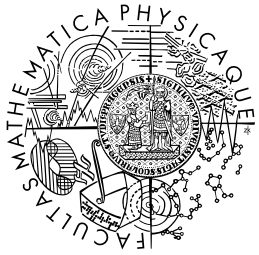

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Bc. Marek Černý

Persistent Homology and Neural Networks

Computer Science Institute of Charles University

Supervisor of the master thesis: doc. Mgr. Robert Šámal, Ph.D.
Consultant: M.Sc. Bastian Alexander Rieck, Ph.D.
Study programme: Computer Science
Study branch: Discrete Models and Algorithms

Prague 2021

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

I would like to thank my advisor Robert Šámal for teaching me great subjects and his advice during the assembly of my findings into this thesis. As well, I would like to thank my consultant Bastian Rieck for his open arms, discussions, and impulses leading to the genesis of this work.

Title: Persistent Homology and Neural Networks

Author: Bc. Marek Černý

Institute: Computer Science Institute of Charles University

Supervisor: doc. Mgr. Robert Šámal, Ph.D., Computer Science Institute of Charles University

Consultant: M.Sc. Bastian Alexander Rieck, Ph.D., Machine Learning and Computational Biology Lab, ETH Zurich

Abstract: Deep learning has solved various computer vision problems in the last decade. We use computational topology, namely persistent homology groups, to analyze the spaces of internal feature representations of convolutional neural networks (CNNs). We observed the correspondence between the summaries of the homological persistence groups of the decision boundary and ResNet-18 CNN training error during the phenomenon of deep double descent. Furthermore, a considered specific persistence summary suggests an analogy to epoch-wise double descent so that we may better understand the internal representations during the critically parametrized regime.

Mainly, we develop and compare multiple approaches to CNN models processing using the persistence homology. Our best method, the Differentiable Persistence Accuracy Estimator (DPAE), achieves very high accuracy in predicting CNN's performance (R2 score close to 0.99). Our DPAE is a topologically optimized end-to-end architecture involving differentiable persistence computation. Moreover, DPAE overperforms the original classical machine learning-based method on its native Small CNN Zoo dataset. We publicly release the complete source code of DPAE and other presented experiments.

Keywords: topological methods, computational homology, convolutional neural networks, double descent, accuracy prediction, differentiability, persistence

Contents

Introduction **3**

1 Deep Learning Background **6**
1.1 Neural Networks 6
1.1.1 Feed-forward Neural Networks 6
1.1.2 Supervised Training 8
1.2 More on Neural Networks 11
1.2.1 Convolutional Neural Networks 11
1.2.2 Random initialization 11
1.2.3 Strategies of Stochastic Optimization 12
1.2.4 Regularization strategies 13
1.2.5 Datasets 14

2 Computational Homology **16**
2.1 Paths in Topological Spaces 16
2.2 Simplicial Homology 18
2.2.1 Simplicial Complexes 19
2.2.2 Homology Groups 20
2.2.3 Coverings and Nerves 21
2.3 Persistent Homology 23
2.3.1 Filtration 23
2.3.2 Persistent Homology Groups 23
2.3.3 Computation 25

3 Dataset-based Analysis **28**
3.1 Notion of Datasets Topology 28
3.1.1 Filtrations for Labeled Spaces 28
3.1.2 Synthetic and Real-World Problems 29
3.2 Gradual simplification of topology 30
3.3 Homology during Deep Double Descent 32
3.3.1 Our experiments 33
3.4 Discussion 34

4 Data-free Topological Accuracy Prediction **37**
4.1 Related Work 37
4.2 Methodology 38
4.2.1 Predicting from Neural Network Model 39
4.2.2 The Small CNN Zoo dataset 40
4.2.3 Neural Classification of Persistence 40
4.3 Model Filtrations 41
4.3.1 Random Space Filtration 41
4.3.2 Computation Tree Filtration 41
4.3.3 Filtration Of Function Graph 43
4.4 Differentiable Persistence 43
4.4.1 Architecture 44

4.5 Experiments 45
4.5.1 Implementation 47
4.6 Discussion 50

Conclusion **52**

Bibliography **54**

List of Figures **58**

List of Tables **59**

A Attachments **60**
A.1 Double Descent: Topological Summaries 60
A.2 Data-free DPAE Visualization. 61

Introduction

Broadly speaking, geometry has attracted humankind's attention for thousands of years. For instance, the geometry of caves, castles, buildings, or general constructions around us essentially determines their mechanical and static properties. However, geometry itself does not carry the conscious meaning exclusively. In the context of buildings architecture, Earth's gravity is the primary *interpreter* determining whenever the construction *works* or *how well* it works under certain conditions. Indeed, architectural geometry on a minor planet or even in space might be completely different. Moreover, there are examples where the global properties are in the center of meaning instead of local geometry. Consider classical geographical terms, such as peninsula and island, river and lake. Now we more emphasize the *topological properties* such as being *connected*. One of the interpreters could be heat diffusion in oceans, where streams cannot cross the connected lands.

Although we want to observe only the topology of objects similar to connected lands and oceans in our metaphor, we deal with two main challenges. First, it can be unfeasible to compute all topological properties. Thus, we limit the characterization to qualitative topological summaries such as connected components (continents, islands), circles (isolated lakes and seas), voids (underground lakes), and higher-dimensional generalizations. In mathematics, we refer to these summaries as homology of an object, a space. Second, following our metaphor, for *some reason*, we do not know the *sea level* in prior. For example, if there is no water on our planet, there would be only one connected land, and increasing the sea level, some islands would disappear, so there would be no land in the end (Figure 1). Like in the case of some geometric sweeping-line algorithms, an island's creation or disappearance is a discrete event, and we can compute results for all values of the sea level parameter. The recording of homology that changes in steps along with the parameter, we call *persistent homology*.

Over the last decade, people *solved* an unbelievably huge number of problems via deep neural networks gradient-based optimization, referred to as *deep learning*. From computer vision and natural language processing to control and planning, deep learning slowly replaces humans, especially in the case of performing repetitive operations. In this thesis, our attention is attracted by the geometry of the internal representations in neural networks. We emphasize that this geometry also does not stand alone: it is interpreted by the inference through the

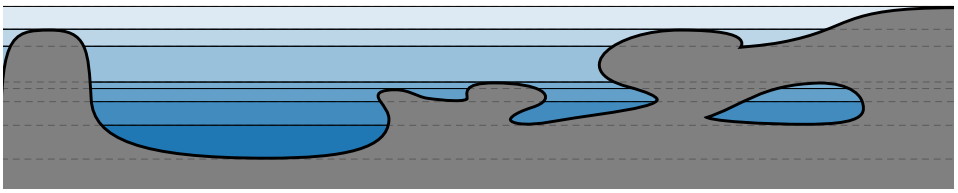

Figure 1: **Increasing sea level.** Every step of the color gradient signalizes one or more water surfaces creation/merge/disappearance.

network's structure and the backpropagation algorithm optimizing it. We wonder whenever the geometry of representation may have more general quantitative properties that can be observed through the lens of persistent homology.

Problem Statement

As mentioned, the principle of deep learning is widely applied. Nevertheless, although the network performs excellently on specific inputs, there are typically no theoretical guarantees on the model's accuracy. Motivated by this success, it is clear that minor but easily applicable breakthroughs can impact the whole industry. One such example is gradient clipping for neural cell models [2012] in natural language processing.

Novel neural models are developed and trained, but we typically only monitor standard statistics such as mean, variance, and quantiles of the internal representations. Typical characteristics of such models are that we can stratify them into layers, differentiable operations between high-dimensional spaces of the internal representations. These spaces have some specificities, such as being partitioned into classes. Topological characteristics may express new unexplored information about the shape of the subspaces of representations or manifolds corresponding to classes of categories. We want to adjust the existing method of computational topology, persistent homology to learn about the properties of the models having an actual impact area such as computer vision.

Next, we identified the potential for directly analyzing the convolutional neural network models without prior knowledge of the underlying dataset. Such type approach has broad applicability; however, there is no standardly used method of networks model analysis using the persistent homology. Moreover, descriptivity and interpretation of homology-based results arising from a potential method are firmly questionable; for example, a network model might induce a non-Euclidian space. Therefore we investigate or develop multiple methods, put them in an appropriate comparison. Nevertheless, we use deep learning itself to read the information in the topological description provided by persistent homology correctly and possibly optimally. Worth a notice, we can be optimistic about the quality of readout by deep neural networks since they are the universal approximators.

Contributions

First of all, this thesis should help readers seeking the intersection of topology and deep learning. Luckily, we have not solved everything in the scope of our thesis. Our findings rather raise new questions for further research!

We empirically relate the homology of boundary manifold to a neural training phenomenon known as double descent in Chapter 3. We observe the stability of persistent homology features during training the ResNet of multiple widths. Also, we found out that the summaries of 1-dimensional persistent homology groups empirically correspond directly to the network accuracy.

The core contribution is given in Chapter 4. We compare four dataset-free ways of constructing complex filtration on real-world networks. As the fourth one, we propose the *Differentiable Persistence Accuracy Estimator* (DPAE). We measure a state-of-the-art performance in predicting the accuracy of small convolutional networks trained on CIFAR-10 without knowing the training/validation data. Furthermore, we observe topological patches, surprisingly supporting the previous findings.

Outline

- Chapter 1 summarizes the underlying deep learning preliminaries. Terms and notation referred to in the next chapters are emphasized. It briefly introduces feed-forward networks, convolutional neural networks. It summarizes random initialization techniques, stochastic optimization methods, and corresponding regularization. It gives the necessary overview and sample of traditional datasets.
- Chapter 2 ensures the reader understands the relation between topological spaces and their functors of algebraic topology based on generalized continuous path maps. It states the central notion of computational topology, which is simplicial homology.
- Chapter 3 refers to a wide range of aspects of persistence-based analysis, such as simplifying spaces of multiple co-embedded manifolds. It gives an experiment visualizing the persistence summaries in the context of an essential phenomenon of deep learning, double descent.
- Chapter 4 provides insights beyond the limits of homological descriptivity. In contrast to Chapter 3, it aims to mine the information from the neural models' structure without knowing the data distribution. Since it proposes a new topological machine-learning pipeline, it needs to make sure that the results are not mainly influenced by other than topological statistics. Therefore it compares to another existing statistics-based method on its native independent dataset.

1. Deep Learning Background

In this chapter, we start with an introduction to multi-layer neural networks and their training procedures. The following part extends to convolutional neural networks (CNNs). After defining the CNN model structure, we describe details of initialization, training, and evaluation metrics. Finally, we overview typical datasets.

Convolutional networks are self-standing objects that we intend to use as input data for the topological analysis. However, various improvements and combinations with other related models exist in practice. Thus we finally list examples of such deep convolutional architectures. On the other hand, to put the reader into a broader context, Deep Learning Book, Goodfellow et al. [2016], covers the topic starting from Linear Algebra, Information Theory, and Machine Learning.

1.1 Neural Networks

1.1.1 Feed-forward Neural Networks

The predecessor of modern deep networks is a *feed-forward neural network* developed in the mid-1980s. Networks from that time have multiple units. Each unit has its own parameter set called weights, and then it passes a weighted sum of inputs through a scalar addition and an *activation function* on the output. Importantly, units can be connected using the output of one as an input for another.

Layers. The keyword *feed-forward* enforces that any of the unit inputs does not depend circularly on that unit's output. We can thus stratify neural networks into *layers* of units by the distance from networks input. The last layer of a network is usually called *output layer*, we refer to other layers as *hidden layers*. See Figure 1.1.

Notably, the feed-forward neural network with one hidden layer can approximate every function. Informally, by the *Universal Approximation Theorem* [1989, 1999] it is sufficient to use enough, possibly exponentially many, units of a hidden layer followed by a fixed non-polynomial activation function.

Forward computation. For a given network unit, we can express the sum of the input vector $x \in \mathbb{R}^n$ weighted by $w \in \mathbb{R}^n$ as a dot product $w \cdot x$. Indeed, the computation of one hidden layer having m units can by expressed as an affine transformation passed through activation function

$$y = f(Wx + b), \tag{1.1}$$

where $y \in \mathbb{R}^m$, $W \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, and $f \in \mathcal{C}(\mathbb{R})$. Notice that we apply the activation function f element-wise so that we keep outputs of the layer independent. The matrix W is called *weight*, and b the *bias* of the layer.

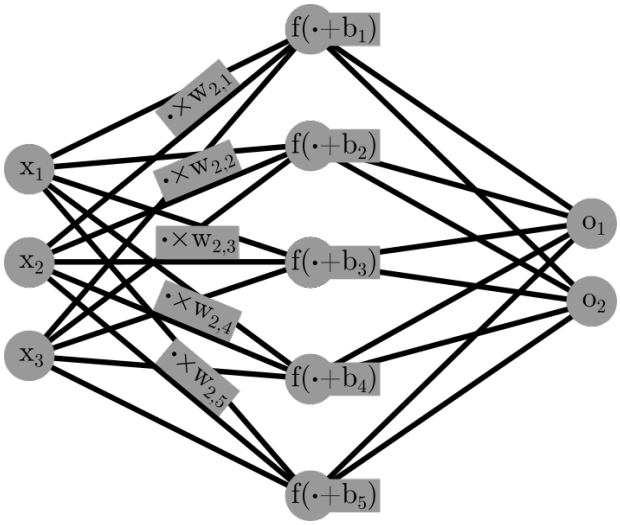

Figure 1.1: **Simple neural network.** Network of two layers. The first one is labeled: input vector $x \in \mathbb{R}^3$, weights $W \in \mathbb{R}^{3\times5}$, bias $b \in \mathbb{R}^5$. Notice that in the second layer is the second corresponding weight matrix in $\mathbb{R}^{5\times2}$, bias vector in $\mathbb{R}^2$, and activation function f that results in the output units' vector $o \in \mathbb{R}^2$.

Activation functions. An activation function is a component giving the neural network its expressive ability to approximate functions. People often refer to activation function as a non-linearity. The name is motivated by the fact that otherwise, the network would compute a composition of affine transformations, thus a single affine transformation. When choosing an activation function, we often think about non-linearities in two slightly distinct ways.

First, *activations in hidden layers* have an impact on internal representations. From a computational perspective, researchers want to reach the sweet spot between an increasing number of units with faster-evaluated activations and an increasing expressivity of a single unit. The main representative of faster-evaluated activations is ReLU, *rectified-linear unit*,

$$x \mapsto \max(0, x), \tag{1.2}$$

which returns the positive part of a number. A less radical variant of rectified-linear unit is called *leaky ReLU* defined by $x \mapsto \max(0.1x, x)$. *Hyperbolic tangent* is the typical example of a slightly more involved function

$$x \mapsto \tanh(x). \tag{1.3}$$

Output functions. In the second case, we select the activation function for the output layer. The range of the selected function then determines the range of our network. The output shape is purpose-driven. For specific problems, we wish to predict value from interval $(0, 1)$ for which the output layer contains a

single unit with *sigmoid*, or logistic, activation

$$x \mapsto \frac{1}{1+e^{-x}}. \tag{1.4}$$

Other times, we may need to estimate a categorical distribution. In other words, we want to predict a non-negative vector having the sum of its coordinates one. We construct the output layer with the unit for each vector coordinate. The value normalization does the *softmax* function

$$v_i \mapsto \frac{e^{v_i}}{\sum_{j=1}^{n} e^{v_j}} \text{ for } v_i \text{ of } (v_1, v_2, \ldots, v_n)^\top. \tag{1.5}$$

Softmax differs from other presented activation functions. It operates on vectors, complete layer output, instead of on scalar values, single units.

1.1.2 Supervised Training

Supervised training uses a set of input samples assigned to ground-truth output values,

$$D = \{(x_1, y_1), (x_2, y_1), \ldots, (x_d, y_d)\} \tag{1.6}$$

where $d \in \mathbb{N}$, called dataset. The output values y_i we call *labels*. In machine learning generally, we divide our data into two parts. The first part is used by the network for the *training*, optimization of network parameters, and the second part for *testing* so that the network is evaluated on data that have not been seen before. The sizes of training and test we denote as the *split ratio* having typical values 50/50, 70/30, or 90/10.

In addition, people want to build a new network or modify the network's hyperparameters. During model selection, one picks the model with the best performance on the test set. However, these decisions can undirectly adjust the model for the concrete test set; therefore, one needs to create a new subset of data. The new third set called *validation set* then is independent of the model selection bias and gives better information about the performance on the unseen samples.

Loss surface. Let us for a while model the neural network independently on its structure. We can say that the network is a function

$$F \colon \mathbb{R}^n \times \mathbb{R}^p \to \mathbb{R}^k, \tag{1.7}$$

where n is the input size, p is the dimension of real-valued parameters vector θ, and k is the output dimension. The loss function is a criterion which the network should minimize on the training data. Formally,

$$\mathcal{L} \colon \mathbb{R}^p \times \mathbb{R}^k \times \mathbb{R}^k \to \mathbb{R}, \tag{1.8}$$

the loss function $\mathcal{L}$ takes parameter vector θ, networks prediction, and the ground truth label from the dataset. It can be seen as functional, taking the network F as the first parameter, but that is unnecessary in the scope of this work. Properties a function $\mathcal{L}$ determine the p-dimensional loss surface,

$$J(\theta) = \frac{1}{d_T} \sum_{(x,y)\in D_T} \mathcal{L}(F(x,\theta), y), \tag{1.9}$$

where we denote by D_T the training subset of D of the size d_T. Indeed, the optimization performs a series of descending steps on this surface until it reaches a (local) minima.

Loss functions. We give two examples of loss functions, each for the previously described output function. Firstly, the cost of the regression task having only one unit at the output and sigmoid activation we often estimate by *(mean) square error*, or *MSE*,

$$(\theta, \bar{y}, y) \mapsto (\bar{y} - y)^2, \tag{1.10}$$

where $(x, y) \in D$, and $\bar{y} = F(x, \theta)$.

Secondly, we recall that for classification to k categories, the distribution is a k-dimensional vector. The label $y \in \mathbb{R}^k$ is thus a vector having 1 on precisely one position. We need to evaluate the difference between predicted and ground-truth distribution. To address this requirement, we need to understand the maximum likelihood estimation (MLE). MLE is a more general concept, and the reader can find more information in [Goodfellow et al., 2016, page 34]. We call *cross-entropy*, the function derived from MLE,

$$(\theta, \bar{y}, y) \mapsto \sum_{i=1}^{k} -y_i \log(\bar{y}_i), \tag{1.11}$$

where $(x, y) \in D'$, and $\bar{y} = F(x, \theta)$.

Gradient backpropagation. Given a neural network model F, and loss function $\mathcal{L}$, we can write down the gradient of loss surface in θ. Point θ of the space of parameters of network F:

$$\nabla_\theta J(\theta). \tag{1.12}$$

The natural idea that follows is updating the position of the parameters θ by the ϵ-step in the opposite direction, $\theta - \epsilon \nabla_\theta J(\theta)$.

The feed-forward neural networks compose multiple hidden layers, and we obtained the mean loss of all training data samples. From the mathematical analysis perspective, we apply a chain rule along the topologically sorted nodes of the computation graph to get partial derivative value for each network parameter in θ. We call this *backpropagation* algorithm, or *backprop*. We illustrate with the formula of partial derivative with respect to the hidden layer parameter and

inputs. Let $a = Wx + b, y = f(a)$ be the equations computed by the hidden layer and assume we already know $\nabla_y J$, then

$$\nabla_b J = (\nabla_y J \cdot f'(a)) \cdot 1, \tag{1.13}$$
$$\nabla_W J = (\nabla_y J \cdot f'(a))x^\top, \tag{1.14}$$
$$\nabla_x J = W^\top (\nabla_y J \cdot f'(a)). \tag{1.15}$$

Gradient descent optimizer. As already noted, an obvious way of how we can greedily optimize, *train*, our network by performing the mean of all forward computations. Coefficient ϵ by which we multiply the updates, $w - \epsilon\nabla_w J$ of $w \in \theta$, we call the *learning rate*. However, computing the gradients for the whole D_T can be time-consuming since, for the larger d_T, we need to spend too much time on a single update of parameters θ. Therefore, there is an online version, *stochastic gradient descent* (*SGD*), that updates θ after a single training sample. SGD updates have a significant variance, so a lower learning rate ϵ could be necessary for the training convergence. Another more practical way is to compromise the former and the latter online optimization by updating θ after smaller sets of samples, *batches*. After the optimizer iterates over all the training data D_T, no matter if at once, one-by-one, or using batches, we say that we trained the network for one *epoch*. Usual training takes several epochs.

The batch size is a heavily studied *hyperparameter*, as well as the learning rate, network structure, or the choice of the loss function, and it impacts the final performance of the network. Nevertheless, we are more educated observers of a hyperparameter space than designers of new models, and thus we can skip more details on explanation now.

Evaluation The loss function can always give us information about the model quality when evaluating our model on the test set D_V of size d_V. However, since our test set is a discrete set of samples, not the distribution itself, we often want to measure concrete performance on these samples. For regression of the (probability) value, we can be interested in how well our predictor compares to (optimal) constant function. The R^2 *score* is defined by the ratio of mean square error of predictions and the variance of the ground-truth values

$$\mathrm{R}^2 = 1 - \frac{\sum_{(x,y)\in D_V}(F(x,\theta) - y)^2}{\sum_{(x,y)\in D_V}(\mu_{D_V} - y)^2}, \tag{1.16}$$

where $\mu_{D_V} = \frac{1}{d_V}\sum_{(x,y)\in D_V} y$. We use the R^2 score as an absolute measure since the MSE loss does not reflect enough the complexity of the problem. Its values are negative, or between 0 and 1; in the former case, the predictor performs worse than always optimal constant predictor. The latter is a relative performance scale, where the value close to 1 signalizes a strong performance.

For the classification task, we define the *accuracy* as the percentage of correctly classified samples using the highest argument value in the predicted distribution.

1.2 More on Neural Networks

1.2.1 Convolutional Neural Networks

In the previous section, we pointed out that the hidden-layer neural network can universally approximate. On the other hand, the cost of computation and requirements on the training set size limit the dimensionality of the input data. Consider a digitalized photography scaled to resolution 1024×1024 pixels, each pixel having three channels. Its size is too large to process with a wide enough fully connected neural network.

Hierarchical structure. One can observe that the features determining the captured objects are often local – contours, corners. Furthermore, photos often contain patterns independent of the precise position in the image. The intuitive solution of the former is to have only local connections between input image patches and the first hidden layer. For the latter, we want somehow to have the small local network copied multiple times. Convolutional layers address these requirements. Interestingly, they can be stacked vertically, creating a hierarchical feature extraction inside the network. Moreover, this concept showed so well-performing that several convolutional layers are sufficient to follow only by the output layer of the feed-forward network.

Convolution. We place parameters of the convolutional layer to a matrix $K \in \mathbb{R}^{k_w \times k_h \times c \times f}$, often called *kernel*, where $k_w, k_h \in \mathbb{N}$ denotes kernel's width, height; $c, f \in \mathbb{N}$ number of layer's input, and output channels, respectively. Next, let $I \in \mathbb{R}^{w \times h \times c}$ be the input for our layer having width w, height h. Finally, using brackets for indexing dimensions, we define the *convolution operation* on I as follows

$$(K * I)[i, j, o] = \sum_{m=1}^{k_h} \sum_{n=1}^{k_w} \sum_{l=1}^{c} I[i + m, j + n, l] \cdot K[m, n, l, o], \tag{1.17}$$

for i in $\{1, 2, \ldots, h\}$, j in $\{1, 2, \ldots, w\}$, and o in $\{1, 2, \ldots, f\}$.

1.2.2 Random initialization

We discussed how to perform inference and gradient backpropagation in the neural network instance. Nevertheless, what network parameters should we choose to start the first training epoch? There exist several criteria to be considered.

To start with, gradient-based optimization has no guarantees on reaching the global optima, making the starting point of the training influential. Observing the parameters of a single layer, having all the same value leads to getting the same backpropagated gradient, and thus the layer remains symmetrical. To address this issue, we often randomly choose values of the weight matrices, breaking the layer's symmetry. Consequently, the number of pattern combinations from the random initialization is exponential in the depth of the network, and interestingly, with the increasing width of the layers, these patterns are speeding up the overall training convergence from a certain threshold.

Unfortunately, the universally optimal answer to initialization is not known. Naturally, the initializer can generate random values using a distribution with μ and σ^2 parameters, such as *normal initialization*, or *truncated normal initialization*.

Exploding and vanishing. The reader experienced in numerical computation, i. e. for linear algebra, is not surprised that the vectors' norm impacts computation stability. In particular, chaining multiplication of tenths of matrices for millions of training steps affects gradients' size. When the weights are too small gradient can become insignificant, *vanish*, during the backpropagation. On the other hand, too large initialization leads to the exponential growth of gradients. Thus, we say that gradients are *exploding*.

The following approaches thus take into account the dimensions of the kernel K that has c input units and f output units. Glorot and Bengio [2010] showed theoretically for the networks without non-linearities and experimentally for *tanh* activations that taking elements from the uniform distribution,

$$k \sim U\left[-\sqrt{\frac{6}{c+f}}, \sqrt{\frac{6}{c+f}}\right] \text{ for } k \text{ element of } K, \tag{1.18}$$

keeps the same variance, $\sigma^2 = 2/(c+f)$, of the outputs, and gradients overall the network. The most used method for *ReLU* activated models was introduced by He et al. 2015. They suggest to sample from truncated normal distribution centered at $\mu = 0$ with the variance $\sigma^2 = 2/c$,

$$k \sim \mathcal{N}\left(0, \frac{2}{c}\right) \text{ for } k \text{ element of } K. \tag{1.19}$$

The last listed *orthogonal initialization*, Saxe et al. [2014], is a way how can we scale the outputs by choosing the weights of a two-dimensional square matrix. The matrix $A \in \mathbb{R}^{n \times n}$ is sampled from a normal distribution, and then the QR decomposition returns $A = QR$, where Q is the desired orthogonal matrix. Next, the scalar multiplication of Q matrix by a positive value, *gain*, does the intended scaling. In practice, we can reshape kernels from the orthogonal matrix of size $g \times g$, where $g = max(c, f)$, after removing the surplus rows/columns.

1.2.3 Strategies of Stochastic Optimization

More complex models, for example, compositions of branches of dense neural networks and convolutional networks, are sometimes hard to optimize. There exist modified optimizer extensions of SGD (Gradient descent optimizer 1.1.2) that can heuristically move over the saddle points and flat local minimum.

RMSProp. One example of an advanced optimizer is *RMSProp* (Tieleman and Hinton [2012]), whose name is a shortcut for the root mean square propagation. It extends the SGD with the exponential running mean correction to adapt the gradients norm for every parameter. The running average uses decay rate parameter β having 0.9 as its initial value.

Algorithm 1: RMSProp Optimization

Input : Neural network F with the initial parameters θ.
Params.: Learning rate ϵ, and decay $\beta = 0.9$, small constant $\delta = 10^{-6}$.
$r \leftarrow \vec{0}$
while *stopping criteria is not met* **do**
  Compute gradient g for minibatch from D_T
  $r \leftarrow \beta r + (1 - \beta)g^2$
  $\theta \leftarrow \theta - \frac{\epsilon}{\sqrt{\delta + r}} g$

Adam. *Adaptive momentum optimizer* (Kingma and Ba [2014]) is probably the most popular optimizer nowadays. The advantages are empirically faster convergence, stability of training, and lower sensitivity to the choice of learning rate. It extends the RMSProp by adding momentum for gradient computation. Adam also removes the accumulated biases of its moments by normalizing the power of momentum parameters. A disadvantage is that the optimizer needs to keep in memory two additional values for each trained parameter in θ instead of one in RMSProp or zero in SGD. Also, examples were found where the Adam optimizer overfitted more on the training data than other simpler optimizers.

Algorithm 2: Adam Optimization

Input : Neural network F with the initial parameters θ.
Params.: Learning rate ϵ, momentum decays $\beta_1 = 0.9$ and $\beta_2 = 0.999$, small constant $\delta = 10^{-6}$.
$r \leftarrow \vec{0}$
$t \leftarrow 0$
while *stopping criteria is not met* **do**
  $t \leftarrow t + 1$
  Compute gradient g for minibatch from D_T
  $s \leftarrow \beta_1 s + (1 - \beta_1)g$
  $r \leftarrow \beta_2 r + (1 - \beta_2)g^2$
  $\hat{s} \leftarrow \frac{s}{1-\beta_1^t}, \hat{r} \leftarrow \frac{r}{1-\beta_2^t}$
  $\theta \leftarrow \theta - \frac{\epsilon}{\sqrt{\delta + \hat{r}}} \hat{s}$

1.2.4 Regularization strategies

The model overfitting can appear when the networks have a small training dataset or when the training lasts many epochs. We describe the dropout and weight decay method, which we use extensively in this thesis. Note that this is not a complete list since other regularization techniques such as batch normalization, layer normalization, label smoothing are practical and widely used.

L_2 Regulatization. The *weight decay* or *L2 regularization* introduce additional term to the loss function. This term is multiplied by a small constant, usually around 10^{-5}. It takes all the weights from model params θ that are not the additional bias weights. The intuitive effect is that the less important parameters are getting smaller quickly. Moreover, weight decay probably leads to

Figure 1.2: **Examples by the dataset.** (a) MNIST dataset; (b) Fashion-MNIST dataset; (c) SVHN dataset; (d) CIFAR-10 dataset; and (e) CIFAR-100 dataset.

the effect that the data are seen having a higher variance by the network, making the overfitting more complicated. More rigorous reasoning for simpler linear regression can be found in Goodfellow et al. [2016].

Dropout. Consider, for example, a convolutional neural network that hierarchically creates features layer by layer. We can enforce a more robust decision if we mask out some parts of the feature during the training. Practically, we choose a *dropout rate* r, e. g. $r = 0.8$, and we mask out every input of the concrete neuron with the probability $1 - r$. We want to perform this operation only while we train our network. Thus we multiply the remaining inputs by $\frac{1}{1-r}$ to compensate for the higher (but usual) mean value of inputs during evaluation.

1.2.5 Datasets

Data-driven learning approaches solve many real-world problems nowadays. Thus there exist a significant amount of domain-specific datasets. In this section, we cover those smaller ones that appear in research papers most traditionally. A couple of them are maybe too historical from today's perspective. However, it appears helpful to run a high number of experiments with simpler models on smaller datasets to observe trends that can appear more generally in wider models and hopefully more complex data.

The following four datasets have small resolutions between 28 by 28 to 32 by 32 pixels, and each dataset has images of 10 categories. In this case, already been solved, but the usual task is to classify the image into the correct category. See Figure 1.2 for examples described bellow:

- **MNIST,** Modified NIST (Lecun et al. [1998]), is a database of handwritten digits. It consists of 60000 training samples and 10000 validation images.
- **Fashion-MNIST** introduced by Xiao et al. [2017], this dataset is intended

as a more complex drop-in replacement for the original MNIST. Having the same image sizes, data format and structure, allows us to further benchmark methods working well on MNIST.

- **SVHN** (the Street View House Numbers dataset Sermanet et al. [2012]) is a colored real-world set of cropped numbers from the Google Street View images.
- **CIFAR** dataset, introduced by Krizhevsky [2009], has two variants CIFAR-10, CIFAR-100 of 10, and 100 classes, respectively. Both consist of 60000 colored images of the resolution 32×32, displaying real-world objects such as airplanes, birds, dogs, frogs, or trucks. The CIFAR's complexity is relatively close to datasets of larger resolution.

Image-Net. ImageNet is a database of 14 million images proposed by Deng et al. [2009]. Its subset of 150,000 images divided into 1000 classes was used in the Large Scale Visual Recognition Challenge, organized annually between 2010 and 2017. This competition nicely illustrates the recent progress of Deep Learning since it ended due to convolutional neural network architectures decreasing the classification error to the labeling error of the dataset, and thus it was recognized as solved.

2. Computational Homology

This chapter introduces the reader to computational homology, a subfield of computational topology, a world where general topology meets computer science. Besides homology, this field also includes algorithmic aspects of the low-dimensional manifolds or the knot theory. We expect that the reader is familiar with basic terms of topology, such as topological space or continuous map. Otherwise, we suggest reading an introductory chapter from one of the following books. For purposes of this work and this chapter, we read, selected, fused, and adjusted the most relevant information from the article Topology and Data, G. Carlsson, 2009 and the following textbooks: Computational Topology: An Introduction, H. Edelsbrunner and J. Harer 2010; Algebraic Topology, A. Hatcher, 2002; and Mathematics++: Selected Topics Beyond the Basic Courses, I. Kantor, J. Matoušek, and R. Šámal 2015.

Fundamentally, topology conceptualizes neighborhoods, coverings, or continuous paths. The topological theory is based on these concepts and develops other properties of spaces such as separation axioms, compactness, or contractibility. One can observe that every listed object is by definition free of describing elements of the concrete topological space. Regardless of the description that indeed depends on the coordinates, geometric realization, dimensionality, or metric function, topology can abstract these qualitative properties independently of the concrete quantitative choices.

The above suggests that methods based on topology can be robust in situations where the coordinates of the data do not have a definable or unique meaning, but still, we want a tool to compare those spaces. The first two sections of this chapter summarize the theory that simplifies a topological space into a finite discrete combinatorial structure. On the other hand, in computer science, we usually do not need this simplification since we only can work with discrete samples of a topological space. Surprisingly, the Nerve Theorem states a relation between the coverings of our discrete samples and the combinatorial structure of the underlying space. Next, we define homology preserved by the noted relation and persistent homology that is not dependent on the covering scale. Finally, we learn how to represent the persistent homology properties and the space of its diagrams, and we define metrics under which persistence stabilizes small perturbations in the discrete data.

2.1 Paths in Topological Spaces

The nature of topological spaces is highly diverse. General topology uses separation axioms to distinguish properties of open sets for pairs of points in such spaces. However, we do not often meet a non-Hausdorff space or even a non-normal space in the computational world, so this property may not be discriminative enough. The next natural idea is to study the continuous maps between two spaces since the definition of continuity is the motivation behind the open sets in topology. To overcome the relativity of continuous maps, we fix the first well-known space, the

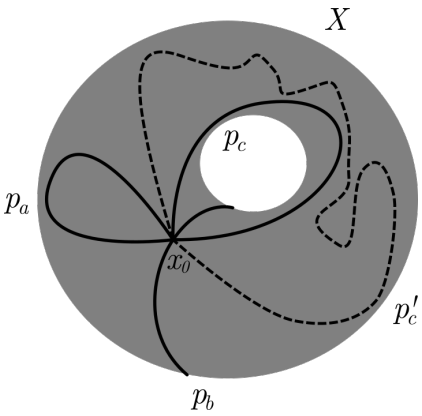

Figure 2.1: **Pointed torus.** Example of two-dimensional surface (X, x_0) embedded in $\mathbb{R}^3$. There are four loops, p_a, p_b, p_c, and p'_c, and every pair is homotopy non-equivalent, except for the last p_c and p'_c two homotopic paths. Since X is connected all pointed fundamental groups are equivalent, and thus $\pi_1(X) = \mathbb{Z} \times \mathbb{Z}$, this group is generated by p_b and p_c. Note that p_a is homotopy equivalent to a constant map to x_0, and thus both are lying in the same equivalence class, which is a neutral element of this group $\pi_1(X)$.

real closed interval between 0 and 1 usually, to study the structure of its images in the observed space.

Path connectedness. A significant example of continuous maps' utilization is the definition of sufficient a condition for spaces connectivity. We recall that the topological space is connected if we can partition it into at least two open sets. Let us consider the continuous map of a simple unit interval I to a topological space X, namely

$$p : I \to X, \text{ such that } a = p(0),\ b = p(1). \tag{2.1}$$

We call this map *path p from a to b* in X. We say, indeed, that X is *path-connected* if there is a path for each point to all other points. One can prove that path-connected spaces are connected. Two paths p, q can be joined when $p(1) = q(0)$ then we define the *composition* $p \circ q$ as a new path having p's domain linearly scaled on $[0, \frac{1}{2}] \subset I$ and q's domain linearly scaled on $[\frac{1}{2}, 1] \subset I$. Note that path p' defined by reversing the domain $t \mapsto 1 - t$ is starting and ending in the initial point.

Homotopy type. The number of possible paths in a topological space X is often uncountable. We now interrupt for a more general definition, which we will use to factorize the paths space. Given maps $f, g\colon X \to Y$, a continuous map $H : X \times I \to Y$ is called *homotopy* if $f = H(\cdot, 0)$, $g = H(\cdot, 1)$. Note that this relation is an equivalence, and we write $f \simeq g$.

Two topological spaces X and Y are said to be *homotopy equivalent* if there exist two continuous maps $f : X \to Y$ and $g : Y \to X$, such that $f \circ g \simeq \mathrm{Id}_Y$ and $g \circ f \simeq \mathrm{Id}_X$. Again, this relation is an equivalence and we extend our notation to spaces, $X \simeq Y$, moreover, we often say that X and Y are of the same *homotopy type*.

Fundamental group. When path p goes both from and to the same point $x_0 \in X$, we call p a *loop*. We can define a set of all loops pointed in x_0 denoted by $P(X, x_0)$. The set $P(X, x_0)$ forms a group together with composition operation, inverse as we defined on paths and neutral element $t \mapsto x_0$ for $t \in I$. By factorizing with the homotopy equivalece, requiring $H(0, \cdot) = H(1, \cdot) = x_0$, we get the *fundamental group* of X

$$\pi_1(X, x_0) = {}^{P(X, x_0)}\!/_{\simeq}\,. \tag{2.2}$$

For the path-connected topological space X, every two fundamental groups pointed in points x_0, y_0 from X are isomorphic. It follows from the fact that loops c in $P(X, x_0)$ are extensible with a fixed path p from x_0 to y_0 to be loops $p^{-1} \circ c \circ p$ in $P(X, y_0)$. We then apply the $\simeq$ equivalence. Once we have finished, we can denote the up-to-isomorphism unique fundamental group simply by $\pi_1(X)$. See Figure 2.1.

Functoriality. The structure of $\pi_1(X)$ describes the topological space X as a group. This group is not generally abelian. The notation suggests that π_1 is a kind of mapping. Since we require paths $p\colon I \to X$ to be continuous, we can compose them with continuous maps $f\colon X \to Y$ in the category of *pointed topological spaces* $\mathbf{Top}_\bullet$. Therefore, $f \circ p$ are paths in Y. And π_1 is a functor from pointed topological spaces to the category of groups,

$$\pi_1 \colon \mathbf{Top}_\bullet \to \mathbf{Grp}. \tag{2.3}$$

Since the non-abelian groups of π_1 are often too complex for computations, the question of abelianization of these groups by quoting by the commutant subgroup arises. We will see further that this has a simple interpretation. The second point is whether we can generalize $\pi_1(X)$ for higher dimensions. Taking the maps from n-dimesional cube I^n to X instead of the paths works, $\pi_n(X)$ then denotes *n-th homotopy group*. However, as expected, the complexity of π_n is quite non-straightforward, even for spheres. For those interested in following this tangent, we suggest the Hatcher's textbook of algebraic topology. In the following two sections, we define a class of functors $\mathbf{Top} \to \mathbf{Ab}$ that agrees with abelianizations of π_1 in dimension one. Nevertheless, the homological generalization is based on a different structure than n-cubes for the higher dimensions.

2.2 Simplicial Homology

There exist variants of homology group definitions. Besides the simplicial homology, one can name singular homology or cellular homology for CW-complexes. In the computational part, the simplicial variant is the most computationally effective. We start by introducing a foundational element of the simplicial homology, a combinatorial structure that we will use to approximate the topological space.

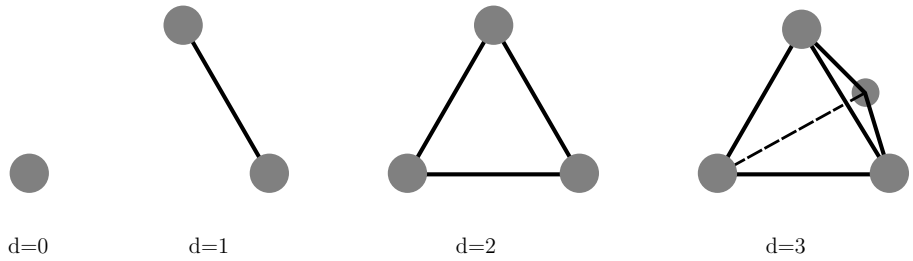

Figure 2.2: **Simplices.** Geometric simplices of dimension d.

2.2.1 Simplicial Complexes

Definition 1 (Abstract simplicial complex)**.** *Let V be a set. The finite collection of subsets $K \subseteq \mathcal{P}(V)$ is a* abstract simplicial complex *if for all $F \in K$ and all $E \subseteq F$ holds $E \in K$.*

The element F of K is *simplex*. Simplicies have assigned *dimension* by the cardinality, i.e., $n = |F| - 1$, allowing us to refer to F as *n-simplex*. We call the 0-simplices *vertices*.

Triangulation. The definition of simplicial triangulation of topological space has intermediate steps. We start with an abstract simplicial complex and realize it as a geometrical structure in some Euclidian space $\mathbb{R}^m$, and then we take the topological subspace of $\mathbb{R}^m$ used by the realized complex. If a topological space X is homeomorphic to this subspace, we consider it a triangulation.

Definition 2 (Geometric n-simplex)**.** *The* geometric n-simplex *is a convex hull of the set A of n affinely independent points in some $\mathbb{R}^m$. We call the convex hull of a subset of A a* face *of the concrete geometric simplex.*

The n-simplices proved to be less complex than the n-dimensional cubes, and importantly we can mix simplices in the abstract simplicial complex independently on the dimension. Topologically, the path is homeomorphic to 1-simplex, and the circle is homeomorphic to 2-simplex. To use mapping from the abstract simplicial complex, we need to realize this abstraction in the space. See Figure 2.2.

Definition 3 (Geometric simplicial complex)**.** *The* geometric simplicial complex *is a set Δ of geometric n-simplices for various n, if*

- *for all $\sigma \in \Delta$ and all ρ faces of σ holds $\rho \in \Delta$,*
- *and for all $\rho, \sigma \in \Delta$ the intersection $\rho \cap \sigma$ is a face of both ρ and σ.*

Definition 4 (Triangulation)**.** *Let $\Delta \subseteq \mathbb{R}^m$ be a geometric simplicial complex, let K be the corresponding abstract simplicial complex, and finally let X be a topological space. We say K is a* triangulation *of X if the topological subspace $\bigcup \Delta$ of $\mathbb{R}^m$ is homeomorphic to X.*

We call the topological space *trianguable* if it has at least one triangulation. It can be shown that every abstract simplicial complex has a realization by a geometric simplicial complex a suitable $\mathbb{R}^m$. However, as expected, not every topological space is triangulable.

2.2.2 Homology Groups

Suppose we have a triangulable topological space X. We split the assigned abstract simplicial complex K by dimension so that K_n contains n-simplices of K for all $n \geq 0$. Since K is closed on subsets by definition, there is a connection between K_n and K_{n-1}. All inclusions attaching $(n-1)$-simplices to n-simplices σ we can describe contravariantly by mapping

$$\sigma \mapsto \{\sigma \setminus \{v\}\}_{v \in \sigma}. \tag{2.4}$$

Chains. A simplex can be the face of multiple other simplices in a simplicial complex. Therefore we maybe want to count the simplices so that we can extend the map 2.4 to the complexes. For the counting, here we use $\mathbb{Z}_2$ coefficient over integers to slightly simplify the upcoming definitions and the future computations. Moreover, desired topological features will turn out to be independent of a chosen ring.

Definition 5 (Chain)**.** *Let $\{\sigma_1, \sigma_2, \dots \sigma_k\}$ be a set of n-simplices, the* n-chain *is a finite formal sum $\sum_{i=1}^{k} c_i \sigma_i$, where every c_i is in $\mathbb{Z}_2$.*

The set of all n-chains of simplices in K_n with the addition of formal sums is a group denoted by $C_n(K, \mathbb{Z}_2)$ and called the *chain group*. This chain group is a vector space over $\mathbb{Z}_2$, but we use the terminology of groups since for a general ring R, the $C_n(K, R)$ is a free R-module which is still sufficient to define simplicial homology. Because we work only with $\mathbb{Z}_2$ coefficients, we shortcut the notation of chain group to $C_n(K)$.

Definition 6 (Boundary homomorphism)**.** *Let $C_n(K)$ be a chain group for all n, then the* boundary homomorphism *is $\partial_n \colon C_n(K) \to C_{n-1}(K)$ determined by the mapping of simplices,*

$$\sigma \mapsto \sum_{v \in \sigma} \sigma \setminus \{v\}. \tag{2.5}$$

Theorem 1. *The composition of consecutive boundary homomorphisms vanishes,*

$$\partial_n \partial_{n+1} = 0 \quad \textit{for all } n \geq 0. \tag{2.6}$$

Proof. Take an $(n+1)$-simplex σ, then

$$\partial_n \partial_{n+1}(\sigma) = \partial_n \Big(\sum_{v \in \sigma} \sigma \setminus \{v\}\Big) = \sum_{\substack{v,w \in \sigma \\ v \neq w}} \sigma \setminus \{v, w\} = 0.$$

□

Theorem 1 gives us an essential property of chain groups. For $\sigma \in K_{n+1}$, it directly follows $\partial_{n+1}(\sigma)$ lies in $\operatorname{Ker} \partial_n$, or in other words, $\operatorname{Im} \partial_{n+1} \subseteq \operatorname{Ker} \partial_n$. Homology only studies this relation.

Remark. We can extract and further generalize the boundary map property to a *chain complexes* which is any sequence of pairs, $(C_k, \partial_k : C_k \to C_{k-1})_{k=0}^{\infty}$ such that every composition $\partial_k \partial_{k+1}$ vanishes.

Definition 7 (Homology group)**.** *For given two consecutive boundary homomorphisms,* $C_{n+1}(K) \xrightarrow{\partial_{n+1}} C_n(K) \xrightarrow{\partial_n} C_{n-1}(K)$*, we define the* n-th homology group *by*

$$H_n(K) = \operatorname{Ker} \partial_n \big/ \operatorname{Im} \partial_{n+1}. \tag{2.7}$$

Definition 8 (Betti number)**.** *Let* $H_n(K)$ *be a homology group, then the* betti number *denoted by* β_n *is the rank of group* $H_n(K)$*.*

Remark. Since our coefficients are from the field, the β_n corresponds precisely to the dimension of vector space $H_n(K, \mathbb{Z}_2)$.

The interpretation of β_n depends on the dimension n. The reader may already find out that β_0 counts the connected components.

dimension	0	1	2	n
interpretation	connected components	circles	voids/ cavities	n-dim. "holes"

Homotopy and homology. A nice aspect about homology is that it respects the homotopy types. Therefore, we can apply theorems about homotopy types directly to homology groups. The following theorem formalizes this idea.

Theorem 2 (Homotopy invariance.)**.** *If two maps* $g, g' \colon X \to Y$ *are homotopic, then they induce the same homomorphism* $g_* = g'_* \colon H_n(X) \to H_n(Y)$*.*

Proof. See [Hatcher, 2002, Section 2.1, Theorem 2.10, p. 111 – 114]. □

2.2.3 Coverings and Nerves

When we defined the simplicial homology, we assumed the scenario where we have given complete information about the topological space or at least about its triangulation. However, we often have only a set of finite observations of the space. Intuitively the observations carry approximate information about the connectivity of the space. In this subsection, we suppose having preliminary information that we observe a subspace of $\mathbb{R}^m$.

Definition 9 (Nerve)**.** *Let* $\mathcal{V}$ *be a finite collection of sets, then the* nerve *of* $\mathcal{V}$ *is defined by*

$$N(\mathcal{V}) = \{\mathcal{V}' \subseteq \mathcal{V} \colon \bigcap \mathcal{V}' \neq \emptyset\}. \tag{2.8}$$

We can check that the nerve $N(\mathcal{V})$ is always an abstract simplicial complex under the vertex set $\mathcal{V}$ since the property of having a non-empty intersection is closed under subsets. We continue with building complexes from the observations, and it will turn out that sufficiently refined covering agrees on the homotopy type (Section 2.1), and thus on the homology, with the original space.

Definition 10 (Čech complex)**.** *Let* $\epsilon > 0$*, and* V *be a subset of* $\mathbb{R}^m$*. Let us consider closed ball of diameter* ϵ*,* $B_v(\epsilon)$*, for all* $v \in V$*. Then the Čech complex is the nerve of the set of these balls, namely*

$$\check{C}(V, \epsilon) = N(\{B_v(\epsilon) \colon v \in V\}). \tag{2.9}$$

Theorem 3 (Nerve Theorem)**.** *Let $\mathcal{F}$ be a collection of closed and convex subsets of $\mathbb{R}^m$. Then its nerve is homotopic to its union,*

$$N(\mathcal{F}) \simeq \bigcup \mathcal{F}. \tag{2.10}$$

Proof. We omit the proof. □

Using the Nerve theorem, we plan to approximate space using Čech complex if someone gives us the parameter ϵ together with the sampled points. The definition of Čech complex as a nerve of closed balls is straightforward. However, the computation complexity grows exponentially since checking all the subsets asymptotically on the common intersection. Fortunately, even the Čech complex is approximable, for which we introduce another complex that has a quadratic computation cost instead.

Definition 11 (Vietoris-Rips complex)**.** *Once again, let $\epsilon > 0$, and V be a subset of $\mathbb{R}^m$, then the* Vietoris-Rips complex *is defined by*

$$VR(V, \epsilon) = \{V' \subseteq V : B_v(\epsilon) \cap B_w(\epsilon) \neq \emptyset \text{ for all } v, w \text{ in } V'\}. \tag{2.11}$$

Clearly, we can find an example in the plane where the two complexes do not agree, but we can bound the Vietoris-Rips using the Čech complex.

Theorem 4 (Vietoris-Rips Lemma)**.** *For every $\epsilon > 0$, and all V finite subsets of $\mathbb{R}^m$ holds*

$$\check{C}(V, \epsilon) \subseteq VR(V, \epsilon) \subseteq \check{C}(V, \sqrt{2}\epsilon).$$

Proof. The first inclusion comes directly from definitions. For the second inclusion, please see [Edelsbrunner and Harer, 2010, III.2, p. 74 – 75]. □

2.3 Persistent Homology

Persistence. In the namespace of computer science, this term usually emphasizes the extension of an existing object's functionality along a newly added axis. For example, data structures can become persistent in remembering their version before several consecutive operations with the structure in time.

When persisting homology, consider the situation we ended up in the last section: having samples from an unknown space and given the positive ϵ. Making our situation more realistic, suppose that ϵ is unknown. The central idea of persistent homology is to start with a very small initial ϵ_0, continuously increase it, and observe which chain classes persisting in the evolving homology groups. Similarly to the increasing sea level, we saw in Figure 1.

2.3.1 Filtration

Filtration of an abstract simplicial complex K is a sequence of inclusion-ordered simplicial complexes starting with the empty complex and ending with K,

$$\emptyset = K^{(0)} \subseteq K^{(1)} \subseteq \cdots \subseteq K^{(m)} = K, \tag{2.12}$$

and let us emphasize the corresponding inclusion maps explicitly:

$$s^{i,j} \colon K^{(i)} \hookrightarrow K^{(j)} \text{ whenever } 0 \leq i \leq j \leq m. \tag{2.13}$$

Creating filtration. There are variously generalized ways of creating filtration. Carlsson proposes the perspective of category theory by defining persistence objects on small partially ordered categories; Edelsbrunner and Harer show how we can generalize the Morse function for topological spaces to define filtration. We give one algebraic example. Let us consider a monotonous real-valued function f,

$$f \colon (K, \subseteq) \to (\mathbb{R}, \leq). \tag{2.14}$$

Now we invert the function f to the ordered set

$$(\{f^{-1}((-\infty, a]), a \in \mathbb{R}\}, \subseteq). \tag{2.15}$$

This set is of finite size m for the finite K. Since $f^{-1}((-\infty, a]) \subseteq f^{-1}((-\infty, b])$ if $a \leq b$ this set is a filtration of K of size m.

2.3.2 Persistent Homology Groups

We continue with a given filtered simplicial complex K of the space X and the inclusion $s^{i,j}$ for $0 \leq i \leq j \leq m$. We capture the structure of filtered chains with

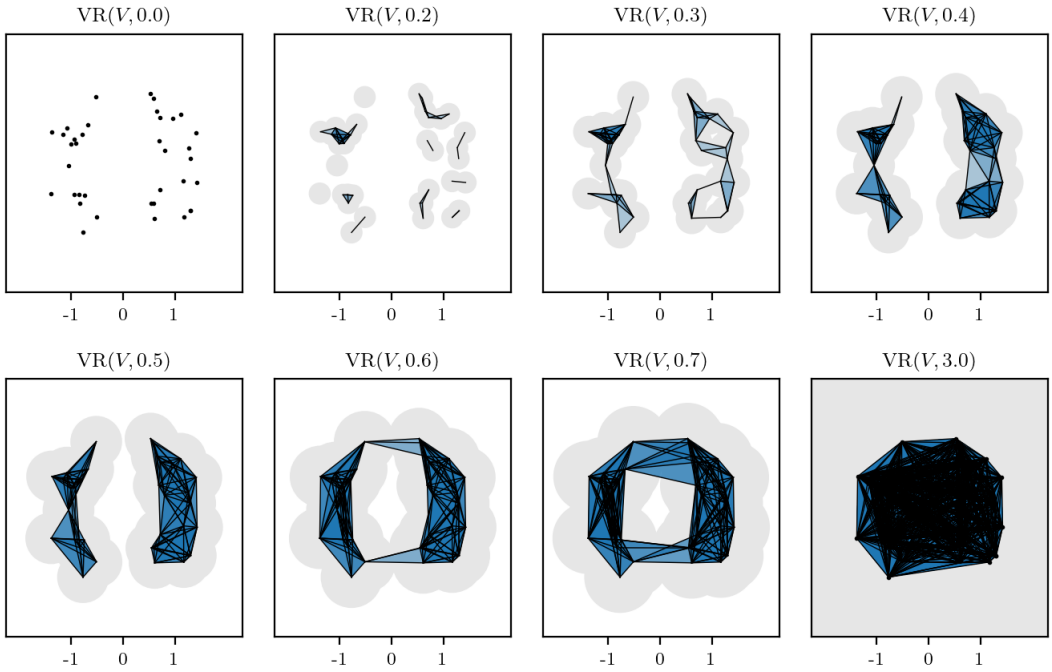

Figure 2.3: **Filtration Example.** To compute the persistent homology of a set V of points in $\mathbb{R}^2$, we construct the filtration of Vietoris-Rips complex K based on monotonous function $f(x) = \min_{y \in V, y \neq x} \|x - y\|_2$. We display simplices up to dimension 2.

the following matrix:

$$\begin{array}{ccccccc} C_{n+1}(K^{(0)}) & \xrightarrow{C_{n+1}(s^{0,1})} & C_{n+1}(K^{(1)}) & \xrightarrow{C_{n+1}(s^{1,2})} & \cdots & \longrightarrow & C_{n+1}(K^{(m)}) \\ \downarrow \partial_{n+1}^{(0)} & & \downarrow \partial_{n+1}^{(1)} & & & & \downarrow \partial_{n+1}^{(m)} \\ C_n(K^{(0)}) & \xrightarrow{C_n(s^{0,1})} & C_n(K^{(1)}) & \xrightarrow{C_n(s^{1,2})} & \cdots & \longrightarrow & C_n(K^{(m)}) \\ \downarrow \partial_n^{(0)} & & \downarrow \partial_n^{(1)} & & & & \downarrow \partial_n^{(m)} \\ C_{n-1}(K^{(0)}) & \xrightarrow{C_{n-1}(s^{0,1})} & C_{n-1}(K^{(1)}) & \xrightarrow{C_{n-1}(s^{1,2})} & \cdots & \longrightarrow & C_{n-1}(K^{(m)}) \end{array} \tag{2.16}$$

Each row of the above equation corresponds to a dimension, and each column is a step in the filtration. We can compute homology groups in each row independently so that the maps $s^{i,j}$ induce homomorphisms $s_n^{i,j}$ between every two homology groups. Notice that $s_n^{i,j}$ are not necessarily inclusions.

$$s_n^{i,j} = H_n(s^{i,j}) \colon H_n(K^{(i)}) \longrightarrow H_n(K^{(j)}) \tag{2.17}$$

Definition 12 (Persistent homology group)**.** *Let K be a simplicial complex and let $K^{(0)} \subseteq K^{(1)} \subseteq \cdots \subseteq K^{(m)}$ be its filtration, and $\{s^{i,j} \colon 0 \leq i \leq j \leq m\}$ a collection of corresponding inclusions. Then the* n-th persistent homology groups *are defined by*

$$H_n^{i,j}(K) = \operatorname{Im} s_n^{i,j} \quad \textit{for } 0 \leq i \leq j \leq m. \tag{2.18}$$

Once we check that for $c + \operatorname{Im} \partial_{n+1}^{(i)})$ in $H_n(K^{(i)})$ it holds $s_n^{i,j}(c + \operatorname{Im} \partial_{n+1}^{(i)})) = c + \operatorname{Im} \partial_{n+1}^{(j)})$ since $\operatorname{Im} \partial_{n+1}^{(i)} \subseteq \operatorname{Im} \partial_{n+1}^{(j)}$, we can conclude

$$\operatorname{Im} s_n^{i,j} = \operatorname{Ker} \partial_n^{(i)} \big/ \operatorname{Im} \partial_{n+1}^{(j)} \cap \operatorname{Ker} \partial_n^{(i)}. \tag{2.19}$$

Definition 13 (Persistent betti number)**.** *Let $H_n^{i,j}(K)$ be a persistent homology group, then the* persistent betti number *denoted by $\beta_n^{i,j}$ is the rank of group $H_n^{i,j}(K)$.*

The Equation 2.19 characterize the persistence homology groups. Easily follows that

- $H_n(K^{(i)}) = H_n^{i,i}(K)$, and, in addition, notice that
- $H_n^{i',j}(K)$ is a subgroup of $H_n^{i,j}(K)$ whenever $i' \leq i$.

Persistence as lifetime. Persistent homology can continuously connect Betti numbers by tracking classes of a homology group, giving insight into the evolving structure of the persistent homology along with the filtration. We say that an equivalence class γ_n of $H_n(K^{(i)})$ is *created* in $K^{(i)}$ if γ_n it is not present in $H_n^{i-1,i}(K)$. Once γ_n appears, we can track its lifetime in other persistent groups using $s_n^{i,j}$ for $j \geq i$. Unlike the creation of class, a destruction of class can be only be done by merging with an *older* class γ_n' created in $H_n(K^{(i')})$ for some $i' < i$, so that γ_n' is in $H_n^{i-1,i}(K)$. We say that our γ_n is *destroyed by entring* $K^{(j)}$, $j \geq i$, if $s_n^{i,j-1}(\gamma_n)$ is in $H_n^{i-1,j-1}$ and $s_n^{i,j}(\gamma_n)$ do not appear in $H_n^{i-1,j}$.

Definition 14 (Barcode)**.** *Let K be a simplicial complex filtered by $f\colon K \to \mathbb{R}$. For dimension $n \in \mathbb{N}$, let $\{H_n^{i,j}(K)\}_{i=0,j=0}^{m,m}$ be the corresponding persistent homology groups. The* barcode of dimension n *is the multiset of tuples in $\mathbb{R} \times (\mathbb{R} \cup \{+\infty\})$, where each tuple $(f(K^{(i)}), f(K^{(j-1)}))$ corresponds to an equivalence class γ created in $K^{(i)}$ and destroyed by entering $K^{(j)}$.*

2.3.3 Computation

Once we construct a complex such as Čech or Vietoris-Rips, we can define the boundary homomorphism, which is essential in the computation of the persistent homology group. We start with representing the boundary homomorphism as a matrix.

Definition 15 (Boundary matrix)**.** *Let $n \geq 1$, K be an abstract simplicial complex and its faces $\sigma_1, \sigma_2, \ldots, \sigma_p$ in a linear order that extends the inclusion. The* boundary matrix $A_\partial \in \mathbb{Z}_2^{p \times p}$ *is defined by*

$$A_\partial[i,j] = \begin{cases} 1 & \text{if } \sigma_i \subseteq \sigma_j \text{ and } |\sigma_i| + 1 = |\sigma_j| \\ 0 & \text{otherwise.} \end{cases}$$

One should not be surprised that, representing σ_i by unit vector e_i, its linear projection $A_\partial e_i$ yields the representation of the formal sum of its boundary $\partial(\sigma_i)$. Because of respecting the inclusion order, matrix A_∂ is *upper triangular.*

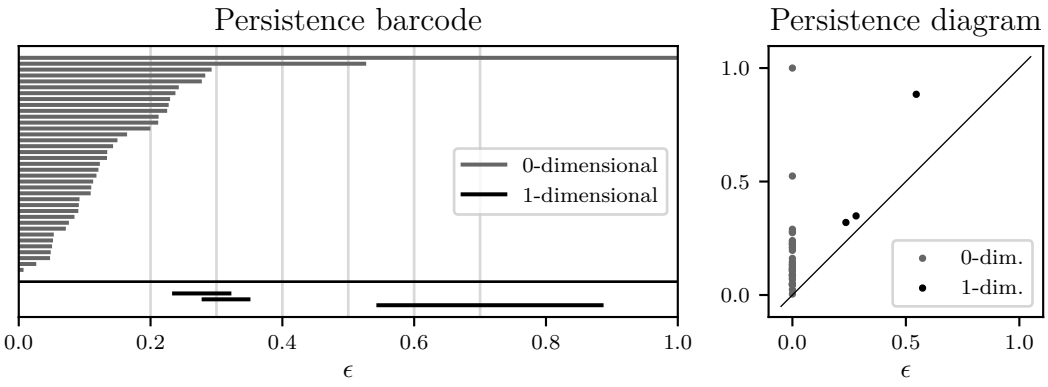

Figure 2.4: **Example of Persistent Homology.** We computed the persistent homology of points in $\mathbb{R}^2$ of the example we saw in Figure 2.3. The 0- and 1-dimensional barcodes correspond to the lifetime of connected components and circles, respectively. ϵ is the second parameter of VR. Left: Persistence barcode of the Vietoris-Rips persistent homology. Right: Equivalent representation of the computed persistence using $\mathrm{Dgm}_0(f)$ and $\mathrm{Dgm}_1(f)$.

Smith normal form. We noticed that the homology in general only requires a principal ideal domain (PID) ring R, unlike linear algebra, where the vectors are over a field. Indeed, we generally *do not need* the power of *Gaussian elimination* when dealing with formal sums instead of vectors. On the other hand, we allow the elementary *column* operations. For those who are interrested, let A be a matrix in $\mathbb{R}^{n\times m}$ with two invertible matrices S in $\mathbb{R}^{m\times m}$ and T in $\mathbb{R}^{n\times n}$ such that SAT has zeros outside the diagonal, and the diagonal is $a_1, a_2, \ldots, a_r, 0, 0, \ldots, 0$ such that a_i divides a_{i+1} for all $1 \leq i < r$. Than we call SAT the *Smith normal form (SNF)* of A.

The above-given definition (SNF) is too general for us since we deal with the upper triangular $\mathbb{Z}_2$-matrix A_∂. Interestingly, allowing only the addition of columns from left to right (meaning the column with the lower index to the columns with the higher index), we can read the complete information about the lifetime of persistence homology classes from this matrix. Before the reader see Algorithm 3, we denote the *maximal non-zero entry row index of column j* by

$$\mathrm{low}(j) = \max\{i : R[i,j] = 1\}. \tag{2.20}$$

Algorithm 3: A_∂ matrix reduction

Input : Matrix A_∂
Output: Matrix R in Smith normal form
$R \leftarrow A_\partial$ **for** $j \leftarrow 1$ *to* m **do**
  while *exists* $j' < j$ *such that* $\mathrm{low}(j') = \mathrm{low}(j)$ **do**
    for $i \leftarrow 1$ *to* n **do**
      $R[i,j] \leftarrow R[i,j] + R[i,j']$

The reduced matrix R can also be written in the form of $A_\partial T$, where T represents performed column operations. Moreover, since we added them exclusively from left to right, the columns and rows still correspond to simplices of K. We distinguish two cases. First, the column of index j of R is *zero*. In other words, it was entirely reduced by adding columns of indices 1 to $j-1$, and thus simplex σ_j is creating a new cycle corresponding to the *creation* of a new homology class. Second, the column of index j of R has at least one *non-zero* entry. In other words, its non-zero rows are the boundary of a chain (that is reflected in T), which means it corresponds to a cycle as well, but now it gives the *destruction* to one homology class. Most importantly, the dying homology class was born with the $\sigma_{\text{low}(j)}$ complex. Commented proof of this statement with an illustrated example reader can find in [Edelsbrunner and Harer, 2010, VII.1, p. 183 – 185].

Persistence diagrams. The reduced boundary matrix motivates a new representation of persistence, equivalent to the persistence barcodes. The n-dimensional *persistence diagram of function* f is a finite multiset of points from $\mathbb{R}\times(\mathbb{R}\cup\{+\infty\})$

$$\text{Dgm}_n(f) = \{(f(K^{(low(j))}), f(K^{(j-1)})) : 1 \leq j \leq p, |\sigma_j| = n+1\}. \tag{2.21}$$

For an example of persistence diagrams as well as computed persistent homology, see Figure .

3. Dataset-based Analysis

In supervised learning, labeled datasets usually represent a problem we model with a neural network. Computer vision problems can be defined so that all the data points, which means images, have a constant dimension: width, height, and the number of channels, meaning that we can naturally think of such a dataset as a finite subsampling of manifold embedded in a high-dimensional space. This chapter intends to give a better insight into how the networks transform the topology of such manifolds.

In recent years, a limited number of short studies observed dataset topology simplification by the layers of neural networks. These experiments together feel slightly isolated from each other and the other deep-learning studies as well. Therefore, we aim to put some parts of known experiments into the context of the topological analysis. We analyze the 18-layered ResNet convolutional network during multiple training regimes of the phenomenon called *double descent*, which plays a crucial role in understanding machine learning in general. Observations of such large-scale real-world behavior can potentially motivate new possible improvements. For example, Hofer et al. [2020] proposed a new state-of-the-art topological persistent homology motivated regularization that targets small (500 samples) training data regimes.

3.1 Notion of Datasets Topology

Classification. As we implicitly outlined in the second part of the first introductory chapter, we are mainly interested in the classification problem, since assigning images into a fixed number of known categories became the first domain where neural networks massively excelled, moreover, there exists a family of related problems such as feature extraction, segmentation, or object detection that repetitively integrates the famous classification-first architectures.

The first step of our analysis is to realize that even if we defined the homology of the whole spaces in the previous chapter, the situation slightly changes now. That is since the space, and consequently, the sampled dataset labels itself into classes regions, and we need to reflect this fact in our filtration construction.

3.1.1 Filtrations for Labeled Spaces

Let us first consider a space M consisting of a partition $M_0 \cup M_1$ and a binary-classifing neural network, a special kind of mapping, transforming M to $M' = M'_0 \cup M'_1$. Generally, we expect M to be significantly transformed, but one may retrieve more information by analyzing M_0 and M_1 separately. Also, since the network accepts all vectors of a fixed dimension on input, we need to handle the euclidian space embedding the manifold M. A reasonable option is to define $M_1 = \mathbb{R}^m \setminus M_0$, and consequently, investigate the topology of M_1, but there exist examples of simple spaces where this impacts the resulting information.

As we increase the number of classes from two, we can still analyze each class

separately or each pair separately of classes. On the other hand, there are not rare datasets with hundreds to thousands of classes (e.g., ImageNet in Section 1.2.5), making quadratically growing pairwise analysis unpractical. Following observation also is that if we have and topological feature, let us consider sphere–hole of dimension two–in $\mathbb{R}^3$ of a fixed class, then it is *significant* for the classification complexity if and only if it contains a data point from a distinct class. These issues are addressed by Ramamurthy et al. [2019] publishing a new notion of the complex for boundaries that we can formulate using the definition below.

Definition 16 (Partitioned Vietoris-Rips Complex)**.** *Let* $\epsilon > 0$ *and* $V_1 \sqcup V_2 \sqcup \cdots \sqcup V_n$ *be a partition of* V *a subset of* $\mathbb{R}^m$*. The* Partitioned Vietoris-Rips complex *is defined by*

$$VR(\{V_1, V_2, \ldots, V_n\}, \epsilon) = \{\{V_i' \subseteq V_i : 1 \leq i \leq n\} : \textit{satisfying (i) and (ii)}\}$$

where

(i) $B_u(\epsilon) \cap B_v(\epsilon) \neq \emptyset$ *for all* u *in* V_i'*,* v *in* V_j' *for some* i *and* j*,*

(ii) if $\{u, v\} \subseteq V_i'$ *then there is* w *in* V_k *such that* $B_w(\epsilon) \cap B_v(\epsilon) \neq \emptyset$ *and* $B_w(\epsilon) \cap B_u(\epsilon) \neq \emptyset$ *for* $i \neq k$*.*

For the defined *Partitioned Vietoris-Rips* filtration, we extended Definition 11 of *Vietoris-Rips* complex, by the condition *(ii)*, which allows connecting two points of the same class only after having at least one common neighbor of a different class. The resulting complex then captures the topological properties of a more abstract manifold but resolves all the dilemmas that come from a large number of classes as well as effectively reducing the information about the *unsignificant* holes.

3.1.2 Synthetic and Real-World Problems

Initially, while learning about the behavior of a new analytical method such as persistent homology, it can be pretty problematic to use real-world data as a starting point: On the one hand, by theory, it limits us missing guarantees of an interpretable topology since we rely on the Hidden Manifold Hypotheses. On the other hand, one may want to decide which scale ϵ in the persistent homology computation result is the most probable, and thus constructing own synthetic labeled space as a dataset gives a little intuition.

Low-Dimensional and Small Examples. In the contemporary literature, new theories on the transformation of the dataset topology by neural networks often relate to pieces of evidence observed on the 2-dimensional or 3-dimensional synthetic datasets. Our plan does not include marking such results as irrelevant for real deep learning applications, but we, especially the newcomers, need to be aware of the following pitfalls:

- *Result size.* The low dimensions of persistence provide a relatively compact description of the dataset. However, this description grows exponentially in the dataset size and more than exponentially with the dataset dimensionality, and thus we may need to use summarization methods more extensively.

- *Interpretability.* The interpretations and implications of higher-dimensional persistent homology properties are not researched and probably remain an open question.
- *High-dimensional Surprises.* Increasing dimensionality by several magnitudes can create quite unintuitive situations, even for standard relations such as the ratio of the volume of ball and cube, [Matoušek et al., 2015, Chap. 2, p. 42 – 43]. Therefore, we must ensure that our intuition is driven by topology rather than local geometry.
- *Nature of Machine Learning.* Special training regimes phenomenon of ML methods, called *double descent*, impacts the neural networks, too. We cover this in Section 3.3.

For a better understanding, we need to design experiments later in this chapter; we have done a survey of used complexes for modeling a topological space with labels before the persistence homology is computed. We distinguish two categories of datasets: synthetic in Table 3.1 and real-world in Table 3.2.

Authors	Description	Dim.	Classes
Guss et al.	8 dsts. of disks and circles s.t. $\beta_0, \beta_1 \leq 5$	2	$M_a \cup M_b = \mathbb{R}^2$
Naitzat et al.	1 dst. of nine disks (a) inside one big disc (b)	2	$M_a \cup M_b \subset \mathbb{R}^2$
	2 dsts. of linked toruses, and concentric spheres of opposite class	3	$M_a \cup M_b \subset \mathbb{R}^2$
Ramamurthy et al.	1 dst. of 25 disks in circles of opposite class	2	boundary manifold

Table 3.1: A survey of papers analyzing synthetically generated datasets.

Authors	Description	Dim.	Classes
Guss et al.	CIFAR-10 embeded in $\mathbb{R}^3$	3	class of 1000 cars images
Naitzat et al.	MNIST embeded in $\mathbb{R}^{50}$	50	one class against 9 other classes
	UCI Banknotes as wavelets	4	two classes of 600 samples
Ramamurthy et al.	MNIST, Fashion-MNIST processed by CNN	128*	boundary of a pair of selected classes
	CIFAR-10 processed by CNN	512*	boundary of a pair of selected classes

Table 3.2: A survey of real-world-based datasets related to computer vision. (*) depends on the used network.

3.2 Gradual simplification of topology

Layer transformations of well-trained networks. We begin with the paper Naitzat et al. [2020] that empirically analyzes topological changes of spaces

between layers of high accuracy (99.99%) achieving networks. Comparing networks with ReLU, leaky ReLU, and hyperbolic tangent activations, they suggest that ReLU accelerates the topology change better because of its non-injectivity, and, on the other hand, tanh should not change the topology because its a homeomorphism of $\mathbb{R}$ and $(-1, 1)$. However, the authors show the non-injectivity of the tanh function evaluation because of the rounding errors of the floating-point operations.

Experimental measurements use synthetic datasets sampled from a manifold M, partitioned into $M = M_0 \sqcup M_1$ for the binary classification. Three datasets consist of concentric spheres and discs in $\mathbb{R}^2$, multiple interlocked tori of opposite category in $\mathbb{R}^3$, and multiple copies of the concentric spheres in $\mathbb{R}^3$. For these datasets, the introduced *topological complexity* is defined by

$$\omega(M) = \sum_i \beta_i(M). \tag{3.1}$$

Since the datasets are generated, we know its topology in advantage, and thus we can estimate the Vietoris-Rips complex parameter directly instead of using the persistent variant of homology. The quantitative experiments on trained fully-connected networks with from 15 to 50 neurons in each layer reveal the following trends:

- the $\beta_i(M_0)$ for i in $\{1, 2, 3\}$ are reduced gradually, layer by layer;
- dataset with higher $\beta_1(M_0)$ was hardest to optimize;
- ReLU activations, bottleneck, i.e., layer of three neurons, or fewer layers accelerate the topological change;
- initial layers decrease the topological complexity slower, transforming indeed mostly geometrically, and the number of layers (4 to 10) has a lower impact on these layers.

The authors also try to test the consistency of noted statements with the more common datasets, using one of the most simple, MNIST, for which they can reduce the dimension of images from $\mathbb{R}^{32\times32}$ into $\mathbb{R}^{50}$ with the principal component projection algorithm. On these projections, they observe gradual topology simplification again, in multiple scales of computed persistent homology.

Empirical lower-bound on network's capacity. The authors Guss et al. [2018] define dataset to be the partitioned space $\mathbb{R}^m = M_0 \cup M_1$. For a function (or neural network) F, we call $\mathrm{dom}(F) \setminus F^{-1}[0]$ the *support of* F, denoted by $\mathrm{supp}(F)$. The crucial argument of the paper is supported by the following theorem:

Theorem 5. *Let $X = X_0 \sqcup X_1$ be partitioned topological space such that X_1 is open. Let $\mathcal{F}$ be a subset of $\{0, 1\}^X$ such that $H_n(\mathrm{supp}(F)) \neq H_n(X_1)$ for some n, then for all f in $\mathcal{F}$ there is $A \subset X_1$, $A \neq \emptyset$ such that $F(A) = 0$.*

Theorem 5 is used to state the necessary condition that the neural network F needs to have an *expressive power* to classify the dataset without the loss of generalization. Next, they suggest choosing the number of units of a concrete neural network based on the Betti numbers of the dataset. Worth a notice, this

is not the most reasonable way to construct the neural network in practice since many architectures can take great advantage of having way more parameters than necessary, as we learn in the next section. They also published experiments on synthetic data, where they observed a decrease in topological complexity, $\omega(M_1)$, after the first fully-connected layer with its increasing size. The experiment gradually employs one to seven hidden units in the first layer of 1-layered or 2-layered networks. This observation is very similar to more general observations done by Naitzat et al. [2020] two years later.

3.3 Homology during Deep Double Descent

When experimenting with machine learning, one may experience the *training set overfitting*. Meaning that our model is getting too specific to the training data that it starts performing poorly on the test set.

Bias-variance explanation and its limits Related is the concept of statistical learning, a *bias-variance trade-off*: increasing the number of parameters or training epochs minimizes bias. However, with the growing model size, the model is more sensitive to small local specificities, which increases its variance. Contradicting this bias-variance theory, many machine learning researchers reported that larger models learn faster and might achieve a higher degree of generalization. A recently published paper by Nakkiran et al. [2019] provided experiments on several deep learning domains, that approximately shows the existence of a so-called *critically parametrized regime* when the testing error might increase (model is overfitting), but this is limited since the error starts descending again after a while. Therefore the name, *double descent*. The authors provide an informal definition of the parametrized regimes by stating the following model complexity measure, note they emphasize the role of a concrete training procedure:

Definition 17 (Effective Model Complexity)**.** *Let $\mathcal{D}$ be the data distribution and n the number of training iterations (possibly averaged into the batches). The* Effective Model Complexity *of a training procedure $\mathcal{T}$, with respect to distribution $\mathcal{D}$ and parameter $\epsilon > 0$, is defined as:*

$$\mathrm{EMC}_{\mathcal{D},\epsilon}(\mathcal{T}) := \max\{n : \mathbb{E}_{S\sim\mathcal{D}^n}[\mathrm{Error}_S(\mathcal{T}(S)) \leq \epsilon]\} \tag{3.2}$$

They state a hypothesis describing the *critically parametrized regime* as a situation where $\mathrm{EMC}_{\mathcal{D},\epsilon}$ is close to n, where n is the number of training iterations. That means the situation in the training interval around the point where the model archives a close-to-zero training error. The situation where n is sufficiently larger, respectively smaller, is then called *under-parametrized*, respectively *over-parametrized regime*. The critically parametrized regime is interesting since, as we noted, the training error may or may not temporarily increase. Furthermore, they observed the double descending trend in multiple scenarios: epoch-wise (increasing the number of training epochs), model-wise (increasing the model's size in the sense of the number of layers' units, the number of layers is constant), and surprisingly sample-wise during the increasing amount of the training examples.

To demonstrate that the deep double descent is a robust phenomenon, the authors, as the major experiment, trained the *ResNet-18* [2015] convolutional network on the CIFAR-10 dataset. Under the standard conditions, the double descent is not significant. However, when they added 15% of noise into the training labels, the phenomenon of double descending is robustly observable both in the epoch-wise and the model-wise manner. Using the given data, we reproduced the experiment of which the equivalent results we visualized in Figure 3.1.

The available literature on the topology modification of data in neural networks does not consider this phenomenon since it typically deals with smaller models and less complex datasets, for both reasons falling safely within the (standard) under-parametrized regime. However, the topological properties may also be influenced by this phenomenon. Therefore, observing the topology of real-world models during parametrized regimes is essential for our further understanding.

3.3.1 Our experiments

As already noted, we reproduced the major experiment of the paper by training the ResNet-18 CNN of multiple width parameters on the CIFAR-10 dataset. We used Adam optimizer with the learning rate 0.0001 for all training sessions of 4 000 epochs. The width parameter is applied linearly to the proportional size of the number of channels in each layer and the network of width 64 is the standard version of ResNet-18. A basic data augmentation such as horizontal flips and random padding and cropping is was used. Our extension of the experiment is that we computed the persistent homology of dimensions 0 and 1 of the internal network *features* of test data. Precisely, we call *feature* the input if the last layer of the CNN, which is the final representation of the input image transformed by the convolutional part of the network, since the last softmax-activated layer is fully-connected. The test set has 10 000 images, and since CIFAR-10 has ten distinct classes, we used a filtration defined by partitioned complex, Definition 16 so that all features of the same class corresponds to a partition. The result is a topological data analysis of the boundary complex described by Ramamurthy et al. [2019].

We tested various summaries of the computed persistence diagrams, such as counting the number of features, summing the overall persistence, the norm of persistence diagram, or even more extended embeddings of diagrams. In the end, we summarized the hundreds of points of a single diagram by a real value so that we could observe potential trends during the deep double descent. We give the most exciting 2d summaries. See Figure 3.2. Although we trained isolated networks, the topology of the computed homology of the boundary is smooth in both directions—epoch-wise and model-wise. This is because persistent topology is a robust descriptor. On the other hand, the topological noise, i.e., points near the diagonal of the persistence diagram (having a short persistence), is unstable: we can observe this evidence in the comparison of Figure A.1 and Figure A.2. Last but not least, we also computed higher persistence groups and corresponding diagrams up to dimension 3 on the fixed subset of the test set, having its size 500 points. Interestingly, the shape of 2d summarizations was not essentially different

from the here given figures, so we omit this without confusing the reader.

Homological Double Descent. We experimented with the smaller CNN's noted in the original article. First, we compared the training procedure where the double descent occurs to the one where the double descent phenomenon is not present. We were able to distinguish visually between those two concrete cases by looking at the raw persistence diagrams. We also noticed the decreased number of 1-dimensional persistence classes approximately in the critically parametrized regime. In Figure 3.2 (d), we can observe the same analogical trend, where the number of 1-dimensional persistence classes. We hypothesize that by entering the critically parametrized regime, the features became non-usable for further optimization, which is forced by the optimizer. Therefore the network might need to slightly modify the representation so that the training error can decrease to a small ϵ. Finally, this hypothetical 'representation' is maybe impossible to happen in networks of a smaller number of parameters. Therefore we also included the case of 80-width ResNet-18, which is not covered in the original paper. Our experiment suggests that there might be more going on to be observed in the case of ResNet-18 models even wider than 80.

Implementation The implementation of the experiments is done using the PyTorch framework [2019], and we used Giotto-PH [2021] for the computation of Vietoris-Rips persistence. The overall training and persistent homology computation took two weeks of machine time on Ryzen 5 processor and NVIDIA GeForce RTX 3070 graphic card. We published the Python 3 code of all experiments included in this thesis in GitHub[1], experiments are fully repeatable with the default seed options.

3.4 Discussion

Homology is a very general theory that can characterize various topological spaces. However, to compute the persistent homology of spaces having labeled parts (or partitions), we needed to extend the definition of filtrations complex. Although we found multiple solutions in the literature, most of the used solutions were intended to be used only for the binary partitioned space. Therefore, we chose to analyze the homology of boundary manifold between distinct classes using the partitioned Vietoris-Rips complex.

Furthermore, according to our survey, most experiments observing the homology of the dataset modified by the neural network used only small fully-connected networks instead of larger accurate convolutional models. Then, we give details about the non-linear behavior of the phenomenon known as deep double descent to the reader. We selected and replicated the central experiment, during which we measured various summaries of computed persistence homology. The results were partial as we expected, in the sense of stability and robustness of such measure (we also give an example where the poorly defined summaries are not stable and robust if based on the topological noise.) The resulting 2d plots correspond

[1]PersiAn Github Repository: `https://github.com/marek094/persian`.

closely to the generalization of the neural network. The second surprising part is that the homology is hypothetically responsive to some precursors of the critically parametrized regime, according to the trend observed in the last plot of Figure 3.2. We also considered computing persistence homology groups of smaller subsets of the test set. However, we did not see any additional aspects. Indeed, the observations done in this section may not be optimal since we interpreted the persistence diagrams by simple statistics, such as sum, mean, etc. Therefore, in the next chapter, we focus on extracting the descriptive information from diagrams using various deep learning-involving techniques, which we further put into the comparison.

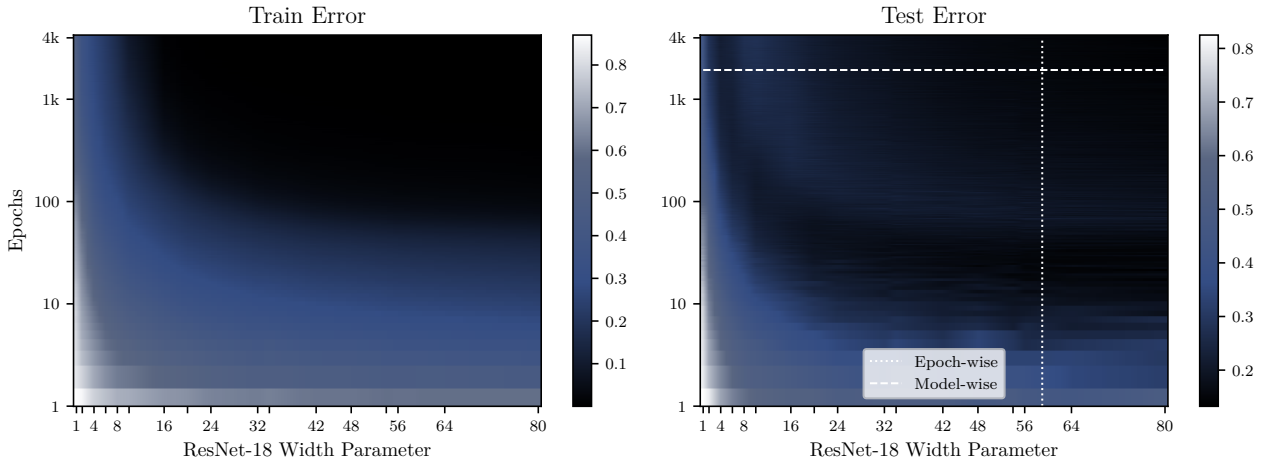

Figure 3.1: **Deep Double Descent.** The deep convolutional network ResNet-18 trained on CIFAR-10 dataset with 15% added label noise. The missing values are interpolated between all neighboring ticks.

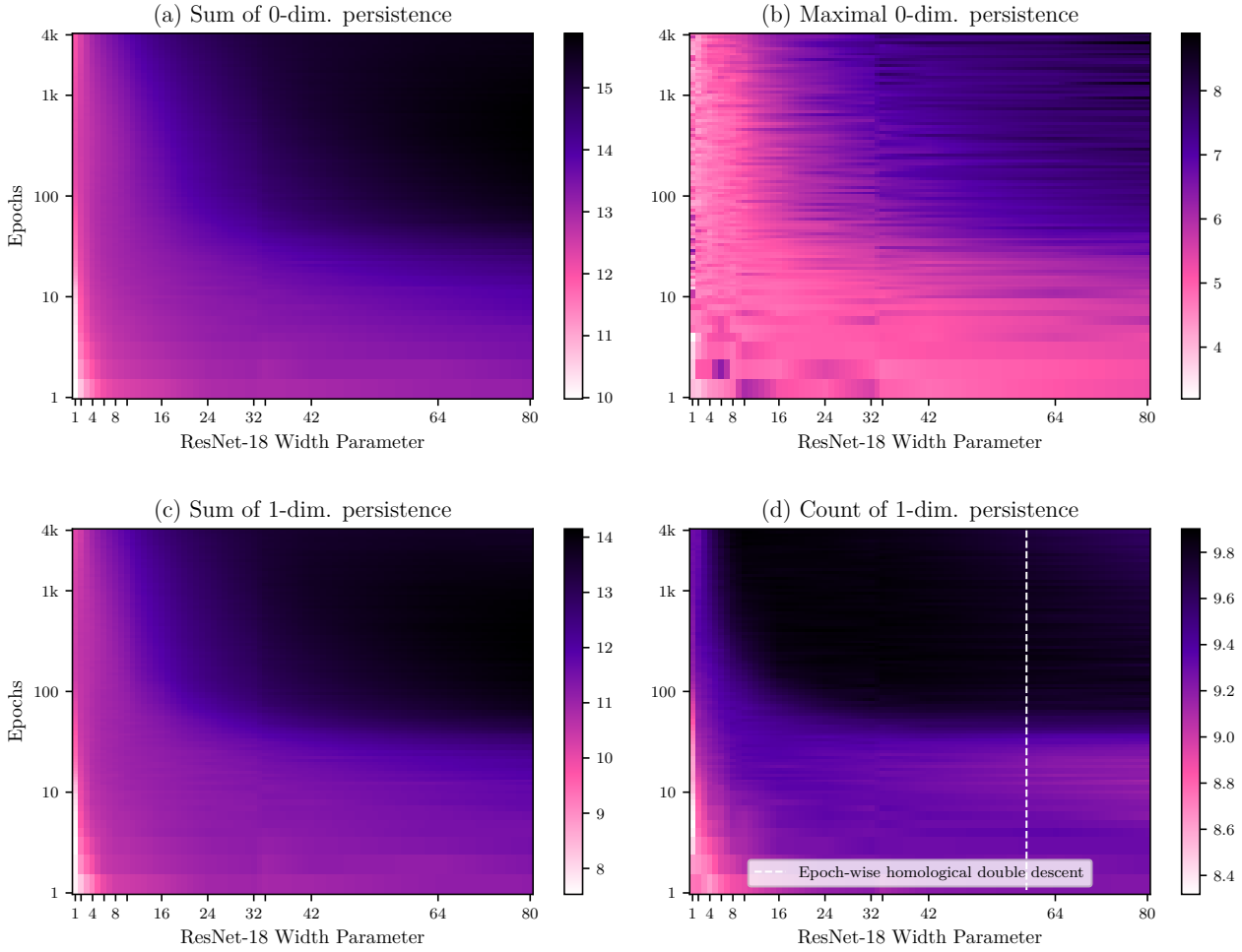

Figure 3.2: **Homology.** Behauviour of the boundary complex' 0-dim. and 1-dim. persistent homology observed on the test data during the double descent.

4. Data-free Topological Accuracy Prediction

In this chapter, we develop and evaluate several approaches to the construction of simplicial complex filtration that models a neural network. In addition, we do not want to involve any example of training or validation data during that process. Initially, there exists no obvious definition of the simplicial complex based on a convolutional neural network compared to the analysis of the dataset/features transformed by the neural network. We suppose that functional filtration can give us more insight into the importance of specific structures in the network. Moreover, we may not have access to the training data or the training procedure in practical situations, but still, we would like to infer more about the model properties.

We start with a summary of related work connected to model analysis of using persistent homology. Consecutively, we give information about the choice dataset and give details on other papers involved in accuracy prediction. Then we present definitions of three constructions with the context of deep learning phenomena. Next, we propose our fourth best-performing method called Differentiable Persistence Accuracy Estimator (DPAE). We compare DPAE to other published methods, and we summarize these experiments. Last but not least, we observe the results of gradient descend optimization and discuss the connection to the existing knowledge about the topology of 3 by 3 image patches.

4.1 Related Work

Neural persistence. As far as we know, the first relevant approach of analyzing the neural network models (in a data-free manner) using persistent homology was published by Rieck et al. [2019]. In contrast to previous approaches, they proposed applying the persistent homology to the model without interrogating the network with training or validation input data: the approach structuralizes the layers of fully-connected neural networks. They model the dense layer as a weighted bipartite graph G, the first partition formed by m input units and the second by n output units. For the k-th layers weight matrix W_k in $\mathbb{R}^{m\times n}$ they define W'_k in $[0,1]^{m\times n}$ having absolute values of W_k entries further normalized by the entry of the largest absolute value. Then W'_k weights also all mn edges between its partities. Indeed, edges of graph G are filtered by the weights absolute value, magnitude, in descending order. Since this graph cannot form simplices of dimension higher than 1, edges, they compute with only the 0-dimensional persistent homology.

A resulting novel measure they call *neural persistence of the k-th layer* defined by the norm of persistence diagram Dgm_0 induced by W'_k. They note that this method is fast since only Dgm_0 is needed. Moreover, they show by an experiment involving hundreds of simple networks trained on MNIST that neural persistence can distinguish between freshly uniformly initialized, trained, and diverging

regimes. Furthermore, they present an early stopping criterium based on neural persistence. The supplementary material extends the theory to CNNs by unrolling the convolutional kernel into a dense layers-like weight. However, as the authors expected, the empirical observation did not transfer in their experiment with two-layered neural networks.

Topological measurement. A similar but independent approach to modeling neural networks by simplified computation tree was employed by Watanabe and Yamana [2021]. They propose a more sophisticated filtrations version by defining graph G on vertices formed by all neural network units and edges between all vertices. For the kth-layer weight W_k, they construct W_k'' by replacing negative values with zero and normalizing by the sum of all entries. Next, they give weight to the edges corresponding to the heaviest paths between neurons by a simplified version of *relevance* that does not depend on a concrete input. The relevance function is motivated by backpropagation. However, they did not suggest any extension of the proposed filtration to CNNs. The authors set up experiments with two-layered networks trained on MNIST, varying the number of classified classes. Unfortunately, the conclusion is not very strong since the experimentation was quite limited.

4.2 Methodology

According to the recent publications of topological modeling, we realized there are not enough benchmarks for evaluating filtrations. Usually, for evaluating persistent homology features, the following datasets are used

- Pointclouds of human poses
- Grayscale textures
- MNIST dataset

In our opinion, these datasets do not leverage the key advantages of persistent homology because of their low dimensionality and natural interpretation. Even worse, these methods applied to the dataset do not show a clear advantage of using advanced theory such as persistent homology since other state-of-the-art methods often exist. According to the motivation described in Introduction, we decided to search for complex data that are not yet well understood. Such investigation can potentially give new insights along the way rather than showing a neat usage of persistence. These ideas lead us to analyze the (convolutional) neural networks models. Finally, we admit there exist exceptions to our findings, one of them is the usage of persistent homology as a part of the readout function in graph neural networks (GNN) Hofer et al. [2021]. However, the graph classification lies outside of the scope of our work to discuss it in detail.

Not only analyzing neural networks using other neural networks. The first thing to realize is that comparing the filtration methods objectively on their descriptivity is a hard and challenging task. The objective comparison would require the development of and unique and interpretable method giving

possibly optimal results for each selected persistence-generating method. Consequently, we decided to employ neural networks, a universal approximator, as a tool extracting the information inside of the resulting persistence diagrams, also following a recent trend according to a survey by Pun et al. [2018]. After we have done preliminary experiments using a dataset of neural networks trained by us, we search for comparison with other methods. We select a dataset of models where deep learning does not outperform other machine learning methods on hand-crafted features in the following section.

Validation Accuracy. For the rest of our experiments, we focus attention on predicting neural network accuracy. First of all, we plan to measure the properties of convolutional neural networks with persistence homology. Second, we also evaluate the *descriptivity* of persistent homology, which is constructed on more abstract topological spaces. For example, applying persistent homology to 3d point clouds or 2d meshes classification is questionable since deep learning methods can usually outperform such effort.

4.2.1 Predicting from Neural Network Model

We identified two publications on using only the model to predict its properties, namely by Eilertsen et al. [2020], Unterthiner et al. [2021]. Both papers employed machine learning techniques, but the former strongly emphasizes (backward) predicting the network's hyperparameters, whereas the latter focuses on the (forward) prediction of the network's validation accuracy. To demonstrate the achieved results, authors of those publications created new datasets of networks, both of them they published, the version of Unterthiner et al. has a more straightforward and regular format. Furthermore, even though their method achieves at least 0.98 R^2 scores in the prediction task, the underlying dataset, called *Small CNN Zoo*, appears to be sufficient to benchmark our topological approaches. Therefore, we proceed in detail with this paper and its Small CNN Zoo.

The authors highlighted as the significant contribution their experiments surprisingly showed that the network's generalization ability could be predicted only by looking into the weights and biases of the trained model. Therefore the paper name, *Predicting Neural Network Accuracy from Weights*. They also mentioned:

> The studies presented in this paper may raise more questions than they answer, but we hope that this will serve as starting point for other researchers to make progress in understanding deep learning phenomena.

Predicting from weights. They employed three model types, two of them worked better: neural network and gradient boosting machine (GBM). First of all, they tried to predict the validation accuracy directly from all the networks' parameters flattened to a single vector which performed "ratherly strong", most significant role, however, played weights of the last layer. Finally, they manually computed statistical summaries for each layer's weight/kernel and bias entries:

the used models are generally not powerful enough to sort values, so they provided on the input the *mean*, the *variance*, the *minimum*, the *maximum*, the *median*, and two *quantiles* of 25% and 75%. Having these inputs, NN and GBM gave the best results for all four versions of the Small CNN Zoo, i.e., trained on CIFAR-10, SVHN, Fashion-MNIST, and MNIST. Interestingly, the NNs achieved an R^2 score of 0.980, whereas GBMs achieved a better R^2 score **0.984** on the hardest version, CIFAR-10.

4.2.2 The Small CNN Zoo dataset

The *Small CNN Zoo* dataset is part of the above-noted paper of Unterthiner et al. First, as we already listed, it has four versions, each corresponding to CIFAR-10, SVHN, Fashion-MNIST, and MNIST; all of them are listed together with examples in Section 1.2.5. Each version consists of approximately 30 000 trained convolutional networks. The CIFAR-10 and SVHN standardly consist of color images, but they converted them to greyscale. Therefore networks in the dataset are of the same structure and thus of an equal number of parameters: there are convolutional first three layers, each has 16 output channels and is followed by dropout. At the end of all networks, we compute the global average for all channels and pass the feature to the full-connected layer. This relatively standard architecture has trainable 4 180 parameters. Denoting uniform distribution by U and log-uniform by U_{log}, let us briefly summarize the zoo of their hyperparameters:

activation function	ReLU, tanh	Section 1.1.1
learning rate	sampled from $U_{log}\,[0.0005, 0.05]$	Section 1.1.2
initialization method	normal, truncated, Glorot, He, orthogonal	Section 1.2.2
initialization variance	sampled from $U_{log}[0.001, 0.5]$	Section 1.2.2
biases initialization	zeros	
optimizer	SGD, Adam, RMSProp	Section 1.2.3
regularization (L_2)	sampled from $U_{log}[10^{-8}, 10^{-2}]$	Section 1.2.4
dropout rate	sampled from $U[0, 0.7]$	Section 1.2.4
training data size	10%, 25%, 50%, 100%	

Furthermore, they uniquely seed the pseudorandom generator for every CNN's initialization and training procedure. These networks do not achieve state-of-the-art validation accuracy nor minimize the training loss to zero because of the small number of parameters. Indeed, all of them represent the *under-parameterized regime* being discussed in the previous chapter. The Small CNN Zoo dataset is available online [1] under CC-BY 4.0 license.

4.2.3 Neural Classification of Persistence

In the previous subsections, we found out that predicting convolutional neural networks generalization success only from the model is an example of a problem where the input features must be manually computed rather than learned.

First of all, we convert the neural network to a space where we can construct

[1] Small CNN Zoo Repository: `https://github.com/google-research/google-research/tree/master/dnn_predict_accuracy`

space, define filtration, and compute the persistence. Then we use these features as an input for another fully-connected neural network to predict properties that interest us. One of such properties, validation accuracy, seems to be very important, and it is discussed in the literature, so validation accuracy is our main focus. Before we use the above-noted setting to measure descriptivity of persistent homology, there still remains a question: *So, what is a good way of constructing the space and filtration?* In the next section of this chapter, we outline some possible answers.

4.3 Model Filtrations

In this section, we define three ways of constructing filtration. In detail, we propose two our methods, namely the *random space filtration*, the *function graph filtration*. The method called *computation tree filtration* is taken from the literature. We also give an example of constructing them on the CNN Zoo dataset.

4.3.1 Random Space Filtration

For now, we think of CNN's layers as a source of dimensionality of our space. Namely, we propose *random space* the set of vectors such that the i-th coordinate is chosen randomly from the i-th network layer's weight. We fix our random choices to be the same for all networks:

Construction 1 (Random Space)**.** *Let F be l-layered neural network. Let K_1, $K_2, \ldots, K_k$ be its convolutional kernels (Equation 1.17), and W_1, $W_2, \ldots, W_{l-k}$ its fully-connected weights (Equation 1.1). Then the* random space *of F is defined as follows*

$$S_{rnd}(F) \subset_{\#} \{(k_1, k_2, \ldots, k_k, w_1, w_2, \ldots, w_{l-k}) : \tag{4.1}$$

$$k_i \text{ entry of } K_i,\ w_j \text{ entry of } W_j\}. \tag{4.2}$$

The sign $\subset_{\#}$ stands for a random but fixed-size subset.

The random space can have an optional size n, but we choose n as the number of entries of the smallest network layer's weight or kernel. After we generate this space and add l_2 metric, we compute standard Vietoris-Rips complex filtration by Definition 11. Although this method is abstract, we can observe its performance on neural models having a variable number of layers, layer sizes, or generally parameters of variable dimension. Such a simple but generally applicable construction might be a sufficient baseline for other, hopefully, better topological approaches. Nevertheless, we shown this baseline in *ICLR 2021 Challenge for Computational Geometry & Topology* organized by Miolane et al. [2021], demonstrating the usage of *Giotto-TDA* library (Tauzin et al. [2020]). The library was published in spring 2020, a couple of months after starting this thesis, and we adapted it for our prototyping.

4.3.2 Computation Tree Filtration

Another way of computing filtration of neural network's computation tree was published by Rieck et al. [2019]. They simplify the computation tree to a weighted

graph. The graph is filtered by adding edges between layers corresponding to networks weights ordered by their descending absolute value. The computational graphs are, however, directed and acyclic, which require us to adjust the definition of complexes in a similar sense to Definition 16. Furthermore, the weights of computed convolution are fixed; therefore, the graph is very symmetrical, and thus we expect the filtration to have the subcomplexes copied multiple times.

Construction 2 (Computation Tree Space)**.** *Let F be a neural network, and let $W \in \mathbb{R}^{n\times m}$ be a weight in a layer of F. We construct a bipartitioned space $S(W)$ by $(\{1,2,\dots,n\}\times\{0\}) \sqcup (\{1,2,\dots,m\}\times\{1\})$. For elements of $S(W)$ we define a distance function $d\colon S(W)\to\mathbb{R}_0^+$ by*

$$d((i,a),(j,b)) = \begin{cases} 0 & \text{if } i=j \text{ and } a=b, \\ (\min\{|w| : w \text{ entry of } W\})^{-1} & \text{if } a=b, \\ (|W[i,j]|)^{-1} & \text{otherwise.} \end{cases} \tag{4.3}$$

Let W_1, $W_2,\dots,W_l$ be all weights of a neural network. Then the computation tree space *is defined as follows*

$$S_{cpt}(F) = \bigoplus_{l'=1}^{l} S(W_{l'}), \tag{4.4}$$

assigning infinity where the union of distance functions would be undefined.

For the convolutional networks, we can always extend the multiple copies of convolution's *kernel* to a *weight* by flattening the inputs and the outputs and filling zero-weights into non-existing connections. However, this operation will introduce multiplicities in the complexes of the filtration and possibly also the multiplicities of the resulting persistence features.

Compared to our constructions, the computation graph construction is distinct because we construct space on a set of bipartite graphs instead of a single geometrically realizable space. Indeed, we need to employ the Partitioned Vietoris-Rips Complex, using Definition 16, to all the two partitioned spaces.

Lottery Ticket Hypothesis. The reader may notice the specific way of constructing distance function d by adding the edges in the descending order of their magnitude. Clearly, filtering edges in an opposite order would not result in d being metric either, so there is a choice made by Rieck et al. [2019]. However, an independent theory proves that networks weights of a higher magnitude are essential in both fully-connected and convolutional networks. The initial paper was written by LeCun et al. [1990], noticing that if we remove the small-magnitude weights from the network, we get a similar accuracy. This method is called *pruning*. Pruning was shown to be effective repeatedly, and usually, we can remove 80% to 90% of weights in a fully-connected network. One may ask why we are training such large models when the most parameters are not necessarily? Frankle and Carbin [2019] gave a possible explanation by stating *the Lottery Ticket Hypothesis*:

> A randomly-initialized, dense neural network contains a subnetwork that is initialized such that—when trained in isolation—it can match

the test accuracy of the original network after training for at most the same number of iterations.

They support this hypothesis by a series of experiments selecting the *winning tickets* by a variety of pruning techniques, for instance, iterative pruning and one-shot pruning. Therefore, with the S_{cpt} (Construction 2), we also investigate the topology of the network according to the (one-shot) decreasing pruning rate, which is a crucial aspect of understanding deep learning.

4.3.3 Filtration Of Function Graph

Until now, we looked at the neural network models from the side despite representing a composition of functions that evaluate the inputs. Specifically for the classification task, networks produce logits passed through softmax non-linearity. Thus, we can ask a natural question: What are the topological properties of the graph of the dependence on inputs?

Construction 3 (Function Graph Space)**.** *Let F be a neural network with the input dimension d. Let I be the set of finite number of samples of a fixed manifold in $\mathbb{R}^d$. The* function graph space *is defined as follows*

$$S_{fnc}(F, I) = \{(z, F(z)) : z \in I\}. \tag{4.5}$$

4.4 Differentiable Persistence

In the implementation of Construction 3, we needed to use the trial and error approach to define the possibly most versatile choice, the manifold of the input space I. Being self-restricted only on the model and not the original training dataset makes our situations quite unnatural. We take the unique advantage of working with not a general function F but an inherently differentiable neural network. The second advantage is that we are already using a framework that handles for us the gradient backpropagation automatically and we can freeze the neural model's weights, kernels, and biases so that the optimizer will only update the input space. That being said, the crucial central question remains, whenever we can backpropagate through the persistent homology computation. A positive answer is given.

Propagating gradients through the persistence functor is proposed in the recent literature. We found it first used for optimizing the filtration function as a part of feature extraction on graphs published by Hofer et al. [2021], and two days later as a part of general-purpose Topological Layer by Brüel-Gabrielsson et al. [2020]. Both descriptions of the gradient derivation have some additional requirements. However, instead of formulating them and getting rid of them, we give the reader a perspective that is similar to *max-pooling* in the following two paragraphs.

Differentiable analogy. Max-pooling [1988] has been a method inside of convolutional networks for decades. To explain it, consider a two-dimensional sliding window on the output of the convolutional layer of dimension $w \times h \times c$ returning the maximum of the window for each channel. The number of channels, therefore,

remains the same. Usually, the window is also shifted by a stride value along both dimensions to downsample the result. (Worth noticing, max-pool-based downsampling can improve the scale invariance of the architecture.) Since we selected maximum in each channel, values will remain unused for the computation in the next layer. Hence we naturally backpropagate the gradient through the maximal and do not for the unused values. Note that we need to remember the maxima indices to reverse the direction of gradients. The reader may wonder, what would happen if more than one maximum value in a window exists? The implementation often virtually introduces lexicographical ordering of pairs of the value and its index, which solves the problem practically. Moreover, when we initialize the network with random values, it is improbable that such a situation occurs.

Vietoris-Rips Differentiability. When computing the persistent homology, we first need to build a filtration. In the case of having n points in the space, we use the Vietoris-Rips filtration. By definition, the filtered VR complex depends purely on the pairwise distances, and since there is $k = n(n-1)/2 + 2$ of them (including 0 and ∞), the filtration has k stages. We may assume all distances are distinct, but we can extend the maximality criterion by using the value index, as in the case of max-pooling. Therefore for a fixed filtration, the persistent feature, i.e., its barcode, both borns and dies only in the set of k points. Thus, there are at most $k(k-1)/2$ distinct barcodes in each dimension. Hypothetically, we can compute all $\mathcal{O}(n^4)$ possible barcodes and simultaneously compute the persistent homology and, in each dimension, select only the barcodes that prove to be the results of the computation. Differentiation is thus done analogically with the maxima selection when performing *max-pooling*. We theoretically compute the possible barcodes by simple (and differentiable) tensor operations such as pairwise p-distance and then a transformation into barcode-like tuples. We backpropagate the gradients only via the selected barcodes. Formal proof of this insight gave Hofer et al. [2021].

Now, on the one hand, we give the formal even more simplified construction, but on the other hand, we plan to add gradient learning to modify the input space in our future end-to-end architecture.

Construction 4 (Function Output Space)**.** *Let F be a neural network with the input dimension d. Let I be the set of finite number of samples of an optimized manifold in $\mathbb{R}^d$. The* function output space *is defined as follows*

$$S_{out}(F, I) = \{F(z) : z \in I\}. \tag{4.6}$$

4.4.1 Architecture

Since the backpropagation is possible through persistent homology computation, we intended to test an end-to-end training. After we tested constructions S_{rnd} 1, S_{cpt} 2, and S_{fnc} 3 during the development of our final architecture to select S_{fnc} 3, because of the highest subjectively identified potential.

DPAE. The architecture of *Differentiable Persistence Accuracy Estimator* (DPAE) has several stages illustrated by the Figure 4.1. First of all, we prepare our

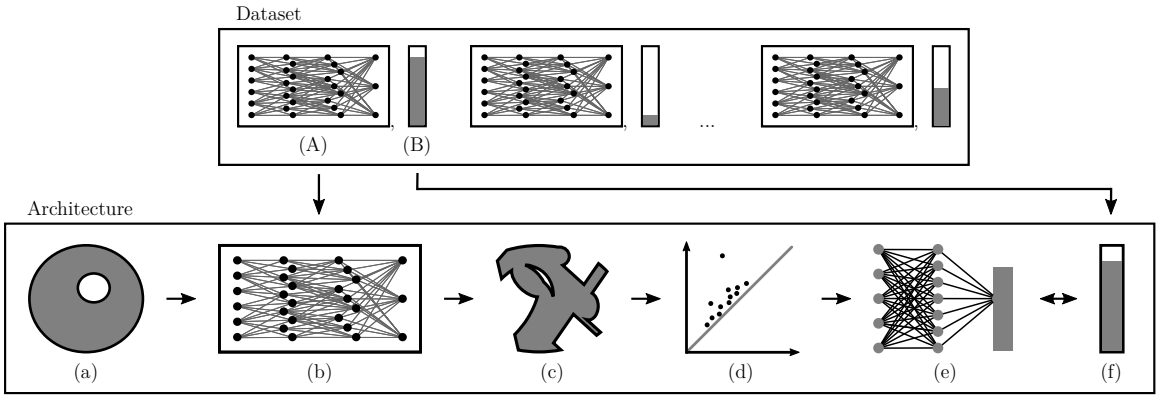

Figure 4.1: **Differentiable Persistence Accuracy Estimator.** Top: *Dataset* with convolutional neural networks (A) as inputs and test accuracy (B) as labels. Bottom: Schema our proposed *pipeline*: (a) input manifold; (b) network of the set (A); (c) modified samples of the input manifold; (d) persistence diagram(s) of topological features; (e) neural network predicting the networks' accuracy using the topological features; (f) the corresponding ground-truth test accuracy value of the set (B). We backpropagate in the opposite direction of the pipeline's arrows.

dataset: pairs of inference state (frozen) neural networks and estimated parameter—accuracy in our case. According to the given Construction S_{out} 4, we randomly sample *the input space*. Each sample has a shape of images used for networks training. The number of input samples, as well as the initialization method, is treated as a hyperparameter. Next, we let process the input space sample by sample with a batch of given neural networks. In the previous chapter, we saw some articles suggesting using dimensionality reduction algorithms for images before computing persistent homology. Notice that the small CNN's reduce the dimension of the input sample $32 \cdot 32$ or $28 \cdot 28$ to the number of classes 10. Vietoris-Rips constructed using the l_p-metric is used to compute the persistent homology of this 10-dimensional space. The dimension of computed persistence homology groups is treated as a hyperparameter, too. We flatten and pad the resulting points of persistence diagrams to be an input to a *trainable* regression neural network. We compose this fully-connected regression network of a couple of layers followed by leaky ReLU activations and the sigmoid output function at the end. The loss function is the standard mean square error of the predicted and ground-truth batch. During the backpropagation, the parameters of the regression networks are updated. Then the gradients are propagated through a persistence-based selection, pairwise distance matrix computation, and a frozen CNN to the input space samples. These samples are therefore optimized, too, as we already noted.

4.5 Experiments

We evaluated all four constructions in test accuracy prediction learning for the CIFAR-10 version of the Small CNN Zoo dataset. Fixing one random dataset split of 50/50 ratio into training and testing sets, we imitate the procedure of

Method	R^2	MSE
Statistics + neural network *(orig.)*	0.980	0.00028
Statistics + gradient boosted machine *(orig.)*	0.984	0.00022
S_{rnd} (Con. 1), VR (Def. 11) + neural network *(ours)*	0.804	0.00270
S_{cpt} (Con. 2), Part. VR (Def. 16) + neural network *(ours)*	0.768	0.00321
S_{fnc} (Con. 3), VR (Def. 11) + neural network *(ours)*	0.845	0.00214
S_{out} (Con. 4), VR (differentiable) + neural network *(ours)*	**0.987**	**0.00018**

Table 4.1: **Test Accuracy Prediction – Results.** We measured R^2 scores (higher is better) and the mean square error (lower is better) of test accuracy prediction on the CIFAR-10 variant of the Small CNN Zoo dataset.

the original Small CNN Zoo paper. We also fixed other random generators with seeds during the training procedure to make our experiments repeatable. Hyperparameters were tunned by hand during several runs, without any expensive hyperparameter tuning method. Next, although the last method, DPAE, requires computing the persistent homology during every training step, we precomputed the non-differentiable persistence homology before the training for the remaining three methods.

Starting with non-differentiable persistence methods, we precomputed the persistence diagrams of dimensions 0 and 1 for each of the 30 000 networks according to the previous definitions. Since the size of the H_1 is not fixed, we computed the maximum of all sizes, and for simplicity [2], we padded all dimension-corresponding diagrams to this maximum size. The vector of such tuples is flattened and processed with a standard fully-connected network of five layers. We trained the network with Adam optimizer (Algorithm 2), with learning rates between 0.001 and 0.0005, and decayed this learning rate after each epoch. The batch size was 128, but 256 or 512 performs similarly. We tested multiple sizes of hidden network layers, but our results were practically stable; we also used dropout (20%) after each layer. The S_{fnc} was given 256 samples of the input space; the final result was obtained by initializing the input tensor using the orthogonal initialization (Section 1.2.2). However, we also tested sampling from the $32 \cdot 32$-dimensional cube, ball, and sphere. A summarization of measured results is present in Table 4.1.

The reader may have already noticed that the last differentiable method in Table 4.1, DPAE (S_{out}), achieves significantly better results than its more straightforward variant, S_{fnc}. However, optimizing the *DPAE architecture* has certain specifics. For each training iteration, we computed H_0 and H_1 persistence diagrams of the input space processes by the CNN on the input. We also flattened and padded the points of the diagrams. As a distance function, we used the l_1, which gave us empirically better convergence than l_2. The finally used networks have five layers, each of 512 hidden units and Leaky ReLU activation. The input space sample set consists of 64 points (images), and it is initialized by random values from the $32 \cdot 32$-dimensional cube. When increasing the number of samples,

[2] We also tried to use vectorization techniques for the embedding of persistence diagrams (persistence images [2016], silhouettes [2013]). However, we did not observe significant improvements even after employing the convolutional networks. Our diagrams are probably too heterogeneous, or the precision is not sufficiently preserved during the vectorization.

Method	MNIST	Fashion-MNIST	CIFAR-10	SVHN
Original	**0.993**	**0.993**	0.984	0.986
Ours	0.989	0.990	**0.987**	**0.987**

Table 4.2: **Performance of Differentiable Persistence Accuracy Estimator.** R^2 scores of the accuracy prediction of DPAE for the other variants of the CNN Zoo dataset (higher is better).

we have got a smoother convergence, but longer training time, and vice versa for decreasing their number We used Adam optimizer for the training, starting with the initial learning rate 0.001, decayed by 0.9 after each epoch. The computation on the input space was done by 128 simultaneous input networks in one training batch. We realized the optimization performs significantly better when the input space is trained with approximately ten times higher learning rate.

Extending DPAE to other datasets. After comparing DPAE to other methods, we want to test if our final method also works on other variants of the Small CNN's Zoo dataset. The training is analogical to the CIFAR-10 variant with only minor adjustments for slightly better results. We learned that the orthogonal initialization of the input space samples representing tensor makes the training more stable, with higher scores as a result. For the SVHN variant, we decreased the learning rate three times together with slowing its per-epoch decay to 0.948 ($\approx \sqrt{0.9}$). The Fashion-MNIST and MNIST variants were trained without additional adjustments (except for the orthogonal initialization). We compared our results to the method in the original paper in Table 4.2.

Input Space Visualization. Our Differentiable Persistence Accuracy Estimator architecture starts with an initialized tensor as an input for the batches of neural networks. We checked this image-like interpretable data for the models trained to archive the scores in Table 4.2. Since we used 64 in training samples, we can visualize them easily. For the CIFAR-10 version, see Figure 4.2, for the SVHN version see Figure 4.3, and for the MNIST version see Figure A.2 in Attachment A.2. Generally, we can clearly see shapes having specific structures instead of observing any objects. For instance, in Figure 4.2 we can see global structures (d4), (a6), (a7), (d8), contrasting patches oriented in certain directions (f1), (g5), (a6), partially blurred patches (d4), (a6), (d8), etc. We also tried to train DPAE with 16 input samples being initialized only in the 4×4 area of the 32×32 image, and we identified similar representants of these classes. On the other hand, our end-to-end architecture optimized different inputs on the SVHN-based dataset since we can mainly see the rounded structures that may be more convenient to photos of numbers. In addition, the structures generated by MNIST-based dataset optimization are also exciting but way simpler, indicating the grid of the convolutional kernel.

4.5.1 Implementation

Our implementation was iteratively evolving along with our new ideas and observations. Moreover, other relevant topological libraries and articles were published

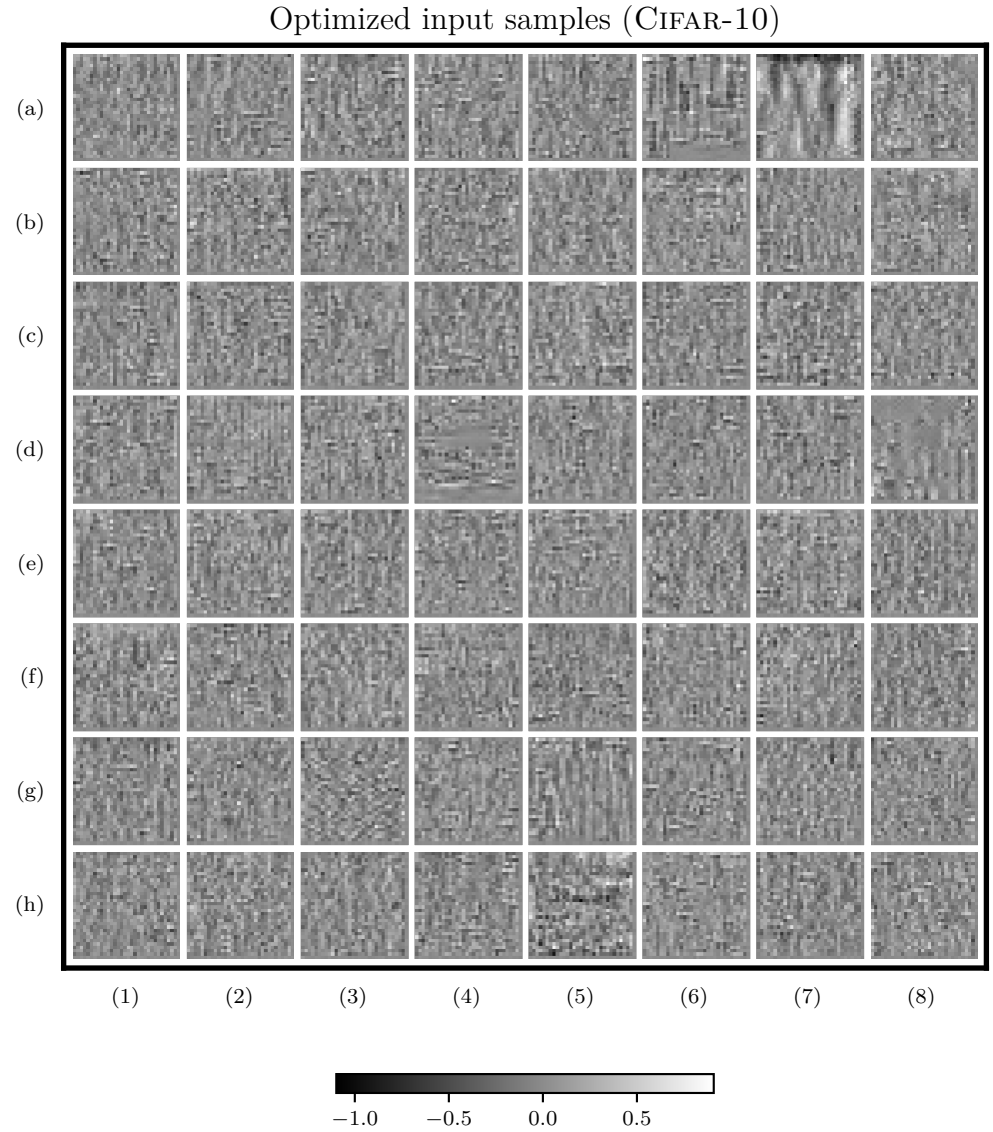

Figure 4.2: **Topologically optimized input samples.** Data optimized by topological gradient-descent inside our DPAE architecture trained on CIFAR-10-based CNN dataset.

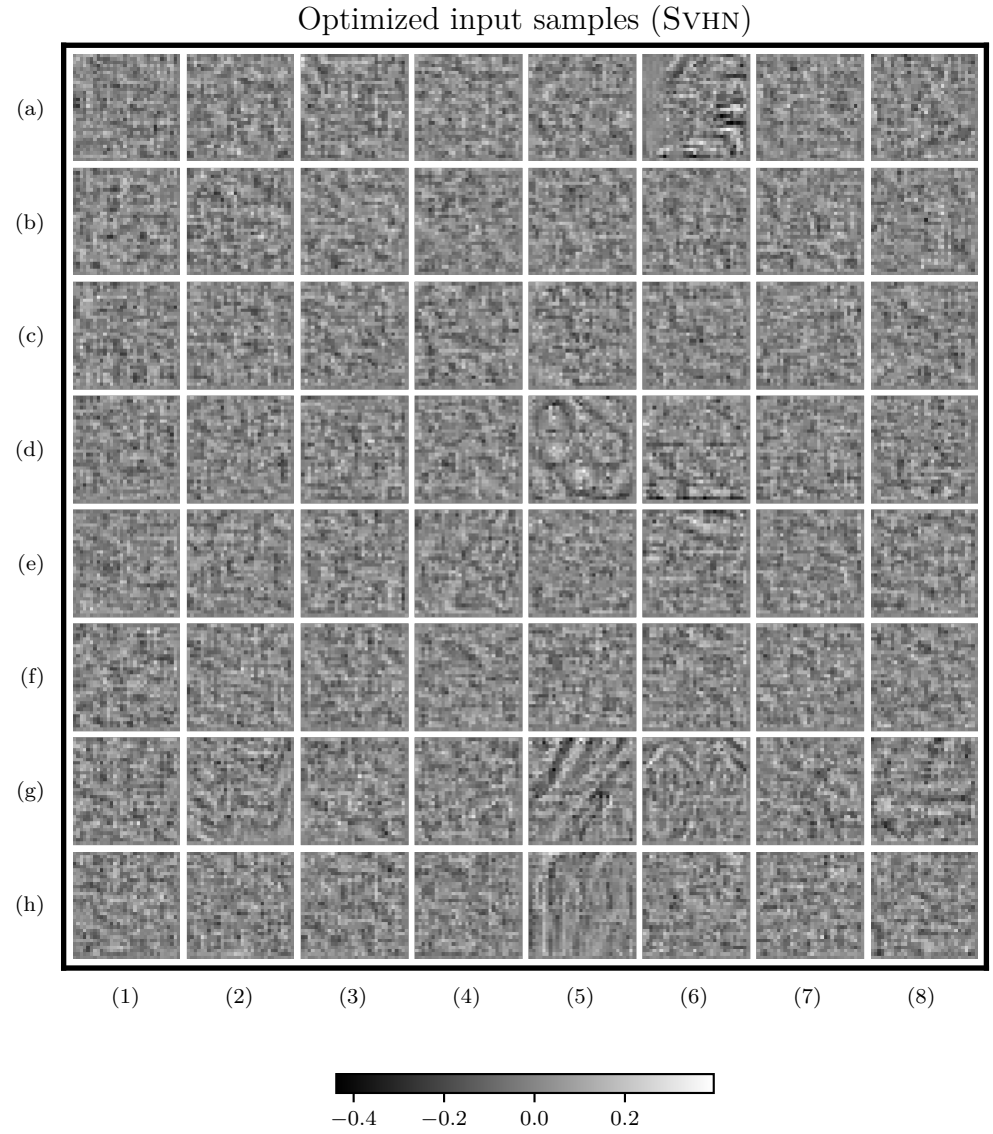

Figure 4.3: **Topologically optimized input samples.** Data optimized by topological gradient-descent inside our DPAE architecture trained on SVHN-based CNN dataset.

during our work on this thesis. Since the Small CNN Zoo dataset was natively developed in framework Tensorflow 2 [2015], for the sake of simplicity, we have done the first comparisons in the Jupyter Notebook [2016] using this framework. The more advanced part of the work we implemented in the PyTorch framework [2019] and wrote the conversion code to port the Small CNN's to PyTorch. We used the Giotto-TDA [2021] and Giotto-PH [2021] for the CPU computation, and for the differentiable GPU computation, we used the TorchPH library by Christoph D. Hofer and Roland Kwitt [2019].

Source code. We published complete Python 3 source code in the *Pers*istence *An*alysis Repository [3]. Nevertheless, our random space filtration experiments are available separately as well [4].

Performance. Since nowadays deep learning frameworks need GPU to run effectively, we managed to fit our experiments to NVIDIA GeForce RTX 3070 with 8GB of chip memory. For Constructions 1, 2, 3, the computation of the persistence homology is the main bottleneck taking several hours for the whole dataset; however, the training itself took around 2 seconds per epoch. For DPAE, we can load all 30 000 networks to the GPU's memory at once, and since we use a relatively small number of training samples (64), one training epoch takes approximately 3 minutes 25 seconds.

4.6 Discussion

We listed four methods of constructing the filtration from convolutional neural networks without any sample from the dataset. Since comparing their descriptivity was unclear, we decided to employ a fully-connected network to evaluate the resulting topological summaries on the test accuracy prediction task. To see whenever our regression takes advantage of the topological shape description instead of only other statical parameters, we selected the existing work and dataset that uses a machine learning method to predict test accuracy from such statistics. We also proposed a universal but straightforward baseline method. In the comparison, the performed *filtration of function graph* better than *computational tree filtration* and *random space filtration*. One of the reasons is that computational tree filtration and random space filtration do not reflect learned biases and use only information in weight matrices. The second possible reason, *computational tree filtration* is not adaptable enough to CNNs, as its authors noted and tested. However, to overperform the simple statistics-based machine learning method, we presented the end-to-end *Differentiable Persistence Accuracy Estimator*.

DPAE can topologically optimize input space for the networks in the dataset, which prevents overfitting and, at the same time, enables a comparable performance to the original method. Moreover, we archive even better results of 0.987 R^2 score on the Small CNN dataset's CIFAR-10 and SVHN versions. In addition, we did not tune the hyperparameters extensively, so it might be possible

[3] PersiAn GitHub Repository: `https://github.com/marek094/persian`

[4] Random Space Experiments: `https://github.com/geomstats/challenge-iclr-2021/tree/main/marek094__Investigating-CNN-Weights-with-Topology`

to achieve even better scores, but that is not essential for this chapter's scope. The key advantage of DPAE compared to that original method by Unterthiner et al. [2021] is the reflection of accidental changes in the network, such as using different activation, randomly shuffling the weight matrix, etc. One could stress this by involving such modified networks dataset where DPAE would have *arbitrary better* results since it involves the input network inference.

The next aspect we want to emphasize, there exist methods optimizing out-of-distribution inputs to fool a *concrete* classifying network by maximizing the output softmax activation of a concrete class, e. g. Nguyen et al. [2015]. We need to realize that the DPAE method works fundamentally differently since it optimizes the most general inputs instead of specific because of training on 15 000 networks. Furthermore, instead of examining the activations, it projects space of the topologically optimized inputs (processed by the network) with a stable functor, persistent homology, and thus it is only analyzing the connectivity in the process of the increasing distance parameters.

The best evidence of the previous facts is that the visualization of the input space of DPAE trained on the CIFAR-10-based version shows similar contrasting subpatterns majorly observed during the topological data analysis of 3 by 3 patches of natural images by Carlsson and Ishkhanov [2007], Carlsson [2009]. These patterns probably do not naturally arise from the initial noise since they were not very significant on SVHN or MNIST-based versions of the input space. Finally, in some sense, our method of DPAE can be loosely seen as a machine learning metaphor to the Rorschach test[5] in the field of psychology.

[5]Rorschach test: —`https://en.wikipedia.org/wiki/Rorschach_test`

Conclusion

In this last chapter, we briefly summarize the results of our work concerning the initial goals of the thesis, together with listing our contributions and their implications. The last section provides questions arising from our experiments that are subjectively worth special attention. Furthermore, we also mention other possible directions beyond the persistent homology.

First, we summarized the preliminaries of the future conclusions. Keeping in mind that multiple backgrounds, we paid special attention not to distract the reader with non-essential details but at the same time to provide a sufficient basis for understanding the further findings.

We followed with the summarization of the existing literature on the analyzing datasets modified by the layers of neural networks. We realized that most of the published articles do not put the major emphasis on the large models used in the impact area of deep learning. Thus we needed to adjust the best suitable existing persistent-homology-based method slightly to analyze the behavior of the widely known, used and studied ResNet-18 model. We observed that the topological shape of the feature representation on the test set empirically corresponds to the error of the network. Importantly, we observed the stable robustness of the topological description, providing a smooth visualization. On the other hand, we noted the example of instability when our description relied on the topological noise. The most interesting observation we have repeatedly seen also with another simpler CNN model than ResNet is the epoch-wise accidental decrease of the otherwise ascending function of 1-dimensional persistence classes number in the part of the plot where we simultaneously observe the double descent phenomenon. Therefore, the chapter describing this experiment provides new data-driven insight into the homological properties of networks' internal representations.

The structure of our experiment shows a general principle we followed in the scope of this thesis. Since there is a wide range of research on many specific aspects of deep learning and neural networks, we combine and compare our findings to other existing deep learning-native experiments, hypotheses and conclusions. So that our insights provided by the computational homology can help step toward a better understanding.

In the second part of this thesis, we compared the descriptivity of various methods of constructing filtered complex modeling neural networks. After a short survey on CNNs accuracy prediction, we selected a recent public dataset of trained small convolutional neural networks. Using this independent data, we put in comparison four methods of persistent computing homology. One of the methods (DPAE) we developed trainable end-to-end using the recent differentiable persistent homology computation. DPAE achieved the best results compared to the remaining three methods. Moreover, DPAE architecture also outperformed the non-topological method of the original article in predicting the accuracy of small convolutional neural networks trained on the CIFAR-10 dataset achieving the R^2 score of 0.987. This result shows the (hidden) potential of using topological methods and neural networks over other standard statistics and classical machine

learning-based methods, such as the original one. As a side but interesting observation, we found out that the topologically optimized input space that the network uses for the above-noted accuracy prediction contains contrasting structures different from random noise. This might be evidence endorsing relatively famous topological data analysis of 3 by 3 natural image patches for higher resolution. Another argument supporting our idea is that we did not observe these structural patterns when predicting the accuracy of networks trained on more synthetic datasets such as SVHN, MNIST.

Future Directions

The discussion sections of both Chapters 3 and 4 provide impulses for us to ask new questions. A significant question arising from the former chapter is whether the homological version of epoch-wise double descent occurs in other domains than computer vision. In the latter chapter, we chose probably an appropriate dataset for comparing various filtration constructions, but it did not show us the limits of the DPAE method that might be clearer using a dataset of other, probably larger convolutional models.

In future research, we suggest stating formal hypotheses where we observed possible evidence in our experiments and try to prove them by applying the formal properties of the persistent homology, such as stability under bottleneck distance. On the other hand, instead of using persistent homology as an input for neural networks, we would like to employ other topology-related objects directly, such as graph neural networks (GNN) [2009] or cell complex neural networks (CXN) [2021].

Bibliography

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`.

Henry Adams, Sofya Chepushtanova, Tegan Emerson, Eric Hanson, Michael Kirby, Francis Motta, Rachel Neville, Chris Peterson, Patrick Shipman, and Lori Ziegelmeier. Persistence images: A stable vector representation of persistent homology, 2016.

Rickard Brüel-Gabrielsson, Bradley J. Nelson, Anjan Dwaraknath, Primoz Skraba, Leonidas J. Guibas, and Gunnar Carlsson. A topology layer for machine learning, 2020.

M. Dixit C. Hofer, R. Kwitt and M. Niethammer. Connectivity-optimized representation learning via persistent homology. In *ICML*, 2019.

Gunnar Carlsson. Topology and data. 46(2):255–308, 2009. ISSN 0273-0979. doi: 10.1090/S0273-0979-09-01249-X. URL `http://www.ams.org/journal-getitem?pii=S0273-0979-09-01249-X`.

Gunnar E. Carlsson and Tigran Ishkhanov. A topological analysis of the space of natural images. 2007.

Frédéric Chazal, Brittany Terese Fasy, Fabrizio Lecci, Alessandro Rinaldo, and Larry Wasserman. Stochastic convergence of persistence landscapes and silhouettes, 2013.

George V. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2:303–314, 1989.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

Herbert Edelsbrunner and John Harer. *Computational Topology: An Introduction*. American Mathematical Society, 2010. ISBN 978-0-8218-4925-5.

Gabriel Eilertsen, Daniel Jönsson, Timo Ropinski, Jonas Unger, and Anders Ynnerman. Classifying the classifier: dissecting the weight space of neural networks, 2020.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

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

Mustafa Hajij, Kyle Istvan, and Ghada Zamzmi. Cell complex neural networks, 2021.

Allan Hatcher. *Algebraic Topology*. Algebraic Topology. Cambridge University Press, 2002. ISBN 9780521795401.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

Christoph Hofer, Florian Graf, Marc Niethammer, and Roland Kwitt. Topologically densified distributions. In *International Conference on Machine Learning*, pages 4304–4313. PMLR, 2020.

Christoph D. Hofer, Florian Graf, Bastian Rieck, Marc Niethammer, and Roland Kwitt. Graph filtration learning, 2021.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, and Carol Willing. Jupyter notebooks – a publishing format for reproducible computational workflows. In F. Loizides and B. Schmidt, editors, *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87 – 90. IOS Press, 2016.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann, 1990.

Jiří Matoušek, Ida Kantor, and Robert Šámal. *Mathematics++: Selected Topics Beyond the Basic Courses*. Student mathematical library. American Mathematical Society, 2015. ISBN 9781470426231.

Tomáš Mikolov. *STATISTICAL LANGUAGE MODELS BASED ON NEURAL NETWORKS*. Ph.d. thesis, Brno University of Technology, Faculty of Information Technology, 2012.

Nina Miolane, Matteo Caorsi, Umberto Lupo, Marius Guerard, Nicolas Guigui, Johan Mathe, Yann Cabanes, Wojciech Reise, Thomas Davies, António Leitão, Somesh Mohapatra, Saiteja Utpala, Shailja Shailja, Gabriele Corso, Guoxi Liu, Federico Iuricich, Andrei Manolache, Mihaela Nistor, Matei Bejan, Armand Mihai Nicolicioiu, Bogdan-Alexandru Luchian, Mihai-Sorin Stupariu, Florent Michel, Khanh Dao Duc, Bilal Abdulrahman, Maxim Beketov, Elodie Maignant, Zhiyuan Liu, Marek Černý, Martin Bauw, Santiago Velasco-Forero, Jesus Angulo, and Yanan Long. Iclr 2021 challenge for computational geometry & topology: Design and results, 2021.

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

Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. *arXiv preprint arXiv:1912.02292*, 2019.

Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images, 2015.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

Allan Pinkus. Approximation theory of the mlp model in neural networks. *Acta Numerica*, 8:143–195, 1999.

Chi Seng Pun, Kelin Xia, and Si Xian Lee. Persistent-homology-based machine learning and its applications – a survey, 2018.

Julián Burella Pérez, Sydney Hauke, Umberto Lupo, Matteo Caorsi, and Alberto Dassatti. giotto-ph: A python library for high-performance computation of persistent homology of vietoris–rips filtrations, 2021.

Karthikeyan Natesan Ramamurthy, Kush Varshney, and Krishnan Mody. Topological data analysis of decision boundaries with application to model selection. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5351–5360. PMLR, 09–15 Jun 2019.

Bastian Rieck, Matteo Togninalli, Christian Bock, Michael Moor, Max Horn, Thomas Gumbsch, and Karsten Borgwardt. Neural persistence: A complexity measure for deep neural networks using algebraic topology. In *International Conference on Learning Representations (ICLR)*, 2019.

Andrew M. Saxe, James L. McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *CoRR*, abs/1312.6120, 2014.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009. doi: 10.1109/TNN.2008.2005605.

Pierre Sermanet, Soumith Chintala, and Yann LeCun. Convolutional neural networks applied to house numbers digit classification, 2012.

Guillaume Tauzin, Umberto Lupo, Lewis Tunstall, Julian Burella Pérez, Matteo Caorsi, Anibal Medina-Mardones, Alberto Dassatti, and Kathryn Hess. giotto-tda: A topological data analysis toolkit for machine learning and data exploration, 2020.

Guillaume Tauzin, Umberto Lupo, Lewis Tunstall, Julian Burella Pérez, Matteo Caorsi, Anibal M. Medina-Mardones, Alberto Dassatti, and Kathryn Hess. giotto-tda: A topological data analysis toolkit for machine learning and data exploration. *Journal of Machine Learning Research*, 22(39):1–6, 2021.

T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

Thomas Unterthiner, Daniel Keysers, Sylvain Gelly, Olivier Bousquet, and Ilya Tolstikhin. Predicting neural network accuracy from weights, 2021.

Satoru Watanabe and Hayato Yamana. Topological measurement of deep neural networks using persistent homology, 2021.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

Y. T. Zhou and Rama Chellappa. Computation of optical flow using a neural network. *IEEE 1988 International Conference on Neural Networks*, pages 71–78 vol.2, 1988.

List of Figures

1 Increasing sea level 3

1.1 Simple neural network 7
1.2 Examples by the dataset 14

2.1 Pointed torus 17
2.2 Simplices 19
2.3 Filtration Example 24
2.4 Example of Persistent Homology 26

3.1 Deep Double Descent 36
3.2 Homology 36

4.1 Differentiable Persistence Accuracy Estimator 45
4.2 Topologically optimized input samples 48
4.3 Topologically optimized input samples 49

A.1 Plot of the minimal 1-dim. persistence in the diagram 60
A.2 Plot of the maximal 1-dim. persistence in the diagram 60
A.3 Optimized input samples 61

List of Tables

3.1 Synthetic datasets 30
3.2 Real-world-based datasets 30

4.1 Test Accuracy Prediction – Results 46
4.2 Performance of Differentiable Persistence Accuracy Estimator . . 47

A. Attachments

A.1 Double Descent: Topological Summaries

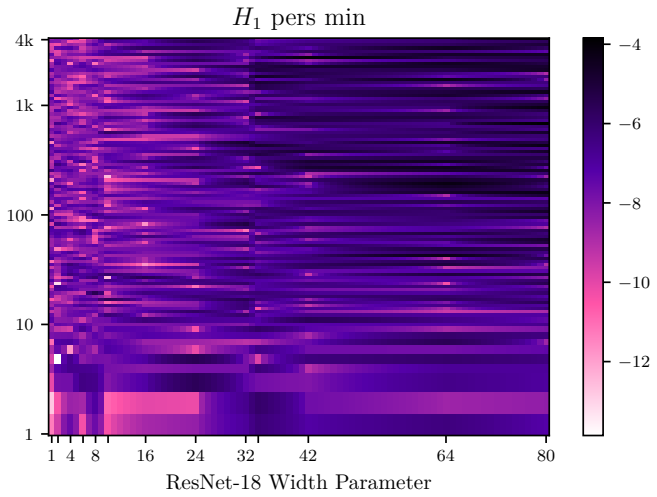

Figure A.1: **Plot of the minimal 1-dim. persistence in the diagram.**

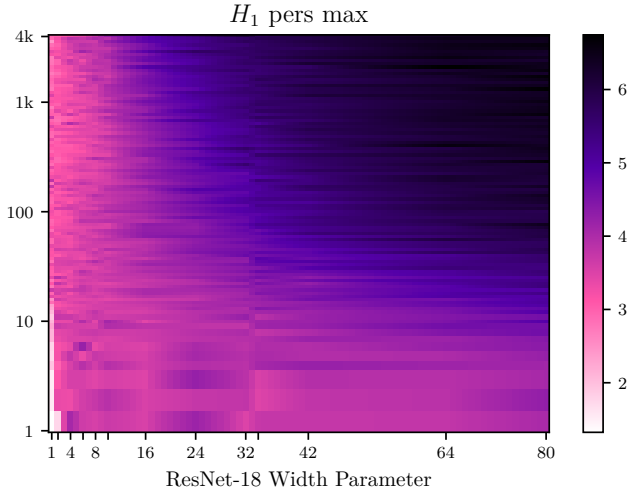

Figure A.2: **Plot of the maximal 1-dim. persistence in the diagram.**

A.2 Data-free DPAE Visualization.

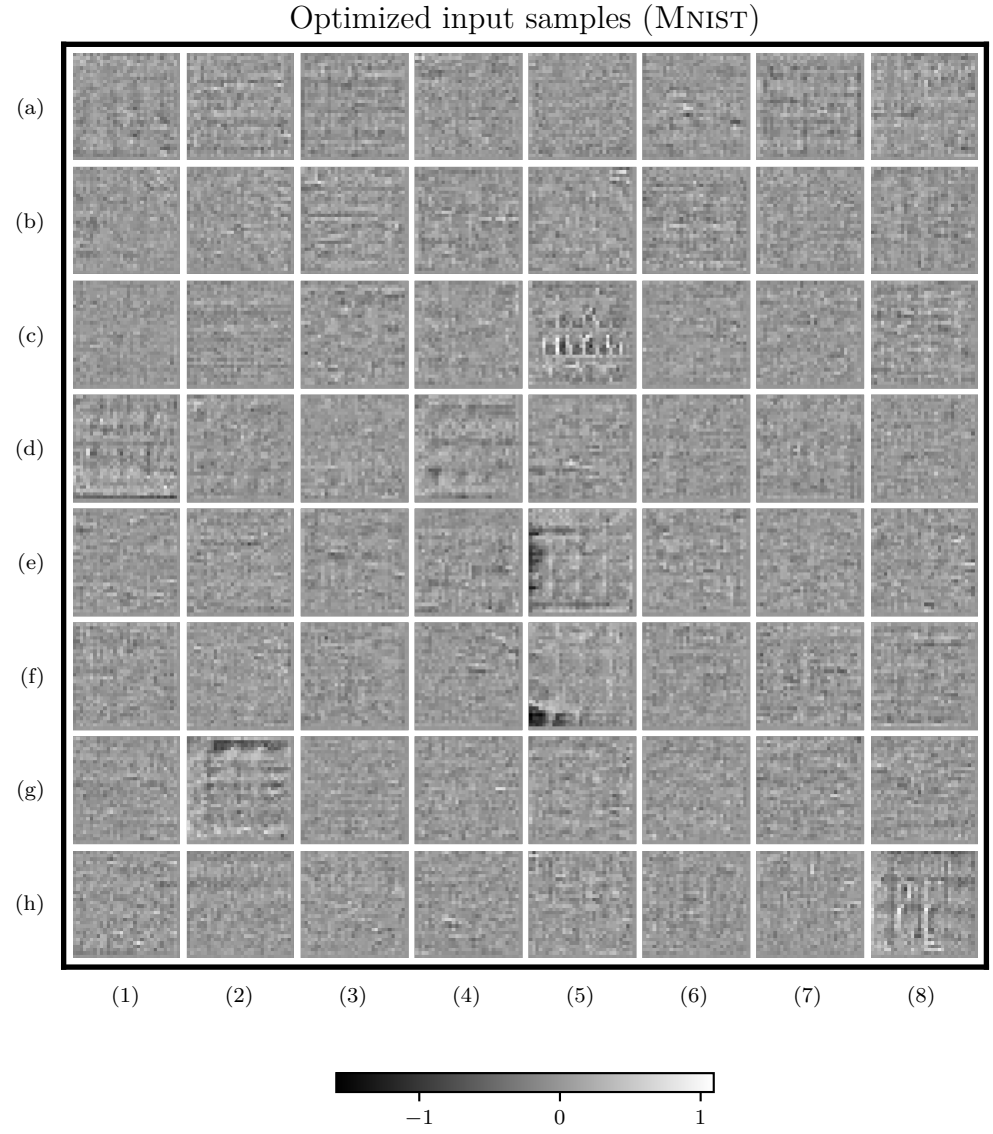

Figure A.3: **Optimized input samples.** Data optimized by topological gradient-descent inside our DPAE architecture trained on MNIST-based CNN dataset.