Learnability of Topological Features in Convolutional Networks

Amir Shanehsazzadeh[*]
Harvard University
Cambridge, MA 02138
amirshanehsazzadeh@college.harvard.edu

Karthik Yegnesh[*]
Harvard University
Cambridge, MA 02138
karthikyegnesh@college.harvard.edu

William Zhang[*]
Harvard University
Cambridge, MA 02138
william_zhang@college.harvard.edu

Abstract

It is known in the literature that Convolutional Neural Networks (CNNs) have difficulty in learning global features of images. We give experimental evidence of this phenomenon using a toy data set to show that basic CNNs are ineffective in learning topological features (holes, connectedness, etc.) in images. We also introduce models using persistent homology, a computational method for extracting topological features from point clouds, and show that these models more effectively learn topological information from our toy data set than basic CNNs. Lastly, we attempt to use topological features of images to improve the accuracy of a basic CNN in a handwritten character recognition task. To do this, we use persistent homology to build a neural network that incorporates both convolutional layers and topological features of images, although the results of our experiments are inconclusive.

1 Introduction

CNNs have become the basis for the state of the art in a number of computer vision tasks such as image classification. One major concern with these models, as well as other models in machine learning, is robustness. A robust model is resistant to small perturbations in the task, such as minor modifications to the data set. CNNs fall victim to learning local structures like textures very effectively while sacrificing information about global structure. Small perturbations can in fact lead to a high-performing CNN model performing tragically. For example, adding a red dot in the corner of certain images or replacing an elephant's skin with cat fur, without changing the shape representing the elephant's outline, causes drastic worsening in image classification (1), (5). This lack of robustness implies that CNN models learn local features of the image but not global features.

The goal of this project is twofold.

1. First, we study the extent to which CNNs can learn topological features in images, such as the number of holes present in the image. As CNNs tend to learn local features in images, we do not expect CNNs to effectively learn topological data, which are global features of images. By running experiments on a toy data set, we show that a basic CNN model struggles to learn the number of holes present in image data. We also introduce

[*]Equal contribution.

neural network models utilizing *persistent homology*, a technique for computing topological features (holes, connectedness, etc.) of data sets. We show that these models perform better than the basic CNN for our initial task, giving some evidence that persistent homology may be useful in extracting topological information that could improve deep learning models.

2. Second, we attempt to use topological features of images (computed using persistent homology) to improve the performance of a basic CNN in image classification. We build a neural network that incorporates both convolutional layers as well as persistent homology data from images and test it against a basic CNN in classifying handwritten letters and numbers. Although our new model achieves a higher test accuracy than the basic CNN in this task, its performance does not change when a random shuffle of the topological data is fed into the network instead of the true topological data. Our experiments indicate that the network essentially ignores the topological features inputted, so that extracting topological features from images did not contribute to better performance. Although this is a disappointing result, it still remains to be seen whether persistent homology data may be useful for image classification in other settings.

The paper is organized as follows: we first give an introduction to persistent homology as it pertains to extracting topological information from images for this project in Section 2. We then describe our experiments pertaining to goal (1) in Section 3 and goal (2) in Section 4. A concluding discussion is given in Section 5.

We document our work in two .ipynb notebooks that we run on the Google Colaboratory environment (`https://github.com/wzhang2022/CS281-Project`).

2 Informal Introduction to Persistent Homology

We give a very brief and informal exposition of persistent homology in this section - formal definitions and constructions pertaining to homology theory and persistence are given in Appendix A. Persistent homology is a powerful method for detecting topological features in point cloud data. By topological features, we refer to k-dimensional "holes" in the data: a 0-dimensional hole is a connected component, a 1-dimensional hole is a usual hole, a 2-dimensional hole is a void, etc. This is made formal via homology theory, but an intuitive picture will suffice for understanding the motivation and utility of persistent homology.

Suppose we have a data set $D \subset \mathbb{R}^n$ that was sampled from some manifold $X \subset \mathbb{R}^n$. We would like to be able to infer the topological features of X (for example how many holes X has or how many connected components comprise it) given the sample data points D. For instance, if X is the disjoint union of two annuli inside $\mathbb{R}^2$, then the sampling D might look like the point cloud in Figure 1.

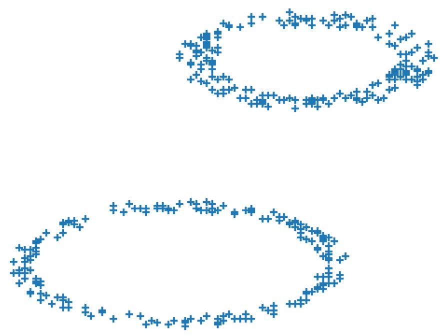


Figure 1: Point Cloud Densely Sampled from Two Annuli

Intuitively, it is clear that D was sampled from a space with two holes and two connected components, but persistent homology provides a computational method for detecting holes in potentially large and high-dimensional point clouds. If we somehow had access to the underlying space X or some combinatorial representation of X, then methods from algebraic topology, such as homology theory, could be used to compute the number of holes in X. However, since we only have access to the sampling D, we need an approach that reconstructs the topological features of X given D.

The key idea of persistent homology is to construct a sequence of increasingly larger spaces (a *Vietoris-Rips* filtration) that represents the structure of the point cloud D. Topological features that "persist" through stages of this filtration are thought of as the largest and most important features of the space, while features that are created and quickly destroyed are regarded as noise in the point cloud. Persistent homology algorithms essentially compute the indices in the filtration at which each feature is created/destroyed, so that we obtain a compact representation of the topology of the underlying space in the form of a collection of points called a *persistence diagram*. A formal construction of the Vietoris-Rips filtration is contained in the appendix.

To illustrate this, we compute the persistence diagram of the point cloud in Figure 1 using Ripser.py, a state-of-the-art Python package for computing persistent homology which is based on the C++ Ripser package (2). The results are shown in Figure 2.

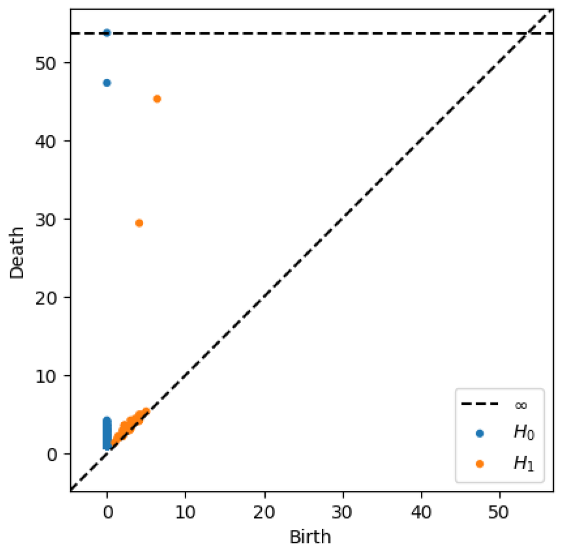


Figure 2: Persistence Diagram Corresponding to Data Sampled from Two Annuli

The blue points in Figure 2 represent the 0-dimensional features (connected components) in the filtration while the orange points represent the 1-dimensional features, or holes, in the data. We refer to the points corresponding to k-dimensional holes as H_k*-points*. Each point $x_i = (c_i, d_i)$ in indicates that a feature was created at index c_i in the filtration and destroyed at index d_i. We refer to $d_i - c_i$ as the *lifetime* of x_i. The lifetime of each point x_i gives a measure of how important the corresponding feature is to the structure of the point cloud - points near the diagonal are part of the noise inherent in the sampling while points x_i with long lifetimes values represent important topological features (these will be far from the diagonal).

In Figure 2, the four points far from the diagonal show that there are two important holes and two connected components in the space the point cloud in Figure 1 was sampled from.

3 Counting Holes

We first ran experiments to test to what extent a vanilla CNN could learn to predict the number of holes in an image, and if a model that directly incorporates persistent homology data from images can more effectively learn topological features. We define holes in a geometric sense to mean the center of an outlined shape or the center of a finite intersection of outlined shapes, where an outlined shape is for example the boundary of a circle or square.

Our broad conclusions from this section are that our basic CNN architecture does not effectively learn to count holes in our toy data set, while our model incorporating persistence data performs much better. Additionally, augmenting the vanilla CNN model with an ordinal regression improves test performance, but this is still not as effective as the model directly incorporating persistence data. The results of our experiments are summarized in Table 1 (accuracy/test loss) and Figure 4 (training/validation loss).

3.1 Data

We constructed a toy data set of images for this task. For each sample we start with a square 200×200 pixel image with a black background and insert white circles. The number of white circles is sampled uniformly between 2 and 5. For each circle, its pixel width is sampled uniformly between 2 and 5 and the coordinates for its center are both sampled uniformly between 40 and 160. We generate 10000 samples and create a 70/10/20 train/validation/test split. For counting the number of holes we instead counted the number of black connected components and subtracted 1. We did this computation using a flood-fill algorithm. To see why these quantities are equivalent consider Figure 3 below. We group all the images with more than 10 holes into a single class due to the rarity of these images in the data, which comprise about $\sim 3\%$ of the dataset. Similarly, we group the images with 2 or 1 holes into the same class. Throughout our experiments, we use stratified sampling to ensure that each class is sampled uniformly.

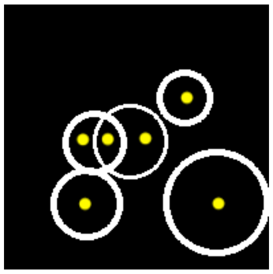


Figure 3: Sample from Data Set with 6 Holes

Model	Vanilla CNN	Ordinal CNN	TopNet	TopCNN
Accuracy	0.1825	0.2725	0.3395	0.2755
Test Loss	3.4660	2.0494	1.5806	1.8910

Table 1: Accuracy and Test Loss for Different Models on Toy Data

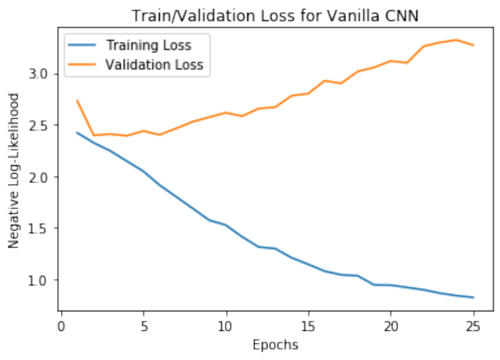
3.2 Baseline CNN Model (Vanilla CNN)

For our baseline model we use a deep 3-layer CNN with max pooling and two fully connected layers and softmax activation applied to the final output. The performance of this model was rather poor, being able to predict the correct number of holes only 18.25% of the time and achieving a test loss (Negative Log-Likelihood) of 3.4660. See Table 1 for full results. We noted that the vanilla CNN does, however, perform better than if it were to simply learn the empirical distribution of the labels. We hypothesize that the model is able to do this by counting the number of black pixels in the image (or equivalently white pixels), which is correlated with the number of holes in the data.

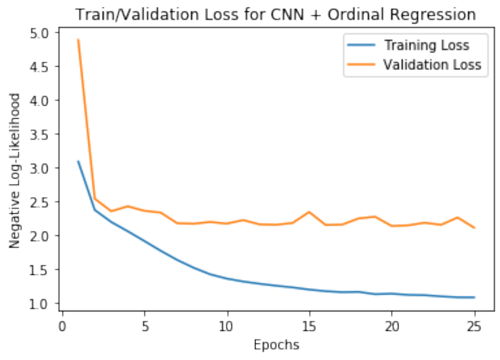
3.3 CNN with Ordinal Regression (Ordinal CNN)

An immediate issue with this first approach in Section 3.2 is that it treats the problem as a standard classification task. However, the true labels are ordinal rather than categorical. We solve this with the cumulative logistic link function (3), a modified form of multi-class logistic regression. Specifically, let σ denote the sigmoid function and call our neural network model N. Suppose that the output of our model for an input x is a real number: $N(x) \in \mathbb{R}$ and that the task is to predict the label of x as one of k ordinal classes. We take a partition $-\infty = x_0 < x_1 < x_2 < ... < x_k < x_{k+1} = \infty$ and consider the intervals $[x_0, x_1], [x_1, x_2], ..., [x_k, x_{k+1}]$. Let $\hat{y}(x, N)$ denote the prediction for sample x under model N. The cumulative logistic link function is defined so that for $j \in \{1, 2, ..., k\}$

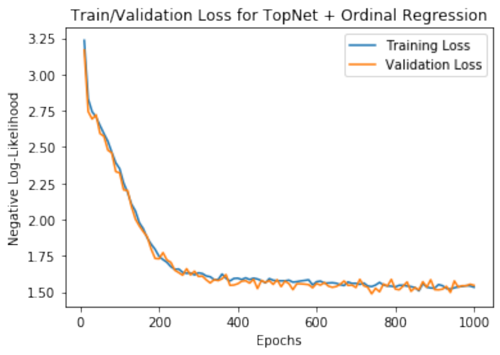
$$\mathbb{P}(\hat{y}(x, N) = j)) = \begin{cases} \sigma(x_1 - N(x)) & j = 1 \\ \sigma(x_j - N(x)) - \sigma(x_{j-1} - N(x)) & 1 < j < k \\ 1 - \sigma(x_j - N(x)) & j = k \end{cases}$$



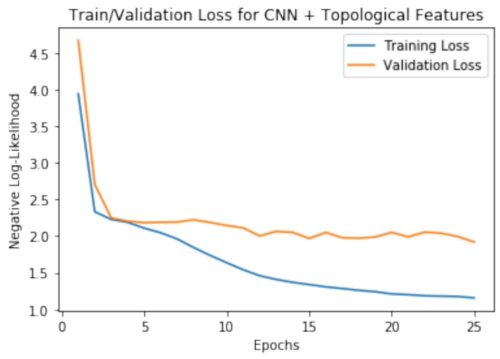
(a) Vanilla CNN



(b) Ordinal CNN



(c) TopNet



(d) TopCNN

Figure 4: Training and Validation Loss for Models on Toy Data

Intuitively, this function offers an improvement since each interval $[x_i, x_{i+1}]$ is adjacent to $[x_{i-1}, x_i]$ so that the closer $N(x)$ is to its appropriate (true label) interval, the more accurate the model N is.

We implemented this link function in place of ordinary softmax and noticed a substantial improvement in both accuracy and test loss (Table 1), motivating the use of an ordinal regression layer for the other architectures. Our implementation used a uniform partition $x_i = i - 1$ for each $i = 1, \dots, k$. Here $k = 10$ because the number of holes are categorized as $\leq 2, 3, \dots, 10, \geq 11$. Because our choice of partition is entirely arbitrary, we attempt to include these cutoffs as trainable parameters. Surprisingly, this leads to worse classification results and oftentimes failing to train. This is likely due to the gradients pushing the $x_1, \dots, x_k$ out of sorted order, causing the model to predict negative probabilities.

3.4 Multi-Layer Perceptron with Topological Features (TopNet)

As our baseline CNN model is unable to learn the number of holes in the images of our toy data set, we build a model relying on persistent homology to more effectively learn the topological features of our data. Our model, TopNet, has a simple feedforward architecture which takes as input the lifetimes of the H_1 points of the persistence diagram corresponding to each image. Figure 5 presents TopNet's architecture.

For each image we use the Ripser package in Python to compute the H_1 points of the persistence diagram of a point cloud sampled from the image. We then store the 20 highest lifetimes of the H_1 points of this diagram. Features with longer lifetimes correspond directly to holes in the image. This data is fed into a sparse linear network with output dimension 20 that is then fed into a fully connected layer with output dimension 1 after applying sigmoidal activations. This is then fed into the ordinal regression method to obtain a predicted number of holes.

The intuition behind this architecture is that we want the network to be forced to count outliers - points with very high lifetimes correspond to holes in the space from which the point cloud was sampled from. The sigmoidal activations then send high values close to 1 and low values close to 0 - the final layer is meant to sum these values, which should approximate the number of outliers (with respect to lifetime) in the H_1 points of the persistence diagram.

The TopNet model achieves a 33.95% test accuracy using our toy data set, which is a significant improvement over the Vanilla CNN performance (although far from perfect). This gives evidence that using persistence data as input into a model allows the model to learn topological features.

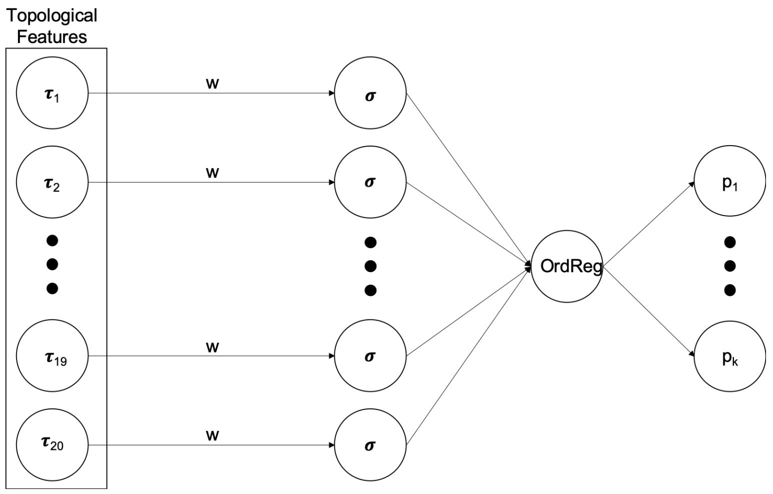


Figure 5: TopNet Architecture

3.5 CNN with Topological Features (TopCNN)

We incorporate the topological features into our CNN model for our toy data set by concatenating the output of the Vanilla CNN with the output from TopNet and applying two fully connected layers into ordinal regression. The performance of this model, which we denote as TopCNN, is comparable, and in fact slightly better, to that of the CNN with ordinal regression. We suspect that the marginally better performance is a result of an approximate averaging effect between the CNN and TopNet models with ordinal regression. Figure 6 presents TopCNN's architecture.

4 Topology for Image Classification

After running our first set of experiments and noticing an improvement in the model's ability to predict global features, we wanted to see if the incorporation of topological features could be applied to image classification. Specifically, we wanted to see if these features could be used to improve the ability of the model to classify images as well as to increase the robustness of the model.

4.1 Data

We used a subset of handwritten images from the Chars74k data set (8). This data set consists of 62 classes (numbers 0-9, lower-case letters a-z, and uppercase letters A-Z) each with 55 labeled examples. Each sample is a 1200×900 pixel image with black and white pixels. We had initially thought of using the more popular MNIST data set, but we choose Chars74k because it has more labels and greater topological diversity.

4.2 Baseline CNN Model (Vanilla CNN)

For this classification task a baseline model is first built using a similar procedure as in Section 3.2. The model is a deep 4-layer CNN with max pooling and 4 fully-connnected layers with softmax

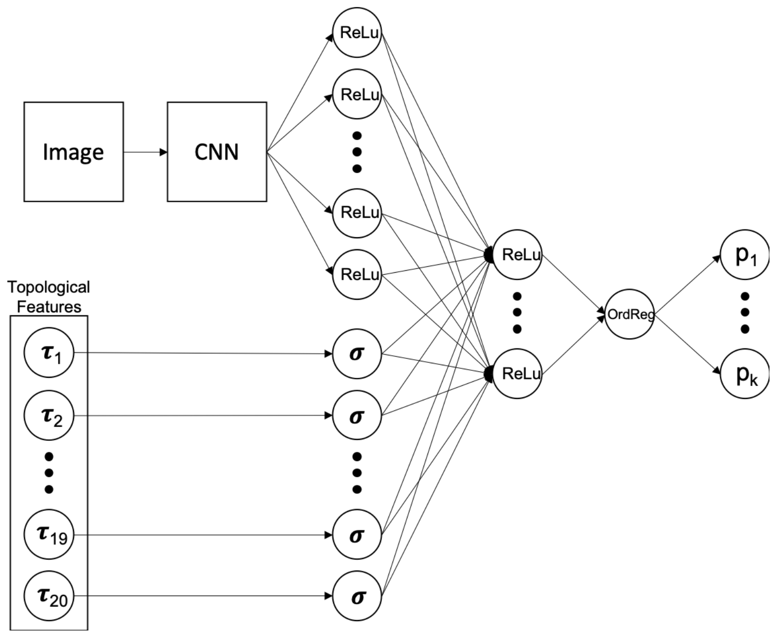
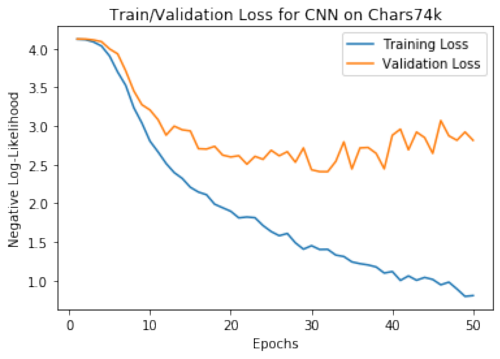
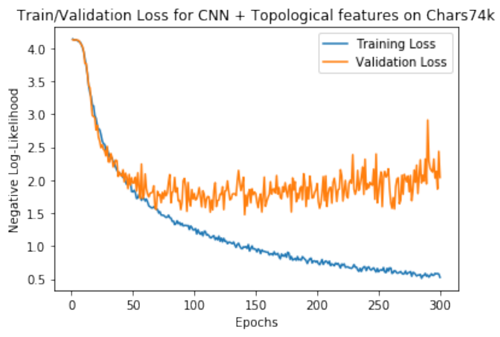


Figure 6: TopCNN Architecture



(a) Vanilla CNN



(b) CNN + Topological Features

Figure 7: Training and Validation Loss for Models on Chars74k Data

Model	Vanilla CNN	TopCNN	TopCNN-shuffled
Accuracy	0.3827	0.5543	0.5616
Test Loss	2.6642	1.8030	1.7312

Table 2: Accuracy and Test Loss for Different Models on Chars74k Data

activation. This model achieves a test accuracy of 38.27% and a test loss of 2.6642. It is clear from Figure 7(a) that the model begins to overfit after approximately 30 epochs.

4.3 TopCNN

To see if the incorporation of topological features would improve classification the TopCNN model was used, with procedure similar to that of Section 3.5. The primary architecture change is the addition of a second fully-connected layer following concatenation. Also, Softmax is used in place of Ordinal Regression since the task is standard classification. TopCNN did perform better than the Baseline CNN achieving a test accuracy of 55.43% and a test loss of 1.8030. Figure 7(b) shows that TopCNN begins overfitting after approximately 125 epochs.

However, the improvement in test accuracy and loss from using TopCNN as opposed to Vanilla CNN is not due to the model learning from the topological features, but rather from the additional size of the model. To show this, the topological features are randomly shuffled in the train set but not in the test set and, as seen in Table 2, the performance of the model is not affected. We choose shuffling the topological features as the appropriate baseline to compare the effectiveness of using topological features because ensures that the topological features are drawn from the same distribution. Surprisingly, we also find that setting all the entries of each topological features vector to 10,000 has little impact on the classification accuracy and test loss. This leads us to conclude that the TopCNN architecture simply learns to ignore the topological features and just make predictions using the image data.

5 Conclusion and Future Directions

This paper presents the results of two experiments. First, we demonstrate in our toy dataset that CNNs fail to learn the topological features that can be extracted through persistent homology. This adds to the growing body of evidence that CNNs are unable to learn global features. Second, we demonstrate that contrary to what we initially hypothesized, these topological features are not always useful for image classification. For future directions, there are many ways to extend this project. We could attempt to search for a natural image data set for which using topological features actually improves classification accuracy. Alternatively, we could try to find topological features of images other than the ones obtained from persistent homology using the tools of topological data analysis. Thirdly, we could test the robustness of topological features in image data by attempting to construct adversarial examples.

Acknowledgments

We would like to thank Jean-Baptiste Tristan for some enlightening conversations regarding this project.

References

[1] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, “Intriguing properties of neural networks,” *arXiv preprint arXiv:1312.6199*, 2013.

[2] C. Tralie, N. Saul, and R. Bar-On, “Ripser.py: A lean persistent homology library for python,” *Journal of Open Source Software*, vol. 3, p. 925, Sept. 2018.

[3] F. Fernández-Navarro, “A generalized logistic link function for cumulative link models in ordinal regression,” *Neural Processing Letters*, vol. 46, pp. 251–269, Jan. 2017.

[4] P. Vsetecka and M. Saska, “Navigation and stabilization of swarms of micro aerial vehicles in complex environment,” May 2015.

[5] R. Geirhos, P. Rubisch, C. Michaelis, M. Bethge, F. A. Wichmann, and W. Brendel, “ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness,” *arXiv preprint arXiv:1811.12231*, 2018.

[6] R. Ghrist, “Barcodes: The persistent topology of data,” *Bulletin of the American Mathematical Society* ,vol. 45, pp. 61–76, Oct. 2007.

[7] S. Ghuffar, "Homology Groups and Persistence Homology", Jun. 2010.

[8] T. E. de Campos, B. R. Babu, and M. Varma, “Character recognition in natural images,” in *Proceedings of the International Conference on Computer Vision Theory and Applications*, Lisbon, Portugal, February 2009.

6 Appendix A: Persistent Homology

We give a slightly more mathematically rigorous development of persistent homology in this section, although a complete development is outside the scope of this paper (and is not necessary to understand our contributions). See (6) for a thorough exposition on the subject. The images in this section were gathered from sources (4) and (7). We assume some knowledge of linear algebra and basic geometry.

6.1 Simplicial Complexes and Simplicial Homology

Simplicial homology is a method of computing the number of holes in simplicial complexes, which are combinatorial objects that are built from gluing together triangles.

Definition 1 *A k-simplex is the convex hull of $k+1$ affinely independent points $x_0, \dots, x_k$ in $\mathbb{R}^k$. Affine independence means that the vectors $x_1 - x_0, \dots, x_k - x_0$ are linearly independent.*

For example, Figure 8 shows k-simplices for $k \in \{0, 1, 2, 3\}$, we refer the the convex hull of $k+1$ affinely independent points as a k-simplex. We will denote a k-simplex σ with vertices $\{v_0, \dots, v_k\}$

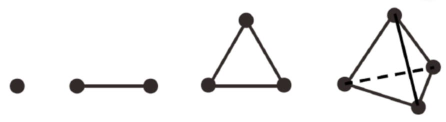


Figure 8: Simplices of dimension $0-4$

as $\sigma = [v_0, \dots, v_k]$. A *face* of a simplex is the convex hull of some subset of its vertices. We let $f_i(\sigma)$ denote the face of σ obtained as the convex hull of the points $v_1, \dots, v_{i-1}, v_{i+1}, \dots, v_n$ for $i \in \{0, \dots, n\}$.

Definition 2 *A simplicial complex S is a set of simplices such that every face of every simplex in S is also in S and the intersection of any two simplices a, b in S is either disjoint or a face of both a and b.*

Intuitively, we construct simplicial complexes by gluing together simplices along their faces. Figure 9 shows a valid and invalid example of this construction - the example on the right is invalid because the simplices are not glued together along a shared face.

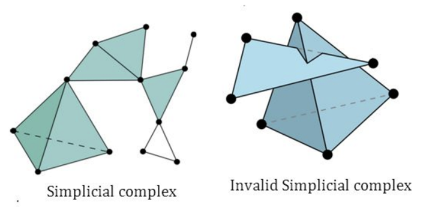


Figure 9: Example and Non-example of Simplicial Complexes

Simplicial homology is a method of computing the topological features (number of holes of varying dimension) of a simplicial complex - it is a special case of a more general construction of homology that computes topological features of a larger class of spaces. We fix a simplicial complex X and a field F. The point of simplicial homology is to associate a vector space $H_k(X)$ over F such that dimension of $H_k(X)$ counts the number of k-dimensional “holes" in X, where a 0-dimensional holes is a connected component, a 1-dimensional hole is a cycle which is not the boundary of a 2-dimensional region, etc.

We now setup the construction of the simplicial homology space $H_k(X)$. Let $C_n(X)$ denote the free vector space over F generated by the set of n-simplices of X for $n \in \mathbb{N}$.

Definition 3 *Let $n \in \mathbb{N}$ and fix an n-simplex σ in X. The n^{th} boundary map $\partial_n : C_n(X) \to C_{n-1}(X)$ acts on basis vectors by sending $\partial_n : \sigma \mapsto \sum_{i=0}^{n} (-1)^i f_i(\sigma)$. The map is extended linearly to the vector space $C_n(X)$.*

The boundary map sends each k-simplex to the (formal) alternating sum of its $(k-1)$-dimensional faces. For example, if $\sigma = [v_0, v_1]$ is a line segment, then $\partial_1(\sigma) = v_1 - v_0$. Intuitively, vectors that are in the the kernel of ∂_i can be thought of as "i-dimensional cycles". For instance if v_0, v_1, and v_2 are the vertices of the 2-simplex Δ^2, then the vector $v = [v_0, v_1] + [v_1, v_2] + [v_2, v_0] \in C_1(X)$, which corresponds to tracing the outside of Δ^2, satisfies

$$\begin{aligned} \partial_1(v) &= \partial_1([v_0, v_1]) + \partial_1([v_1, v_2]) + \partial_1([v_2, v_0]) \\ &= v_1 - v_0 + v_2 - v_1 + v_0 - v_2 = 0, \end{aligned}$$

so $v \in \ker \partial_1$. The simplicial homology spaces record cycles which do *not* correspond to the boundary of a higher-dimensional region, i.e holes. It is a fact that for each index k, we have that $\partial_{k+1} \circ \partial_k = 0$ (this can be verified by a direct computation), i.e $\operatorname{im} \partial_{k+1}$ is a subspace of $\ker \partial_k$. This enables the following definition of simplicial homology.

Definition 4 *The k^{th} simplicial homology group $H_k(X)$ of a simplicial complex X is defined as the quotient space $H_k(X) = \ker \partial_k / \operatorname{im} \partial_{k+1}$.*

The dimension of $H_k(X)$ (over the chosen ground field F) corresponds to the number of k-dimensional holes in X and is known as the k^{th} *Betti number* of X. Practically, the dimensions of the homology spaces can be computed using algorithms for finding the ranks and nullities of the boundary maps ∂_k. For example, computing the simplicial homology of the boundary of the 2-simplex (which is topologically equivalent to a circle), we get that $\dim H_i(X) = 1$ for $i = 0, 1$ and $\dim H_i(X) = 0$ for $i \geq 2$. This matches our intuition as a circle has one hole, one connected component, and no higher dimensional features. If we instead considered the "filled-in" 2-simplex, all of its Betti numbers would be 0 except for a 1 at dimension 0.

6.2 Persistent Homology and Persistence Diagrams

Suppose that we have an n-dimensional point cloud D sampled from a manifold $X \subset \mathbb{R}^n$. We would like to infer information about the homology groups of X with coefficients in a field F from the data D. Persistent homology relies on computing the simplicial homology of a filtered simplicial complex built from D called the *Vietoris-Rips complex.*

Definition 5 *Fix a parameter $\epsilon > 0$ and fix a point cloud $D \subset \mathbb{R}^n$. The Vietoris-Rips complex $R_\epsilon(D)$ is defined as the simplicial complex whose p-simplices are the convex hulls of collections of points C such that each pair of points $x, y \in C$ satisfies $d(x, y) < \epsilon$, where d is the usual Euclidean distance.*

Given a set of points D in $\mathbb{R}^n$ and $\epsilon > 0$, the simplicial complex $V_\epsilon(D)$ is constructed by connecting points that are no more than ϵ apart and then filling in a simplex whenever each face of the simplex is included in the complex. Figure 10 shows this.

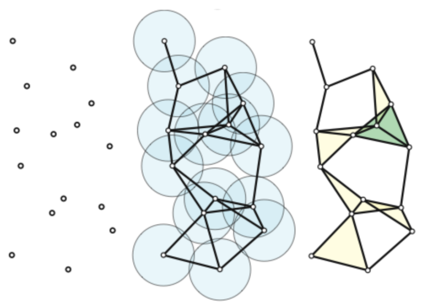


Figure 10: Vietoris-Rips Complex

Note that if $\epsilon \leq \epsilon'$, then $R_\epsilon(D) \subset R_{\epsilon'}(D)$. Thus as we vary ϵ over the positive reals, we obtain a continuous family of nested simplicial complexes $\{V_\epsilon(D)\}_{\epsilon > 0}$ called a *filtration.* Persistent homology records when topological features are created and destroyed across the $V_\epsilon(D)$'s as ϵ is varied smoothly across the positive reals. For instance, when ϵ is less than the minimum distance between any points of D, $R_\epsilon(D)$ is just the discrete set of points D, so many connected components are created. As ϵ increases, more points get connected and eventually 2-simplices and higher dimensional simplices will get added to the complex. This leads to components merging together, with certain components getting destroyed (the convention is the component that was created first survives when two components merge together at some stage of the filtration). An example filtration is shown in Figure 11, with parameters $0 < \epsilon_1 < \epsilon_2 < \epsilon_3$.

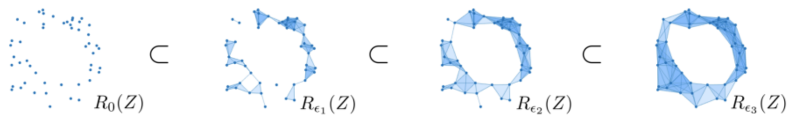


Figure 11: Stages of Vietoris-Rips Filtration of Point Cloud $Z \subset \mathbb{R}^2$

In Figure 11, we start with a discrete set of points sampled from an annulus. As ϵ increases, the corresponding Vietoris-Rips complexes become increasingly connected until eventually there is only one connected component and one hole (in $R_{\epsilon_3}(Z)$). The connected components at the first stage are almost all quickly merged with/destroyed by other components as the filtration progresses - these represent noisy features, while the singular connected component that emerges at the end represents the most significant 0-dimensional feature of the space. The same can be said for the the significant 1-dimensional feature surviving until the end of the filtration corresponding to the most significant hole in the data sample.

Persistent homology records the values of the parameter ϵ at which each feature is created and destroyed, so that one can get a sense of which features are topologically significant (ones that live longer) versus which features are just noise (those with short lifetimes). A convenient way to represent this information is a *persistence diagram*, which for each dimension k gives a collection of points (c_i, d_i) indicating that a k-dimensional topological feature was created at stage c_i in the Vietoris-Rips filtration and was destroyed at stage d_i - points close to the diagonal represent noise while those far from the diagonal and closer to the vertical axis represent significant topological features. Figures 1 and 2 show an example point cloud and the corresponding persistence diagram to illustrate this concept. The actual computation of when topological features are created/destroyed involves computing the images of linear maps between simplicial homology spaces of different complexes in the filtration, which we omit from this appendix for sake of brevity.