,m}}(X; t)$ defines a filtration, which we call the *DTM-based filtration*. By replacing the Čech filtration with the alpha filtration, we can also define the AlphaDTM-based filtration. These filtrations are shown to be robust to noise and outliers.

Persistent homology. Given a filtration $(K_\alpha)_{\alpha\in\mathbb{R}}$ of an n-dimensional simplicial complex K, we have inclusion maps $\iota_\alpha^{\alpha'} : K_\alpha \hookrightarrow K_{\alpha'}$ for any $\alpha \leq \alpha'$. Such inclusion maps induce homomorphisms $(\iota_\alpha^{\alpha'})_* : H_i(K_\alpha) \to H_i(K_{\alpha'})$. Then we have a family of homomorphisms $\cdots \to H_i(K_\alpha) \to H_i(K_{\alpha'}) \to \cdots$. The resulting family is called i-th *persistence module*, and is known to decompose into simpler interval modules, that can be represented as a persistence diagram.

Persistence diagrams.

Let $\mathcal{X}$ be a topological space, and $f : \mathcal{X} \to \mathbb{R}$ a real-valued continuous function. The *α-sublevel set* of $(\mathcal{X}, f)$ is defined as $\mathcal{X}_\alpha = \{x \in \mathcal{X} : f(x) \leq \alpha\}$. Increasing α 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 $\mathcal{X}$. 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 α_b, called the *birth time*, for which a new connected component appears in X_{α_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_{α_b} and disappears (gets "filled") in X_{α_d}. This family of intervals is called the barcode, or *persistence diagram*, 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 persistence diagrams of dimension 0, 1, and 2, respectively.

C Relevant vectorization methods of persistence diagrams

Persistence images (PI). Given a persistence diagram 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 σ^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 $\{(t, t)\} \subset \mathbb{R}^2$) 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 $w(x, 0) = 0$. Now, given a compact subset $A \subset \mathbb{R}^2$ partitioned into domains $A = \bigsqcup_{i=1}^k P_i$—in practice a rectangular grid regularly partitioned in $(n \times n)$ pixels—we set $I(\rho_D)_P := \int_P \rho_D dz$. The vector $(I(\rho_D)_{P_i})_{i=1}^n$ is the persistence image of D. The transformation $\mathrm{PI}: X \mapsto \mathrm{Dgm}(X) \mapsto I(\rho_{\mathrm{Dgm}(X)})$ defines a finite-dimensional vectorization as illustrated in Figure 1.

Persistence landscapes (PL). Bubenik (2015) Given a persistence diagram $D = \{(b_i, d_i)\}_{i=1}^n$, we define a function $\lambda_k(D) : \mathbb{R} \to \mathbb{R}_{\geq 0}$ by

$$\lambda_k(D)(t) = k\text{-}\max_i \min\{t - b_i, d_i - t\}_+,$$

for each $k \in \mathbb{Z}_{\geq 1}$, where k-max denotes the k-th largest value in the set or 0 if the set contains less than k points, and $a_+ = \max\{0, a\}$ for a real number a. The sequence of functions $(\lambda_k(D))_{k=1}^\infty$ is called the *persistence landscape* of the persistence diagram D. In practice, these landscape functions are evaluated on a 1-D grid, and the corresponding values are concatenated into a vector.

D STABILITY RESULTS FOR RIPSNET

D.1 WASSERSTEIN STABILITY OF RIPSNET

In this subsection, we show that RipsNet satisfies robustness properties. A convenient formalism to demonstrate these properties is to represent a point cloud $X = \{x_1, \dots, x_N\}$ by a probability measure $m_X := \frac{1}{N}\sum_{i=1}^{N} \delta_{x_i}$, where δ_{x_i} denotes the Dirac mass located at $x_i \in \mathbb{R}^d$. Let $\mu(f)$ denote $\int f \mathrm{d}\mu$ for a map f and a probability measure μ. Such measures can be compared by Wasserstein distances $W_p,\ p \geq 1$, which are defined for any two probability measures μ, ν supported on a compact subset $\Omega \subset \mathbb{R}^d$ as

$$W_p(\mu, \nu) := \left(\inf_{\pi} \iint \|x - y\|^p \mathrm{d}\pi(x, y) \right)^{\frac{1}{p}},$$

where the infimum is taken over measures π, supported on $\mathbb{R}^d \times \mathbb{R}^d$, with marginals μ and ν. We also mention the so-called Kantorovich–Rubinstein duality formula that occurs when $p = 1$:

$$W_1(\mu, \nu) = \sup_{f:\text{1-Lip}} \left(\int f \mathrm{d}\mu - \int f \mathrm{d}\nu \right). \tag{3}$$

Throughout this section, we fix $\mathbf{op}$ as the mean operator: $\mathbf{op}(\{y_1, \dots, y_N\}) = \frac{1}{N}\sum_{i=1}^{N} y_i$, and let $\mathbf{RN} = \phi_2 \circ \mathbf{op} \circ \phi_1$, where ϕ_1, ϕ_2 are two Lipschitz-continuous maps with Lipschitz constant C_1, C_2, respectively.

D.1.1 POINTWISE STABILITY

If $X = \{x_1, \dots, x_{N-1}, x_N\} \subset \Omega$ and $X' = \{x_1, \dots, x_{N-1}, x'_N\} \subset \Omega$, it is worth noting that $W_1(m_X, m_{X'}) \leq \frac{1}{N}\|x_N - x'_N\|$. 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 $\mathcal{O}(1/N)$. More generally, moving a fraction $\lambda \in (0, 1)$ of the points in X affects the Wasserstein distance in $\mathcal{O}(\lambda)$. $\mathbf{RN}$ satisfies the following stability result.

Proposition D.1. *For any two point clouds X, Y, and any $p \geq 1$, one has*

$$\begin{aligned} \|\mathbf{RN}(X) - \mathbf{RN}(Y)\| &\leq C_1 C_2 \cdot W_1(m_X, m_Y) \\ &\leq C_1 C_2 \cdot W_p(m_X, m_Y). \end{aligned}$$

Proof. We have

$$\begin{aligned} \|\mathbf{RN}(X) - \mathbf{RN}(Y)\| &= \|\phi_2(m_X(\phi_1)) - \phi_2(m_Y(\phi_1))\| \\ &\leq C_2 \|m_X(\phi_1) - m_Y(\phi_1)\| \\ &\leq C_2 C_1 \sup_{f:\text{1-Lip}} \|m_X(f) - m_Y(f)\| \\ &\leq C_2 C_1 W_1(m_X, m_Y), \end{aligned}$$

where we used equation 3 and we conclude $W_1 \leq W_p$ using Jensen's inequality. □

In particular, this result implies that moving a small proportion of points λ in Ω in a point cloud X does not affect the output of $\mathbf{RN}$ by much. We refer to it as a "pointwise stability" result in the sense that it describes how $\mathbf{RN}$ 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 $\lambda \to 0$, may change the resulting persistence diagram by a fixed positive amount, preventing a similar result to hold for PVs. A concrete example of this phenomenon is given in Appendix D.2 for the case of persistence images.

D.1.2 PROBABILISTIC STABILITY

The pointwise stability result of Proposition D.1 can 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 $\mathbf{P}$ denote $P^{\otimes N}$, that is, $X \sim \mathbf{P}$ 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 $X_1, \ldots, X_n \sim \mathbf{P}$, RipsNet is trained to minimize the empirical risk

$$\widehat{\mathcal{R}}_n := \frac{1}{n} \sum_{i=1}^{n} \|\mathbf{RN}(X_i) - \mathrm{PV}(X_i)\|,$$

which, hopefully, yields a small theoretical risk:

$$\mathcal{R} := \int \|\mathbf{RN}(X) - \mathrm{PV}(X)\| \mathrm{d}\mathbf{P}(X).$$

Remark D.2. The question to know whether "$\widehat{\mathcal{R}}_n$ 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 the following noise model: given a point cloud $X \sim \mathbf{P}$, we randomly replace a fraction $\lambda = \frac{N-K}{N} \in (0, 1)$ of its points[1] by corrupted observations distributed with respect to some law Q. Let $Y \sim Q^{\otimes N-K} =: \mathbf{Q}$ and $F(X, Y)$ denote this corrupted point cloud.

Lemma D.3. *Let $C(P, Q) := \mathbb{E}_{P \otimes Q}[\|x - y\|]$. Then,*

$$\mathbb{E}_{\mathbf{P} \otimes \mathbf{Q}}[W_1(F(X, Y), X)] \leq \lambda C(P, Q).$$

In particular, if P, Q are supported on a compact set $\Omega \subset \mathbb{R}^d$ with diameter $\leq L$, the bound becomes λL.

Proof. Set $X = \{x_1, \ldots, x_N\}$ and $Y = \{y_{K+1}, \ldots, y_N\}$. Assume without loss of generality that $F(X, Y) = \{x_1, \ldots, x_K, y_{K+1}, \ldots, y_N\}$, where $K = (1 - \lambda)N$. Let us consider the transport plan that does not move the first K points, and transports $V = \{y_{K+1}, \ldots, y_N\}$ toward $U = \{x_{K+1}, \ldots, x_N\}$ using the coupling $x_i \leftrightarrow y_i$. As this transport plan is sub-optimal, we have

$$W_1(F(X, Y), X) \leq \frac{1}{N} \sum_{i=K+1}^{N} \|x_i - y_i\|.$$

Hence

$$\begin{aligned}
&\mathbb{E}_{\mathbf{P} \otimes \mathbf{Q}}[W_1(F(X, Y), X)] \\
&= \int W_1(F(X, Y), X) \mathrm{d}\mathbf{P}(X) \mathrm{d}\mathbf{Q}(Y) \\
&\leq \frac{1}{N} \int \sum_{i=K+1}^{N} \|x_i - y_i\| \mathrm{d}P(x_i) \mathrm{d}Q(y_i) \\
&\leq \frac{1}{N} \sum_{i=K+1}^{N} \int \|x_i - y_i\| \mathrm{d}P(x_i) \mathrm{d}Q(y_i) \\
&\leq \frac{1}{N} \sum_{i=K+1}^{N} \mathbb{E}_{P \otimes Q} \|x - y\| \\
&\leq \lambda \cdot C(P, Q),
\end{aligned}$$

as claimed. □

We can now state the main result of this section.

[1] As the x_i's are i.i.d., we may assume without loss of generality that the last $N - K$ points are replaced.

Proposition D.4. *One has*

$$\begin{aligned}&\int \|\mathbf{RN}(F(X,Y)) - \mathrm{PV}(X)\| \mathrm{d}\mathbf{P}(X)\mathrm{d}\mathbf{Q}(Y)\\ \leq\ & \lambda C_1 C_2 \cdot C(P,Q) + \mathcal{R}.\end{aligned}$$

In particular, if P, Q are supported on a compact subset of $\mathbb{R}^d$ with diameter $\leq L$, one has

$$\int \|\mathbf{RN}(F(X,Y)) - \mathrm{PV}(X)\| \mathrm{d}\mathbf{P}(X)\mathrm{d}\mathbf{Q}(Y) \leq \mathcal{O}(\lambda + \mathcal{R}).$$

Proof. By Proposition D.1, we have

$$\begin{aligned}&\int \|\mathbf{RN}(F(X,Y)) - \mathrm{PV}(X)\| \mathrm{d}\mathbf{P}(X)\mathrm{d}\mathbf{Q}(Y)\\ \leq\ & \int \|\mathbf{RN}(F(X,Y)) - \mathbf{RN}(X)\| \mathrm{d}\mathbf{P}(X)\mathrm{d}\mathbf{Q}(Y)\\ &\quad + \int \|\mathbf{RN}(X) - \mathrm{PV}(X)\| \mathrm{d}\mathbf{P}(X)\\ \leq\ & C_1 C_2 \int W_1(F(X,Y), X) \mathrm{d}\mathbf{P}(X)\mathrm{d}\mathbf{Q}(Y) + \mathcal{R}\\ \leq\ & C_1 C_2 \mathbb{E}_{\mathbf{P}\otimes\mathbf{Q}}[W_1(F(X,Y), X)] + \mathcal{R}\end{aligned}$$

and we conclude using Lemma D.3. □

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

D.2 Instability of standard Rips persistence images

Here we show that persistence images built on Rips diagrams do not satisfy a similar stability result. Namely, the idea is to replace the "estimator" $\mathbf{RN}$ in the above section by the exact oracle PI, for which $\mathcal{R} = 0$, and to prove that

$$\int \|\mathrm{PI}(F(X,Y)) - \mathrm{PI}(X)\| \mathrm{d}\mathbf{P}\mathrm{d}\mathbf{Q} \not\to 0$$

in the regime $\lambda \to 0$ for some choice of underlying measures P, Q.

We consider the following setting:

- Let P be the uniform distribution on a circle in $\mathbb{R}^2$ centered at 0 with radius 1.
- Let Q be the Dirac mass on 0.
- Fix $K = N - 1$, that is, we move a single point of $X \sim \mathbf{P} = P^{\otimes N}$ to $\mathbf{Q} = Q = \delta_0$, hence $\lambda = 1/N$.
- We consider persistence diagrams of dimension 1, which represent loops in point clouds, and fix the variance σ^2 of the Gaussian used for the PI construction.

In the regime $\lambda \to 0$, that is, $N \to \infty$, we have

$$\mathrm{PI}(X) \to \mathcal{N}((0,1), \sigma^2) =: g_1$$

almost surely. On the other hand, we have

$$\mathrm{PI}(F(X,Y)) \to \mathcal{N}\left((0, 1/2), \sigma^2\right) =: g_2$$

almost surely, hence

$$\int \|\mathrm{PI}(F(X,Y)) - \mathrm{PI}(X)\| \mathrm{d}\mathbf{P}\mathrm{d}\mathbf{Q} \to \|g_1 - g_2\|_1 > 0,$$

which proves the claim.

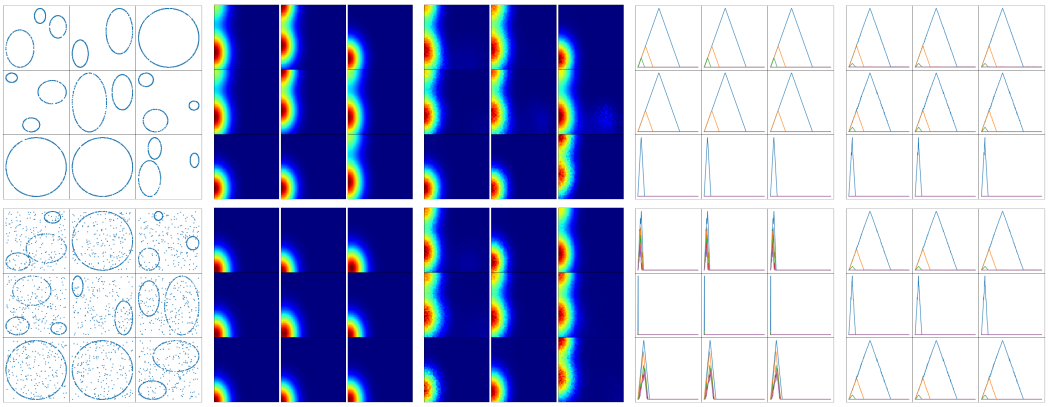

Figure 3: Point clouds (left), `Gudhi` vectorizations (2nd and 4th columns) and $\mathbf{RN}_{\mathrm{synth}}$ vectorizations (3rd and 5th columns) on clean data (1st row) and noisy data (2nd row) for PLs (4th and 5th columns) and PIs (2nd and 3rd columns). For a larger version, see Figure 4.

E ADDITIONAL NUMERICAL EXPERIMENTS

E.1 SYNTHETIC DATA

Dataset. 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 $\mathbf{RN}_{\mathrm{synth}}$ on a data set Tr_1 of 3300 point clouds, using 3000 point clouds for training and 300 as a validation set. The persistence diagrams PD_1 were computed with Alpha filtration in dimension 1 with `Gudhi`, and then vectorized into either the first 5 normalized persistence landscapes of resolution 300 each, leading to 1500-dimensional vectors, or into normalized persistence images of resolution 50×50, leading to 2500-dimensional vectors. The hyperparameters of these vectorizations were estimated from the corresponding persistence diagrams: the landscape limits were computed as the min and max of the x and y coordinates of the persistence diagrams points, while the image limits were computed as the min and max of the x and $y - x$ coordinates, respectively. Moreover, the image bandwidth was estimated as the 0.2-quantile of all pairwise distances between the birth-persistence transforms of the persistence diagram points, and the image weight was defined as $10 \cdot \tanh(y - x)$.

Our architecture $\mathbf{RN}_{\mathrm{synth}}$ is structured as follows. The permutation invariant operator is $\mathbf{op} = \mathrm{sum}$, ϕ_1 is made up of three fully connected layers of 30, 20, and 10 neurons with ReLU activations, ϕ_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 Adamax optimizer with $\varepsilon = 5 \cdot 10^{-4}$, and early stopping after 200 epochs with less than 10^{-5} improvement.

Finally, we evaluate $\mathbf{RN}_{\mathrm{synth}}$ and `Gudhi` using default XGBoost classifiers $\mathrm{Cl}^{\mathrm{XGB}}_{\mathbf{RN}_{\mathrm{synth}}}$ and $\mathrm{Cl}^{\mathrm{XGB}}_{\mathtt{Gudhi}}$ from `Scikit-Learn`, trained on a data set Tr_2 of 3000 point clouds and tested on a clean test set Te and a noisy test set $\widetilde{\mathrm{Te}}$ of 300 point clouds each. In addition, we compare it against two DeepSets architecture baselines trained directly on the point clouds: one ($\mathbf{DS}_1$) with four fully connected layers of 50, 30, 10, and 3 neurons, and a simpler one ($\mathbf{DS}_2$) with just two fully connected layers of 50 and 3 neurons. Both the architectures have ReLU activations, except for the last layer, permutation invariant operator $\mathbf{op} = \mathrm{sum}$, default Adam optimizer, and cross entropy loss from `TensorFlow`, and early stopping after 200 epochs with less than 10^{-4} improvement.

Results. We show some point clouds of Te and $\widetilde{\mathrm{Te}}$, as well as their corresponding vectorized persistence diagrams and estimated vectorizations with $\mathbf{RN}_{\mathrm{synth}}$, in Figure 3. Accuracies and running times (averaged over 10 runs) are given in Tables 2 and 3.

Synth. Data	$\mathrm{Cl}^{\mathrm{XGB}}_{\mathtt{Gudhi}}$	$\mathrm{Cl}^{\mathrm{XGB}}_{\mathtt{Gudhi}^{\mathrm{DTM}}}$	$\mathrm{Cl}^{\mathrm{XGB}}_{\mathbf{RN}}$	$\mathbf{DS}_1$	$\mathbf{DS}_2$
LS	99.9 ± 0.1	99.9 ± 0.1	80.7 ± 3.0	66.4 ± 2.3	66.0 ± 2.4
PI	$\mathbf{100.0 \pm 0.0}$	$\mathbf{100.0 \pm 0.1}$	81.6 ± 5.3	-	-
$\widetilde{\mathrm{LS}}$	66.7 ± 0.0	66.7 ± 0.0	76.3 ± 2.3	66.8 ± 1.0	66.6 ± 2.3
$\widetilde{\mathrm{PI}}$	33.3 ± 0.0	65.0 ± 1.3	$\mathbf{77.4 \pm 4.4}$	-	-
UCR Data	$\mathrm{Cl}^{\mathrm{XGB}}_{\mathtt{Gudhi}}$	$\mathrm{Cl}^{\mathrm{XGB}}_{\mathtt{Gudhi}^{\mathrm{DTM}}}$	$\mathrm{Cl}^{\mathrm{XGB}}_{\mathbf{RN}}$	$\mathbf{kNN}_{\mathrm{D}}$	$\mathbf{kNN}_{\mathrm{E}}$
P	70.5 ± 0.0	56.2 ± 0.0	$\mathbf{88.4 \pm 4.1}$	82.9 ± 0.0	78.1 ± 0.0
$\widetilde{\mathrm{P}}$	22.5 ± 2.6	53.9 ± 2.5	43.0 ± 7.9	$\mathbf{82.9 \pm 0.0}$	78.1 ± 0.6
SAIBORS2	63.6 ± 0.0	66.2 ± 0.0	$\mathbf{80.2 \pm 5.2}$	73.8 ± 0.0	72.4 ± 0.0
$\widetilde{\mathrm{SAIBORS2}}$	56.8 ± 0.8	60.0 ± 1.2	$\mathbf{75.6 \pm 6.6}$	73.7 ± 0.9	72.4 ± 0.4
ECG5000	84.2 ± 0.0	86.2 ± 0.0	90.2 ± 0.2	$\mathbf{93.0 \pm 0.0}$	92.8 ± 0.0
$\widetilde{\mathrm{ECG5000}}$	68.9 ± 0.8	71.6 ± 1.0	75.8 ± 4.7	$\mathbf{93.1 \pm 0.3}$	92.8 ± 0.1
UMD	55.6 ± 0.0	54.2 ± 0.0	$\mathbf{71.1 \pm 6.5}$	68.8 ± 0.0	61.1 ± 0.0
$\widetilde{\mathrm{UMD}}$	51.8 ± 1.9	48.9 ± 1.6	$\mathbf{69.2 \pm 6.4}$	68.3 ± 1.7	61.1 ± 0.4
GPOVY	98.4 ± 0.0	97.8 ± 0.0	90.4 ± 19.0	$\mathbf{100.0 \pm 0.0}$	100.0 ± 0.0
$\widetilde{\mathrm{GPOVY}}$	54.8 ± 0.7	54.3 ± 0.6	82.4 ± 20.7	$\mathbf{100.0 \pm 0.0}$	100.0 ± 0.0

Table 2: Accuracy scores of classifiers trained on Gudhi, $\mathtt{Gudhi}^{\mathrm{DTM}}$ and **RN** PVs generated from several data sets. The highest accuracy of the three topology-based classifiers (middle) is highlighted in red, and the highest accuracy over all models in bold font.

Data	Gudhi (s)	$\mathtt{Gudhi}^{\mathrm{DTM}}$ (s)	**RN** (s)
LS	56.3 ± 1.5	155.9 ± 8.1	$\mathbf{0.3 \pm 0.0}$
PI	69.5 ± 3.1	173.7 ± 13.3	$\mathbf{0.4 \pm 0.0}$
P	5.3 ± 1.4	44.7 ± 6.6	$\mathbf{0.2 \pm 0.0}$
UMD	8.0 ± 1.4	55.7 ± 3.6	$\mathbf{0.2 \pm 0.0}$
$\lambda = 2\%$	118.4 ± 4.7	178.5 ± 8.1	$\mathbf{0.2 \pm 0.0}$
$\lambda = 5\%$	117.8 ± 4.5	180.0 ± 9.2	$\mathbf{0.2 \pm 0.0}$

Table 3: Running times for Gudhi, $\mathtt{Gudhi}^{\mathrm{DTM}}$ and **RN**. The bottom two rows refer to the 3D-shape experiments.

As one can see from the figure and the tables, $\mathbf{RN}_{\text{synth}}$ manages to learn features that look like reasonable PD vectorizations, and that perform reasonably well on clean data. However, features generated by $\mathbf{RN}_{\text{synth}}$ are much more robust; even though $\text{Cl}^{\text{XGB}}_{\texttt{Gudhi}}$ and $\text{Cl}^{\text{XGB}}_{\texttt{Gudhi}^{\text{DTM}}}$ see their accuracies largely decrease when noise is added, $\text{Cl}^{\text{XGB}}_{\mathbf{RN}_{\text{synth}}}$ accuracy only decreases slightly. Note that the decrease of accuracy is more moderate for $\text{Cl}^{\text{XGB}}_{\texttt{Gudhi}^{\text{DTM}}}$ since $\texttt{Gudhi}^{\text{DTM}}$ is designed to be more robust to outliers. Running times are much more favorable for $\mathbf{RN}_{\text{synth}}$, with an improvement of 2 (resp. 3) orders of magnitude over `Gudhi` (resp. $\texttt{Gudhi}^{\text{DTM}}$) both for persistence images and landscapes.

E.2 Time series data

Data set. We apply our experimental setup on several data sets from the `UCR` archive, which contains data sets of time series separated into train and test sets. We first converted the time series into point clouds in $\mathbb{R}^3$ using time-delay embedding with skip 1 and delay 1 with `Gudhi`, and used the first half of the train set for training RipsNet architectures $\mathbf{RN}$, while the second half was used for training XGBoost classifiers. The amount of corrupted points was set up as 2%, i.e. $\lambda = 0.02$.

The hyperparameters were estimated exactly like for the synthetic data (see Section E.1), except that the final RipsNet architecture $\mathbf{RN}_{\text{ucr}}$ was found with 10-fold cross-validation across several models similar to the one used in Section E.1 and that persistence diagrams were computed in dimensions 0 and 1. They were optimized with Adam optimizer with $\varepsilon = 5 \cdot 10^{-4}$ and early stopping after 200 epochs with less than 10^{-5} improvement. We also focused on the first five persistence landscapes of resolution 50 only. The baseline is made of two default k-nearest neighbors classifiers from `Scikit-Learn`, trained directly on the time series: one (named $\mathbf{kNN}_{\text{E}}$) computed with Euclidean distance, and one (named $\mathbf{kNN}_{\text{D}}$) computed with dynamic time warping.

Results. Accuracies and running times (averaged over 10 runs) are given in Tables 2 and 3, and a more complete set of results (as well as the full data set names) can be found in Appendix F. As in the synthetic experiment, $\mathbf{RN}_{\text{ucr}}$ learns valuable topological features, which often perform better than `Gudhi` and $\texttt{Gudhi}^{\text{DTM}}$ on clean data, is more robust than `Gudhi` and $\texttt{Gudhi}^{\text{DTM}}$ on noisy data, and is much faster to compute. The fact that $\mathbf{RN}_{\text{ucr}}$ often achieves better scores than `Gudhi` and $\texttt{Gudhi}^{\text{DTM}}$ on clean data comes from the fact that the features learned by $\mathbf{RN}_{\text{ucr}}$ are more robust and less complex; while `Gudhi` encodes all the topological patterns in the data, some of which are potentially due to noise, $\mathbf{RN}_{\text{ucr}}$ only retains the most salient patterns when minimizing the MSE during training. Note however that when the training data set is too small for $\mathbf{RN}$ to train properly, robustness can be harder to reach, as is the case for the `Plane` data set.

F Additional Experimental Results and Figures

Data	$\mathrm{Cl}^{\mathrm{XGB}}_{\mathtt{Gudhi}}$	$\mathrm{Cl}^{\mathrm{XGB}}_{\mathtt{Gudhi}\,\mathrm{DTM}}$	$\mathrm{Cl}^{\mathrm{XGB}}_{\mathbf{RN}}$	$\mathbf{kNN}_{\mathrm{D}}$	$\mathbf{kNN}_{\mathrm{E}}$
CC	53.4 ± 0.0	52.0 ± 0.0	$\mathbf{55.8 \pm 1.3}$	55.2 ± 0.0	53.2 ± 0.0
$\widetilde{\mathrm{CC}}$	52.9 ± 0.6	53.7 ± 0.2	45.3 ± 2.0	$\mathbf{54.5 \pm 0.7}$	51.7 ± 0.5
PPTW	76.1 ± 0.0	71.2 ± 0.0	76.5 ± 1.1	$\mathbf{78.5 \pm 0.0}$	76.1 ± 0.0
$\widetilde{\mathrm{PPTW}}$	70.7 ± 2.3	68.0 ± 2.1	62.0 ± 7.6	$\mathbf{78.2 \pm 1.0}$	73.1 ± 1.6
P	70.5 ± 0.0	56.2 ± 0.0	$\mathbf{88.4 \pm 4.1}$	82.9 ± 0.0	78.1 ± 0.0
$\widetilde{\mathrm{P}}$	22.5 ± 2.6	53.9 ± 2.5	43.0 ± 7.9	$\mathbf{82.9 \pm 0.0}$	78.1 ± 0.6
GP	80.7 ± 0.0	$\mathbf{84.0 \pm 0.0}$	75.7 ± 6.6	66.7 ± 0.0	72.7 ± 0.0
$\widetilde{\mathrm{GP}}$	50.0 ± 0.7	50.5 ± 0.5	68.1 ± 5.3	66.8 ± 0.3	$\mathbf{72.8 \pm 0.5}$
POC	68.2 ± 0.0	64.2 ± 0.0	71.6 ± 2.8	72.6 ± 0.0	$\mathbf{74.6 \pm 0.0}$
$\widetilde{\mathrm{POC}}$	58.6 ± 0.0	57.4 ± 0.0	53.2 ± 0.0	71.8 ± 0.0	$\mathbf{74.2 \pm 0.0}$
SAIBORS2	63.6 ± 0.0	66.2 ± 0.0	$\mathbf{80.2 \pm 5.2}$	73.8 ± 0.0	72.4 ± 0.0
$\widetilde{\mathrm{SAIBORS2}}$	56.8 ± 0.8	60.0 ± 1.2	$\mathbf{75.6 \pm 6.6}$	73.7 ± 0.9	72.4 ± 0.4
PPOAG	78.5 ± 0.0	79.5 ± 0.0	81.1 ± 2.8	$\mathbf{82.9 \pm 0.0}$	82.9 ± 0.0
$\widetilde{\mathrm{PPOAG}}$	74.6 ± 1.9	72.7 ± 1.5	73.9 ± 3.6	82.0 ± 0.4	$\mathbf{83.1 \pm 1.3}$
ECG5000	84.2 ± 0.0	86.2 ± 0.0	90.2 ± 0.2	$\mathbf{93.0 \pm 0.0}$	92.8 ± 0.0
$\widetilde{\mathrm{ECG5000}}$	68.9 ± 0.8	71.6 ± 1.0	75.8 ± 4.7	$\mathbf{93.1 \pm 0.3}$	92.8 ± 0.1
ECG200	77.0 ± 0.0	70.0 ± 0.0	76.2 ± 1.6	78.0 ± 0.0	$\mathbf{85.0 \pm 0.0}$
$\widetilde{\mathrm{ECG200}}$	73.0 ± 2.9	70.2 ± 4.0	72.8 ± 2.7	78.4 ± 0.5	$\mathbf{85.0 \pm 0.6}$
MI	47.2 ± 0.0	46.8 ± 0.0	56.4 ± 2.2	$\mathbf{63.4 \pm 0.0}$	52.6 ± 0.0
$\widetilde{\mathrm{MI}}$	35.1 ± 2.2	36.6 ± 2.0	44.3 ± 3.4	$\mathbf{63.6 \pm 0.3}$	53.6 ± 0.5
PC	69.4 ± 0.0	74.4 ± 0.0	87.6 ± 6.0	82.8 ± 0.0	$\mathbf{97.8 \pm 0.0}$
$\widetilde{\mathrm{PC}}$	70.0 ± 2.7	72.7 ± 2.4	84.8 ± 5.3	84.2 ± 1.4	$\mathbf{97.9 \pm 0.2}$
DPOC	68.5 ± 0.0	69.2 ± 0.0	$\mathbf{74.0 \pm 2.0}$	73.9 ± 0.0	68.8 ± 0.0
$\widetilde{\mathrm{DPOC}}$	57.8 ± 2.1	58.7 ± 1.7	61.5 ± 3.6	$\mathbf{73.9 \pm 0.5}$	72.0 ± 0.9
IPD	70.4 ± 0.0	70.6 ± 0.0	79.0 ± 0.8	87.6 ± 0.0	$\mathbf{96.4 \pm 0.0}$
$\widetilde{\mathrm{IPD}}$	70.4 ± 0.0	70.6 ± 0.0	79.0 ± 0.8	87.6 ± 0.0	$\mathbf{96.4 \pm 0.0}$
MPOAG	54.5 ± 0.0	50.0 ± 0.0	55.6 ± 1.9	$\mathbf{56.5 \pm 0.0}$	52.0 ± 0.0
$\widetilde{\mathrm{MPOAG}}$	35.3 ± 4.6	33.0 ± 2.8	49.7 ± 7.4	$\mathbf{56.0 \pm 0.5}$	53.2 ± 2.5
SAIBORS1	53.0 ± 0.0	49.8 ± 0.0	$\mathbf{69.5 \pm 3.1}$	50.2 ± 0.0	45.0 ± 0.0
$\widetilde{\mathrm{SAIBORS1}}$	53.1 ± 0.8	49.5 ± 0.7	$\mathbf{67.9 \pm 5.8}$	50.2 ± 0.2	44.9 ± 0.2
UMD	55.6 ± 0.0	54.2 ± 0.0	$\mathbf{71.1 \pm 6.5}$	68.8 ± 0.0	61.1 ± 0.0
$\widetilde{\mathrm{UMD}}$	51.8 ± 1.9	48.9 ± 1.6	$\mathbf{69.2 \pm 6.4}$	68.3 ± 1.7	61.1 ± 0.4
TLECG	67.6 ± 0.0	71.8 ± 0.0	$\mathbf{78.6 \pm 11.0}$	75.8 ± 0.0	54.0 ± 0.0
$\widetilde{\mathrm{TLECG}}$	66.6 ± 1.0	69.6 ± 0.9	69.5 ± 6.6	$\mathbf{74.7 \pm 0.4}$	54.2 ± 0.3
MPOC	68.7 ± 0.0	66.7 ± 0.0	73.2 ± 2.0	74.2 ± 0.0	$\mathbf{77.7 \pm 0.0}$
$\widetilde{\mathrm{MPOC}}$	64.7 ± 1.4	64.2 ± 1.7	65.0 ± 2.6	73.3 ± 0.8	$\mathbf{76.7 \pm 0.9}$
GPOVY	98.4 ± 0.0	97.8 ± 0.0	90.4 ± 19.0	$\mathbf{100.0 \pm 0.0}$	100.0 ± 0.0
$\widetilde{\mathrm{GPOVY}}$	54.8 ± 0.7	54.3 ± 0.6	82.4 ± 20.7	$\mathbf{100.0 \pm 0.0}$	100.0 ± 0.0
MPTW	52.0 ± 0.0	$\mathbf{52.6 \pm 0.0}$	51.7 ± 1.3	50.6 ± 0.0	52.0 ± 0.0
$\widetilde{\mathrm{MPTW}}$	36.5 ± 1.6	35.2 ± 3.5	46.5 ± 4.5	51.0 ± 1.3	$\mathbf{51.2 \pm 1.8}$
CBF	64.4 ± 0.0	63.8 ± 0.0	61.6 ± 10.9	$\mathbf{78.2 \pm 0.0}$	58.4 ± 0.0
$\widetilde{\mathrm{CBF}}$	63.8 ± 1.1	62.5 ± 1.6	55.0 ± 9.3	$\mathbf{79.0 \pm 0.4}$	57.7 ± 0.3

Table 4: Accuracy scores of XGBoost and k-NN classifiers on UCR data sets.

Data	Name	Gudhi (s)	Gudhi^{DTM} (s)	**RN** (s)
CC	ChlorineConcentration	33.6 ± 9.3	310.0 ± 36.1	$\mathbf{0.2 \pm 0.0}$
$\widetilde{\mathrm{CC}}$	-	25.3 ± 2.5	298.2 ± 43.6	$\mathbf{0.2 \pm 0.0}$
PPTW	ProximalPhalanxTW	5.8 ± 1.3	25.3 ± 4.5	$\mathbf{0.2 \pm 0.0}$
$\widetilde{\mathrm{PPTW}}$	-	5.9 ± 1.3	25.6 ± 4.6	$\mathbf{0.2 \pm 0.0}$
P	Plane	5.3 ± 1.4	44.7 ± 6.6	$\mathbf{0.2 \pm 0.0}$
$\widetilde{\mathrm{P}}$	-	5.3 ± 1.5	44.8 ± 6.2	$\mathbf{0.2 \pm 0.0}$
GP	GunPoint	6.6 ± 1.5	53.7 ± 7.9	$\mathbf{0.2 \pm 0.0}$
$\widetilde{\mathrm{GP}}$	-	6.6 ± 1.5	55.1 ± 8.0	$\mathbf{0.2 \pm 0.0}$
POC	PhalangesOutlineCorrect	22.0 ± 5.1	91.6 ± 19.5	$\mathbf{0.2 \pm 0.0}$
$\widetilde{\mathrm{POC}}$	-	11.9 ± 0.0	56.2 ± 0.0	$\mathbf{0.2 \pm 0.0}$
SAIBORS2	SonyAIBORobotSurface2	10.0 ± 0.2	26.3 ± 0.6	$\mathbf{0.2 \pm 0.0}$
$\widetilde{\mathrm{SAIBORS2}}$	-	10.2 ± 0.2	26.9 ± 0.6	$\mathbf{0.2 \pm 0.0}$
PPOAG	ProximalPhalanxOutlineAgeGroup	6.8 ± 0.2	28.9 ± 0.8	$\mathbf{0.2 \pm 0.0}$
$\widetilde{\mathrm{PPOAG}}$	-	6.9 ± 0.2	29.4 ± 0.9	$\mathbf{0.2 \pm 0.0}$
ECG5000	ECG5000	27.6 ± 8.0	187.1 ± 29.2	$\mathbf{0.2 \pm 0.0}$
$\widetilde{\mathrm{ECG5000}}$	-	27.2 ± 8.6	185.5 ± 32.3	$\mathbf{0.2 \pm 0.0}$
ECG200	ECG200	3.8 ± 1.2	14.9 ± 4.3	$\mathbf{0.1 \pm 0.0}$
$\widetilde{\mathrm{ECG200}}$	-	3.8 ± 1.1	15.1 ± 3.9	$\mathbf{0.2 \pm 0.0}$
MI	MedicalImages	16.8 ± 2.5	79.8 ± 9.4	$\mathbf{0.3 \pm 0.1}$
$\widetilde{\mathrm{MI}}$	-	16.5 ± 2.4	80.1 ± 8.1	$\mathbf{0.3 \pm 0.0}$
PC	PowerCons	11.2 ± 2.6	77.1 ± 10.0	$\mathbf{0.2 \pm 0.0}$
$\widetilde{\mathrm{PC}}$	-	11.3 ± 2.8	76.9 ± 11.4	$\mathbf{0.2 \pm 0.0}$
DPOC	DistalPhalanxOutlineCorrect	9.4 ± 1.6	39.2 ± 5.3	$\mathbf{0.2 \pm 0.0}$
$\widetilde{\mathrm{DPOC}}$	-	9.4 ± 1.5	39.4 ± 5.1	$\mathbf{0.2 \pm 0.0}$
IPD	ItalyPowerDemand	2.7 ± 0.5	3.9 ± 0.6	$\mathbf{0.2 \pm 0.0}$
$\widetilde{\mathrm{IPD}}$	-	2.7 ± 0.5	3.9 ± 0.6	$\mathbf{0.2 \pm 0.0}$
MPOAG	MiddlePhalanxOutlineAgeGroup	5.8 ± 1.0	24.1 ± 3.3	$\mathbf{0.2 \pm 0.0}$
$\widetilde{\mathrm{MPOAG}}$	-	5.8 ± 0.9	24.1 ± 3.2	$\mathbf{0.2 \pm 0.0}$
SAIBORS1	SonyAIBORobotSurface2	10.4 ± 1.5	28.4 ± 4.2	$\mathbf{0.2 \pm 0.0}$
$\widetilde{\mathrm{SAIBORS1}}$	-	10.5 ± 1.5	28.6 ± 4.2	$\mathbf{0.2 \pm 0.0}$
UMD	UMD	8.0 ± 1.4	55.7 ± 3.6	$\mathbf{0.2 \pm 0.0}$
$\widetilde{\mathrm{UMD}}$	-	8.0 ± 1.3	55.6 ± 2.6	$\mathbf{0.2 \pm 0.0}$
TLECG	TwoLeadECG	11.0 ± 0.4	40.8 ± 1.5	$\mathbf{0.2 \pm 0.0}$
$\widetilde{\mathrm{TLECG}}$	-	11.1 ± 0.3	41.2 ± 0.9	$\mathbf{0.2 \pm 0.0}$
MPOC	MiddlePhalanxOutlineCorrect	9.6 ± 2.1	39.3 ± 8.1	$\mathbf{0.2 \pm 0.0}$
$\widetilde{\mathrm{MPOC}}$	-	9.6 ± 2.1	39.4 ± 7.9	$\mathbf{0.2 \pm 0.0}$
GPOVY	GunPointOldVersusYoung	12.1 ± 3.8	108.1 ± 21.5	$\mathbf{0.2 \pm 0.0}$
$\widetilde{\mathrm{GPOVY}}$	-	12.3 ± 3.8	108.8 ± 19.6	$\mathbf{0.2 \pm 0.0}$
MPTW	MiddlePhalanxTW	6.3 ± 0.2	26.3 ± 1.0	$\mathbf{0.2 \pm 0.0}$
$\widetilde{\mathrm{MPTW}}$	-	6.4 ± 0.2	26.1 ± 0.8	$\mathbf{0.2 \pm 0.0}$
CBF	CBF	25.2 ± 4.5	120.4 ± 10.1	$\mathbf{0.2 \pm 0.0}$
$\widetilde{\mathrm{CBF}}$	-	25.0 ± 4.4	120.5 ± 10.2	$\mathbf{0.3 \pm 0.1}$

Table 5: Data set names and running times for Gudhi, Gudhi^{DTM} and $\mathbf{RN}_{\mathrm{ucr}}$ on UCR data sets.

λ (%)	$\mathrm{Cl}^{\mathrm{NN}}_{\mathtt{Gudhi}}$	$\mathrm{Cl}^{\mathrm{NN}}_{\mathtt{Gudhi}^{\mathrm{DTM}}}$	$\mathrm{Cl}^{\mathrm{NN}}_{\mathbf{RN}}$	`pointnet`
0	30.4 ± 4.0	30.9 ± 2.0	53.9 ± 2.4	$\mathbf{81.6 \pm 1.1}$
2	30.3 ± 3.2	31.0 ± 2.7	53.2 ± 2.5	$\mathbf{74.5 \pm 1.6}$
5	29.9 ± 4.0	31.0 ± 2.7	55.1 ± 3.3	$\mathbf{63.4 \pm 1.6}$
10	25.2 ± 3.2	29.5 ± 3.1	$\mathbf{51.0 \pm 2.1}$	50.6 ± 1.5
15	22.9 ± 4.6	25.7 ± 3.1	$\mathbf{46.9 \pm 3.0}$	44.9 ± 1.7
25	14.4 ± 4.0	18.1 ± 2.6	$\mathbf{42.6 \pm 2.5}$	11.0 ± 0.2
50	14.0 ± 3.4	13.1 ± 1.9	$\mathbf{31.6 \pm 3.3}$	10.9 ± 0.0
75	11.3 ± 1.5	11.2 ± 2.0	$\mathbf{17.0 \pm 2.3}$	10.9 ± 0.0
90	11.0 ± 2.4	10.8 ± 3.1	$\mathbf{12.8 \pm 2.8}$	10.9 ± 0.0

Table 6: Accuracy scores of simple neural net classifiers of `Gudhi` and **RN** on `ModelNet10`. λ is the noise fraction and $(y-x)^2$ was used as persistence image weight function. The highest accuracy of the three topology based models $\mathrm{Cl}^{\mathrm{NN}}_{\mathtt{Gudhi}}$, $\mathrm{Cl}^{\mathrm{NN}}_{\mathtt{Gudhi}^{\mathrm{DTM}}}$ and $\mathrm{Cl}^{\mathrm{NN}}_{\mathbf{RN}}$ is highlighted in red, and the highest accuracy over all models, including the `pointnet` baseline, is highlighted in bold font.

λ (%)	$\mathrm{Cl}^{\mathrm{XGB}}_{\mathtt{Gudhi}}$	$\mathrm{Cl}^{\mathrm{XGB}}_{\mathtt{Gudhi}^{\mathrm{DTM}}}$	$\mathrm{Cl}^{\mathrm{XGB}}_{\mathbf{RN}}$	`pointnet`
0	32.2 ± 2.8	31.6 ± 2.0	49.1 ± 2.2	$\mathbf{81.6 \pm 1.1}$
2	31.0 ± 4.9	30.9 ± 2.8	48.3 ± 3.0	$\mathbf{74.5 \pm 1.6}$
5	30.4 ± 2.6	30.9 ± 3.0	48.0 ± 3.2	$\mathbf{63.4 \pm 1.6}$
10	28.3 ± 2.0	27.6 ± 2.0	46.0 ± 2.2	$\mathbf{50.6 \pm 1.5}$
15	26.6 ± 2.8	28.2 ± 2.6	43.3 ± 2.7	$\mathbf{44.9 \pm 1.7}$
25	21.6 ± 2.9	25.6 ± 2.0	$\mathbf{40.7 \pm 2.8}$	11.0 ± 0.2
50	15.3 ± 2.0	15.7 ± 1.9	$\mathbf{27.8 \pm 2.7}$	10.9 ± 0.0
75	12.8 ± 1.5	11.9 ± 1.1	$\mathbf{19.4 \pm 1.6}$	10.9 ± 0.0
90	13.0 ± 2.1	11.1 ± 0.9	$\mathbf{13.1 \pm 2.4}$	10.9 ± 0.0

Table 7: Accuracy scores of XGBoost classifiers of `Gudhi` and **RN** on `ModelNet10`. λ is the noise fraction and $(y-x)^2$ was used as persistence image weight function. The highest accuracy of the three topology based models $\mathrm{Cl}^{\mathrm{XGB}}_{\mathtt{Gudhi}}$, $\mathrm{Cl}^{\mathrm{XGB}}_{\mathtt{Gudhi}^{\mathrm{DTM}}}$ and $\mathrm{Cl}^{\mathrm{XGB}}_{\mathbf{RN}}$ is highlighted in red, and the highest accuracy over all models, including the `pointnet` baseline, is highlighted in bold font.

λ (%)	`Gudhi` (s)	$\mathtt{Gudhi}^{\mathrm{DTM}}$ (s)	**RN** (s)
2	118.4 ± 4.7	178.5 ± 8.1	$\mathbf{0.2 \pm 0.0}$
5	117.8 ± 4.5	180.0 ± 9.2	$\mathbf{0.2 \pm 0.0}$
10	117.5 ± 4.6	181.9 ± 8.1	$\mathbf{0.2 \pm 0.0}$
15	120.0 ± 4.7	178.7 ± 8.4	$\mathbf{0.3 \pm 0.0}$
25	121.2 ± 4.4	179.8 ± 7.8	$\mathbf{0.2 \pm 0.0}$
50	127.0 ± 6.4	196.5 ± 10.7	$\mathbf{0.2 \pm 0.0}$

Table 8: Running times on `ModelNet10` data in seconds, for `Gudhi`, $\mathtt{Gudhi}^{\mathrm{DTM}}$ and **RN**, respectively.

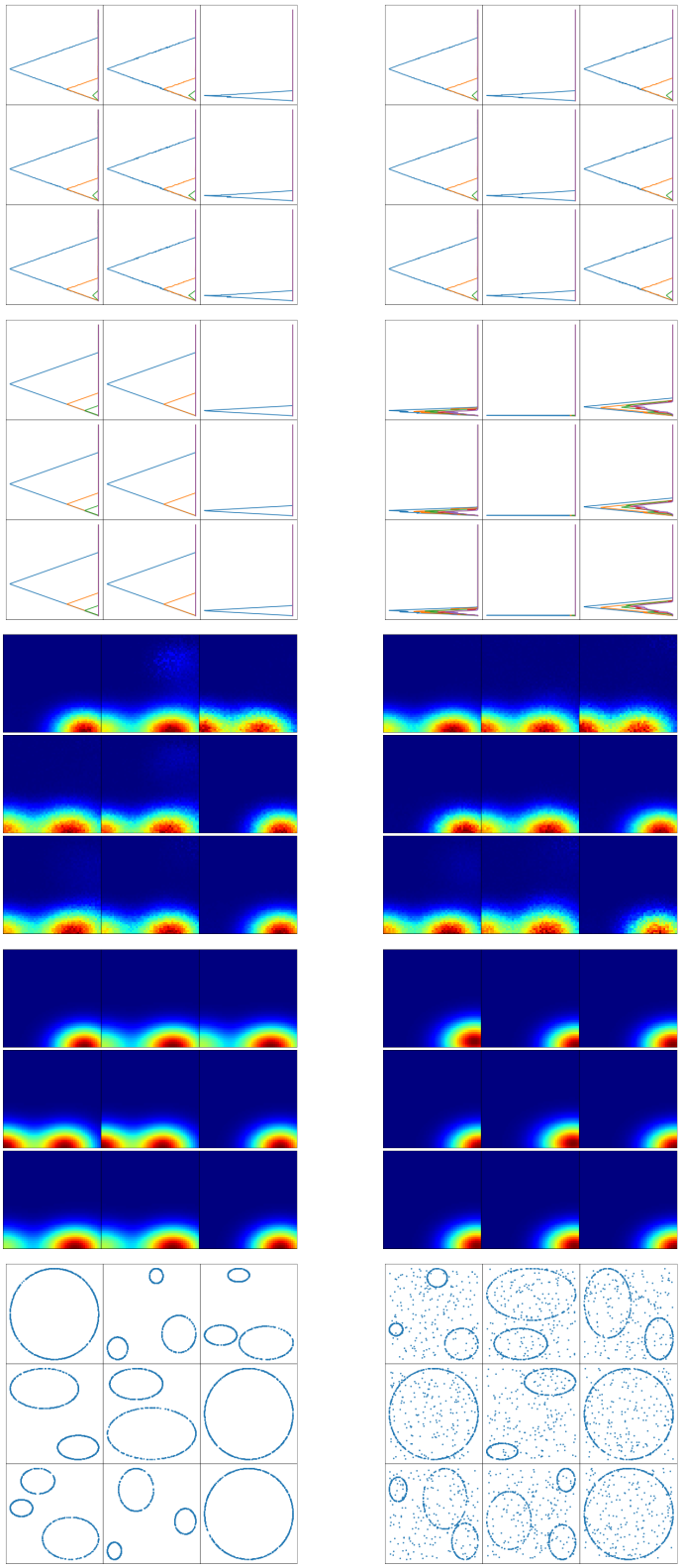

Figure 4: Larger version of Figure 3