DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING, SECOND CYCLE, 30 CREDITS

STOCKHOLM, SWEDEN 2018

An Investigation of Neural Network Structure with Topological Data Analysis

VLADISLAV POLIANSKII

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

An Investigation of Neural Network Structure with Topological Data Analysis

VLADISLAV POLIANSKII

Master in Machine Learning
Date: November 8, 2018
Supervisor: Florian Pokorny
Examiner: Hedvig Kjellström
Swedish title: En undersökning av neuronnätsstruktur med topologisk dataanalys
School of Electrical Engineering and Computer Science

Abstract

Artificial neural networks at the present time gain notable popularity and show astounding results in many machine learning tasks. This, however, also results in a drawback that the understanding of the processes happening inside of learning algorithms decreases. In many cases, the process of choosing a neural network architecture for a problem comes down to selection of network layers by intuition and to manual tuning of network parameters. Therefore, it is important to build a strong theoretical base in this area, both to try to reduce the amount of manual work in the future and to get a better understanding of capabilities of neural networks.

In this master thesis, the ideas of applying different topological and geometric methods for the analysis of neural networks were investigated. Despite the difficulties which arise from the novelty of the approach, such as limited amount of related studies, some promising methods of network analysis were established and tested on baseline machine learning datasets. One of the most notable results of the study reveals how neural networks preserve topological features of the data when it is projected into space with low dimensionality. For example, the persistence for MNIST dataset with added rotations of images gets preserved after the projection into 3D space with the use of simple autoencoders; on the other hand, autoencoders with a relatively high weight regularization parameter might be losing this ability.

Sammanfattning

Artificiella neuronnät har för närvarande uppnått märkbar popularitet och visar häpnadsväckande resultat i många maskininlärningsuppgifter. Dock leder detta också till nackdelen att förståelsen av de processer som sker inom inlärningsalgoritmerna minskar. I många fall måste man använda intuition och ställa in parametrar manuellt under processen att välja en nätverksarkitektur. Därför är det viktigt att bygga en stark teoretisk bas inom detta område, både för att försöka minska manuellt arbete i framtiden och för att få en bättre förståelse för kapaciteten hos neuronnät.

I detta examensarbete undersöktes idéerna om att tillämpa olika topologiska och geometriska metoder för analys av neuronnät. Många av svårigheterna härrör från det nya tillvägagångssättet, såsom en begränsad mängd av relaterade studier, men några lovande nätverksanalysmetoder upprättades och testades på standarddatauppsättningar för maskininlärning. Ett av de mest anmärkningsvärda resultaten av examensarbetet visar hur neurala nätverk bevarar de topologiska egenskaperna hos data när den projiceras till vektorrum med låg dimensionalitet. Till exempel bevaras den topologiska persistensen för MNIST-datasetet med tillagda rotationer av bilder efter projektion i ett tredimensionellt vektorrum med användning av en basal autoencoder; å andra sidan kan autoencoders med en relativt hög viktregleringsparameter förlora denna egenskap.

Contents

Chapter 1

Introduction

1.1 Motivation

Machine Learning, as a major part of Computer Science, has been developing and evolving starting from 1950s[1]. Since then, many algorithms, such as Bayesian classifiers, decision trees, etc., have been developed and successfully applied in various areas. Likewise, artificial neural networks started to develop more than half a century ago[2], and during the last decade together with substantial advances in computing power and parallelism neural networks started to gain notable popularity, outperforming other machine learning algorithms on important benchmark tasks like ImageNet[3] and PASCAL[4] classification challenges. Nowadays, people find use for them in a wide and gradually increasing range of fields, from business to medical applications.[5, 6, 7]

In many cases the usage of neural networks, and many other machine learning algorithms, is quite intuitive and easily understandable for a data scientist. A lot of already established frameworks handle most of the hard work. For example, the Keras framework[8] provides a high-level API to work with neural networks, and the scientific library `scikit-learn` in Python contains a wide range of other ready to use machine learning algorithms. Therefore, for a user, only a small selection of parameters need to be chosen in order to run the algorithms.

Despite that being intuitive is generally a good thing, this term can be applied to neural networks from a different, and not so positive, side. Quite often it is only the intuition that helps with setting up the

algorithm. The reason for that lies in uncertainty about what exactly is happening in the internal layers of the network. Most of the time these algorithms are viewed like "black boxes", even though the structure of a network is always known to a scientist.

A number of attempts was made to develop a way to visualize how the networks operate[9, 10]. For neural networks which accept images as their input (such as networks trained for image recognition) it is primarily done by synthesizing input data which maximizes activation functions of separate neurons. These techniques can provide various information about a trained network, for instance how well-trained are its parts at distinguishing patterns in images. These visualizations also show the general distinctions between layers on different depth levels of any network.

In addition, some interesting features that are common for all neural networks were addressed in [11]. For instance, it was observed that neurons of the first few layers of networks should not be seen as separate entities; instead, they transform the input into a space that contains the semantic information about the data. On the contrary, neurons of deep layers seem to be acting more like individual entities, each of them reacting to its own particular feature in the input data.

These and other findings discussed later in Chapter 2 are a significant step towards getting a better view on artificial neural networks. Nevertheless, it also seems that the area has a lot more to explore, perhaps with different methods of analysis. Many questions arise about the functioning of the networks. What kind of dependencies and relations among points in the input data can be captured by neural networks? What exactly happens during the training process with data representations? How could one compare the performance of two networks without looking only at the accuracy and loss values?

1.2 Research Question

Machine Learning normally operates with reasonably large amounts of data. Since every data element can usually be represented as a vector of real numbers, each of them can be considered as a point in a high-dimensional space, which is then mapped into other spaces iteratively by each layer of a neural network. Therefore, at each slice of a neural network we can view our dataset as a point cloud, and

that opens a possibility of analyzing various geometrical properties embedded in the data and carried by the neural network.

One area which can be particularly useful for such analysis is Algebraic Topology. This domain inherits methods which examine the shape that is formed by the data points, disregarded by most quantitative geometric or statistical approaches and instead using the information about the points' locations in relation to each other for more qualitative properties.

Topological data analysis (TDA) could be used to retrieve various features that appear in a point cloud. For instance, some or all points might be lying on some closed curve, which can be detected by the TDA algorithms. Finding and inspecting such curves might tell us something about a rotational symmetry that is present in the data.

1.3 Approach

The main objective of the thesis is to investigate the feasibility of utilizing topological data analysis, and possibly other geometric methods, to understand the structure of a trained neural network. Of particular interest is the question of whether artificial neural networks are capable of preserving the topological features when the data is passed through the layers of a network, or even detecting new symmetries which were found by the network.

1.4 Thesis Structure

Subsequent chapters of the thesis are organized as follows.

Chapter 2 comprises the summary of work done in related areas. It consists of two sections: the first section describes the methods which were already established to analyze the behavior of artificial neural networks; the second section shows how topological data analysis is used in areas, not necessarily so closely related to machine learning.

Chapter 3 provides an overview of the relevant theory. The chapter covers both topological methods used for data analysis and neural network structures which fall under investigation. The last part of the chapter contains a brief summary of work done in related areas.

Chapter 4 describes the most significant details of the programming part of the work. It includes both the part about ML and TDA

algorithms and the part about a special visualization tool, used in the analysis of the results.

Chapter 5 presents the experiments that were conducted and the results obtained. It is divided into several sections, each defined mainly by the dataset that was used for a block of experiments.

Chapter 6 gives final thoughts about obtained results, as well as ideas about future implications and possible directions of further investigation.

Chapter 2

Related Study

2.1 Insight into Artificial Neural Networks

The field of exploring the qualities and features of artificial neural networks has started to draw increasing attention in recent years.

At the moment, the visualization of the activations of the neurons[12] is commonly applied as a way to analyze inner layers of artificial neural networks. If we consider neural networks which take images as an input, one can find such an input image that will produce the highest activation for a chosen neuron or a group of neurons. This is done, for example, by applying gradient ascent to maximize the activation function with input image as a variable. This might help to understand what inputs are preferred by the network, and what structures and patterns it seeks for in images. It was revealed that different channels of convolutional layers of image networks correspond to specific local features in the images, like wheels of a car.

Another technique, also discussed in [12], is to plot regions of a given image which cause high activation values in a given neuron or channel, which is the other way to present patterns important for a specific element of the network. For example, it was shown that one of the neurons on the fifth layer of the convolutional network which was used in the paper has a high response on human faces on images, even though they did not belong to a separate class in the training dataset.

One can look at the evolution of discussed visualizations throughout the learning process[13]. It can show that lower layers of the network – those which are the closest to the input layer – converge rather

quickly relative to higher layers, which might develop very late. It provides a different look, apart from relating complexity of a network with its overfitting, on the fact that reducing the amount of layers of a network might cause a significant performance improvement of the network.

[13] also investigates the stability of the classifier to transformations of the input, such as translation and scaling to different degrees. The study shows that while the first layers are very sensitive to such changes, the last layers and the output of the classifier are a lot more stable to these modifications. However, the network turns out to be not invariant to rotations in a general case.

[14] presents another way to analyze performance of a neural network layer by layer. The authors introduce an application of kernel principal component analysis to neural networks to observe how good the representations of the input on each layer are. This is done by applying Kernel PCA to outputs of the layers for the whole dataset, building a simple linear regression on obtained data and looking at how well it performs. It was found that properly trained deep networks progressively simplify the general representation of the data they work with.

2.2 Topological Data Analysis in Related Areas

TDA is a rather new area of data analysis, it so far there were just a few attempts in applying it to machine learning. [15] shows an interesting way to utilize persistent homology analysis for geometric representation of neural networks. In the work, simplicial complexes are built on neural network graphs, one complex for each input image, filtering each edge by the value that goes through it (product of edge's weight and a value from the previous neuron) when an image is passed as an input. Later, persistence diagrams of those complexes were compared among each other for two groups of input images, one being the regular MNIST dataset while other being the "conterfeit" dataset - images which look like one type, but being recognized like the other. While the second dataset was misleading the classifier into producing incorrect outcomes, the described topologies for alike looking images from different datasets were similar as well. So it was should that this

method of analysis could potentially be used to detect such falsifying images.

Recently, one way of using TDA for comparing neural networks was shown in [16]. The authors introduce a way to evaluate the performance of generative adversarial networks, which is normally difficult to assess in a traditional way from the loss functions. The core idea of the methods described in the paper lies in comparing the topological properties of the initial data and the obtained data.

[17] presents an analysis of neural network capabilities for datasets of different complexity; the latter is defined in terms of algebraic topology. Potentially, the methods proposed in the article can allow having a more guided selection of an architecture for neural networks, depending on data complexity. Another article[18] proposes ways to analyze the topological properties of the decision boundary. The complexity of the boundary might also provide some useful information regarding architecture and model selection.

Some articles explore the opposite direction of connectivity between computational topology and machine learning. Several studies show how machine learning techniques could be used to enhance topological and geometic analysis of data. One of such studies[19] proposes and evaluates a method of applying kernels to persistence diagrams computed for the input data. The kernel was also proven to be stable with respect to 1-Wasserstein distance between the diagrams. After kernel computation, SVM classification was applied to the diagrams of input shapes, which produced good results.

[20] presents a way of training neural networks by the usage of topological information obtainable from the data. They introduce a novel network layer which processes the persistence diagrams of each input and produces a usable projection onto a vector space. This layer is parametrized and fully differentiable, providing the ability to train the parameters with gradient descent together with other layers of the network. This approach outperformed the current state-out-the-art networks on network graph data and showed the possible benefit of using topological features of the data during training.

Another paper[21] discusses an extension of quantum machine learning algorithms to topological data analysis. By the use of proposed algorithms it would be possible to gain an exponential speed-up on the most common topological algorithms, which might be solving a problem of high computational complexity of methods in computational

topology in the high-dimensional data.

In the article [22], less connected to machine learning, a research was conducted on the set of natural pictures, taken with a digital camera in grayscale. All images were cut into 3×3 pixel patches, which could be regarded as 9-dimensional points in Euclidean space. After the topological analysis was performed, it was concluded that these patches form a 2-dimensional manifold called Klein bottle. Authors present a sensible explanation to how the Klein bottle emerges from the studied cloud of small subimages.

Chapter 3

Theoretical Background

3.1 Machine Learning

The major part of machine learning algorithms used in the thesis are the neural networks. This section gives a brief overview of the structures of neural networks which are used in the work. All considered networks have layered structure, i.e. each can be represented as a sequence of transformations between multidimensional spaces, gradually transforming input layer into the output.

All analysis is performed on image datasets. There are two types of networks which fall under the analysis. One type is a regular N-label classifier – for a given image it outputs a vector of N real values. The loss function which is minimized during the training stage is a cross-entropy of the output for an image and the true classification vector for this image, the latter being a one-hot encoding of the image's class.

The second type, and at the same time the one that is mainly used throughout the paper, are classical autoencoders. The output of such networks has the same format as the input, and the loss function is an L_2-distance between the input and the output images. Normally such networks consist of two parts: encoder and decoder. An encoder is formed from first several layers starting from the input and until some fixed layer l. This layer usually consists of a small number of neurons/dimensions.

The decoder is the second half of the network, which takes the layer l as an input and processes it to the output. Usually its structure is mirrored from the encoder: convolutions and fully-connected layers stay on the same distance from the output, as they do from the input in the

encoder, and pooling layers are replaced with unpooling. Convolutional layers may be replaced with transposed convolutions.

The purpose of an autoencoder is to find some embedding of the input space into a latent space l which gives the least loss in image quality during decoding procedure. For a given image, its encoding can be obtained from the layer l.

Apart from neural networks, another machine learning technique called Principal Component Analysis (PCA) is used in the work. This is another method of dimensionality reduction, although, it does not provide a decoding method and is used to represent a given point cloud as a lower-dimensional data. The algorithm finds an orthogonal projection from the initial space into the latent space which preserves as much information in data as possible, or in other words maximizes the variance of the projection.

3.2 Topological Data Analysis

Topological Data Analysis (TDA) is a relatively new collection of data mining methods that mainly relies on the use of methods from such mathematical branches as topology and computational geometry. It is used to represent discrete sets as global structures and to analyze and extract some topological information from obtained constructions. For instance, a 3D object might be represented as a set of points, sampled from its surface, and methods included in TDA allow to restore the continuous approximation of the initial object.

3.2.1 Simplicial Complexes

One of the key concepts in computational topology are simplicial complexes, which act as a basic concept used to represent an object in a multidimensional space.

An *abstract simplicial complex* $\mathcal{K}$ is a finite collection of sets, which satisfies the following condition:

$$\forall \sigma \in \mathcal{K}, \forall \tau \subseteq \sigma : \tau \in \mathcal{K}$$

The elements of $\mathcal{K}$ are called *abstract simplices*. In the following two paragraphs the word "abstract" will sometimes be omitted, when its presence can be deduced from provided information in the sentence or

is unnecessary. In order to relate abstract simplicial complexes to point cloud analysis we identify elements of its simplices as the points from the point cloud. In other words, each abstract simplex is a subset of the analyzed dataset. By definition, all subsets of an abstract simplex are simplices as well, and are called *faces* of the simplex.

The main characteristic of a simplex is its *dimensionality*, which is defined as its cardinality increased by one: $\dim \sigma = |\sigma|+1$. A simplex σ with $\dim = k$ is called k-simplex. So, for example, 0-simplex consists of a single point. The dimensionality of the whole complex is defined as the maximum dimensionality of its simplices: $\dim \mathcal{K} = \max_{\sigma \in \mathcal{K}} \dim \sigma$.

Although abstract simplicial complexes define a general type of simplicial complexes, `geometric simplicial complexes` are commonly used in computational topology. Let us start with the definition of a simplex in such complex.

A *geometric simplex* σ (from now on just a *simplex*) is a geometric object which can be seen as a generalization of a triangle or a tetrahedron in a higher dimensional space. The main characteristic of a simplex is its dimensionality; in this way, σ with $\dim \sigma = k$ is called k-simplex. Thus, a 2-dimensional simplex would be a triangle, while a tetrahedron is a 3-dimensional simplex.

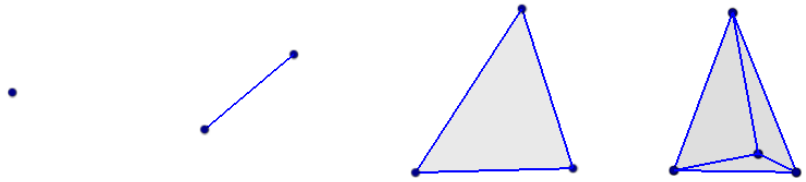
A k-simplex is determined by a set of $(k+1)$ vertices $\{v_i | i = 0..k\} \subset \mathbb{R}^n$ in a general position, meaning that no $(m+2)$ points lie on the same m-dimensional hyperplane. The simplex is then defined as a locus of points, located "in-between" the vertices, i.e. represented as an affine combination of the vertices with non-negative coefficients:

$$\sigma = \left\{ \alpha_0 v_0 + \alpha_1 v_1 + \cdots + \alpha_k v_k \middle| \sum_{i=0}^{k} \alpha_i = 1; \alpha_i \geq 0 \text{ for } i = 0..k \right\}$$

Note that a subset of σ in which some of the coefficients α_i are set to 0 (in other words, only some subset of vertices is used) is also simplex, however of a lower dimensionality. Such simplex is called a *face* of σ. For example, vertices and edges together with their end-points are the faces of a triangle. Each k-simplex would have $(2^k - 1)$ faces, including the empty set and excluding the whole simplex itself.

Figure 3.1 shows examples of simplices for dimensions 0-4.

A *geometric simplicial complex* $\mathcal{K}$ is a set of simplices which follows two additional properties:



(a) $k = 0$: a vertex | (b) $k = 1$: a line segment | (c) $k = 2$: a triangle | (d) $k = 3$: a tetrahedron

Figure 3.1: Simplex examples.

1. If a simplex belongs to some complex $\mathcal{K}$, then all of its faces also belong to the complex $\mathcal{K}$. For example, if some triangle lies in the complex, then all 3 its edges and all 3 vertices also belong to the complex.

2. If two simplices both belong to some complex, then their intersection is either empty, or a face of both simplices. A small example of how this rule could be violated is shown on Figure 3.2.

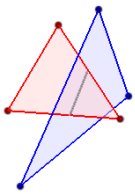


Figure 3.2: Inadmissible simplex intersection. Two triangles intersect by a segment which does not belong to face sets of those simplices.

The dimensionality of a geometric simplicial complex is defined the same way as for an abstract simplicial complex. A 1-dimensional geometric simplicial complex consists only of vertices and non-intersecting edges connecting them, which is a geometric representation of a regular non-oriented graph; likewise, the same is true for 1-dimensional abstract simplicial complexes, only that the latter describes a graph as a pair of sets (V, E). Additionally, simplicial complexes can be seen as generalization of graphs: if the number of dimensions in the complex is higher than 1 we get a hypergraph, and each simplex with $dim \geq 1$ denotes a hyperedge – an edge that connects several vertices simultaneously.

The following subsection illustrates some common ways to construct simplicial complexes over a point cloud in metric space.

3.2.2 Vietoris-Rips and Alpha Complexes

Vietoris-Rips (VR) and alpha complexes are two models which are used to build simplicial complexes given a set of points and a non-negative radius parameter. Although alpha complexes are more important for this work, it is better to start with the definition of the VR complex, since its definition is simpler.

Consider a set S which consists of points in a metric space. A Vietoris-Rips complex with parameter δ is defined as follows:

$$VR_\delta(S) = \{\sigma \subseteq S | \operatorname{diam} \sigma \leq 2\delta\} \tag{3.1}$$

The VR complex is an example of an abstract simplicial complex. Simplices of the VR complex are the subsets of the initial point set, whose diameters are not greater than 2δ. It is easy to see that VR_0 consists only of $|S|$ 0-simplices (each simplex representing a point from the set); on the opposite side, when δ is close to infinity, $VR_\delta(S)) = 2^S$ – set of all subsets of S.

There is another way to visually represent how VR complexes are constructed. Imagine placing closed balls $B_\delta[s_i]$ of radius δ centered around every point s_i. $B_\delta[s_i] = \{x \in \mathbb{R}^n | \|x - s_i\| \leq \delta\}$. Then, a simplex $\sigma = \{s_{i_1}, \ldots, s_{i_k}\}$ belongs to the VR complex if and only if the pairwise intersection of the corresponding balls is non-empty: $\forall 1 \leq j_1 < j_2 \leq k B_\delta[s_{i_{j_1}}] \cap B_\delta[s_{i_{j_2}}] \neq \varnothing$.

Figure 3.3 has an example of a VR complex built on points from $\mathbb{R}^2$ with $\delta = 2.12$. Although in standard case such complex might contain simplices of dimension up to $(|S| - 1)$, only 0-, 1- and 2-simplices are shown here for better readability. Circles around the points have the purpose of helping to understand how the simplices are constructed, and the red segment shows the parameter δ, i.e. the radius of the circles.

Before going into details of alpha complexes, a notion of a Voronoi diagram must be established. The Voronoi diagram is defined on a set of generators $S \subset \mathbb{R}^n$ and divides the whole space into subregions called cells, each cell c_i containing one and only one of the generators $s_i \in S$. A cell is defined by:

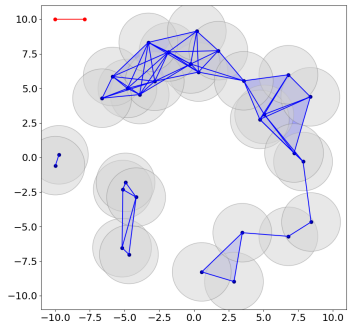


Figure 3.3: An example of a Vietoris-Rips complex.

$$Cell(s_i) = c_i = \{x \in \mathbb{R}^n | \|x - s_i\| \leq \|x - s_j\| \text{ for all } j \neq i\} \tag{3.2}$$

In other words, a Voronoi cell is a locus of points, for which the closest generator is s_i. In Euclidean space each cell c_i is obtained as an intersection of $(|S| - 1)$ half-spaces, bounded by hyperplanes each going through the middle in-between points s_i and s_j $(j \neq i)$ and normal to the vector connecting the points.

The way alpha complexes utilize Voronoi diagrams, is by allowing to connect only those vertices, whose Voronoi cells have a non-empty intersection. Formally, the definition of an alpha complex with parameter α could be written in the following way:

$$A_\alpha(S) = \left\{ \sigma \subseteq S \middle| \bigcap_{s_i \in \sigma} (B_\alpha[s_i] \cap Cell(s_i)) \neq \varnothing \right\} \tag{3.3}$$

Figure 3.4 contains an example for a constructed alpha complex with $\alpha = 2.12$. Gray dashed lines are the boundaries of the Voronoi diagram.

Alpha complexes have several important advantages over Vietoris-Rips complexes.

Firstly, the dimensionality of an alpha complex is the same as the dimensionality of the data, which means that when the analysis is done in 3D, the complex will also have a natural representation in 3D. The number of dimensions in VR complexes is restricted only by the number of points in data. Also, we can see that, unlike Vietoris-Rips complexes, Alpha complexes have direct geometric realization.

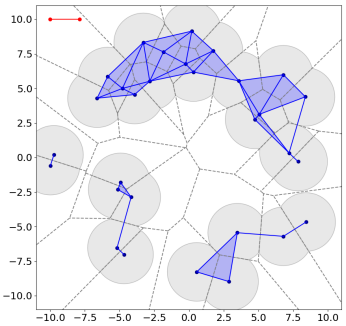


Figure 3.4: An example of an alpha complex.

Secondly, alpha complexes have lower number of simplices for the same radius constant, it is rather trivial that the following inclusion holds: $A_r(S) \subseteq VR_r(S)$. Moreover, if we consider an edge case of $r \to \infty$, then the number of simplices in $VR_\infty(S)$ is equal to $2^{|S|}$, which is extreme for most computational problems. $A_r(S)$ becomes Delaunay complex for sufficiently large r; the name is related to the fact that simplices of the complex basically form Delaunay triangulation of space for given points S. The number of simplices in this complex is estimated as $O(|S|^{\lceil \frac{n}{2} \rceil})$[23] in the worst case. This is a positive side when the work is done in low dimensions, however, it is a lot more difficult to build an alpha complex over a high-dimensional data, due to high computational complexity of Voronoi diagram construction.

3.2.3 Homology Groups

Homology groups $H_k(K)$ for $k \in \mathbb{N}_0$ are abelian groups defined on some topological space K, which in our case is represented as a simplicial complex. Rank (size of group's basis) of a homology group $\operatorname{rank} H_k(K)$ is called a Betti number $b_k(K)$. Each Betti number contains topological information about given space and describes the number of topological features of corresponding dimensionality. These features have good and intuitive geometric interpretations, and informally can be characterized as:

- $b_0(K)$ is the number of connected components in topological space K.

- $b_1(K)$ is the number of "tunnels" in space K. For example, a circle and a cylinder have one tunnel each.
- $b_2(K)$ is the number of enclosed "voids" in space K. For example, a sphere has one void in it.
- $b_k(K)$ with $k > 2$ describe the number of "higher-dimensional holes".

The remaining part of this subsection introduces a mathematical definition of simplicial homology groups (homology groups built on simplicial complexes) $H_k(K, \mathbb{Z}_2)$, which are used in the thesis. $\mathbb{Z}_2$ is the abelian group (binary field) that is used for coefficients in homology groups in the current work.

Consider a simplicial complex K and a non-negative dimension k. A k-chain in the complex is a formal sum of all of its k-simplices with some coefficients and is usually noted as $a = \sum_i c_i \sigma_i$, where $c_i \in \mathbb{Z}_2$ are the coefficients and σ_i are k-simplices in K. Here, coefficients modulo 2 are used, as it is most common in computational topology and simplifies a lot of calculations; however, it is not obligatory, and other groups could be used. Geometrically, a chain can be seen just as a subset of simplices with the given dimension.

Two chains can be added together, and the addition happens componentwise: if $a = \sum_i c_i \sigma_i$ and $b = \sum_i {c'}_i \sigma_i$, then $a + b = \sum_i (c_i + {c'}_i)\sigma_i$. So, if a simplex is present in both chains (i.e., the coefficients before it are equal to 1 in both sums), then the sum of the chains wouldn't contain this simplex.

Together with the addition operation, a set of all k-chains defines an abelian group C_k. The neutral element of the group is a sum of simplices with all coefficients equal to 0, and the inverse operation is the identity function.

Two groups C_k and C_{k-1} are related to each other with a boundary operator $\delta_k : C_k \to C_{k-1}$. For convenience, instead of writing δ_k, δ will be used, since the index of the operator can be deduced from the argument. For a single k-simplex σ, $\delta(\sigma)$ is a sum of all its $(k-1)$-dimensional faces. For instance, the boundary of a triangle is a sum of 3 edges, and the boundary of an edge is a sum of 2 vertices. The boundary of a vertex is always the empty (neutral) element. In order to get the boundary of a chain, boundaries of all its simplices are computed and added with the corresponding indices:

$$\delta(\sum_i c_i\sigma_i) = \sum_i c_i\delta(\sigma_i)$$

In fact, operator δ_k is a homomorphism between C_k and C_{k-1}, since $\delta(a+b) = \delta(a) + \delta(b) \forall a, b \in C_k$.

A k-chain, whose boundary is 0, is called a k-cycle. The set of all k-cycles is denoted as Z_k. If a k-chain is a boundary of some $(k+1)$-chain, then it is called a boundary, and a set of all such boundaries is denoted B_k. Since boundary operator is linear with the addition operation in C_k, then both Z_k and B_k are subgroups of C_k. An important addition would be that every boundary is also a cycle, it is easy to prove that $\delta(\delta(\sigma)) = 0$ for any simplex σ, therefore B_k is a subset of Z_k.

Now, a homology group H_k is defined as a quotient group Z_k/B_k. It means that each element of the group is obtained by taking a cycle from Z_k and adding all boundaries from B_k to it. Two cycles are equivalent (generate the same element in the homology group) if and only if one can be obtained from the other by adding some boundary to it. A figure 3.5 shows an example of two equivalent 1-dimensional cycles and a 2-chain, whose boundary is the difference between them.

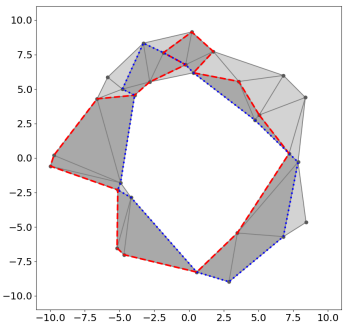


Figure 3.5: An example of two equivalent cycles.
Gray objects are the elements of a simplicial complex K. Red and blue chains are two elements of the cycle group $Z_1(K)$, and the set of dark gray triangles is an element of the chain group $C_2(K)$ whose boundary is the difference between the cycles.
Both cycles represent the same homology class in the group $H_1(K)$.

3.2.4 Persistent Homology

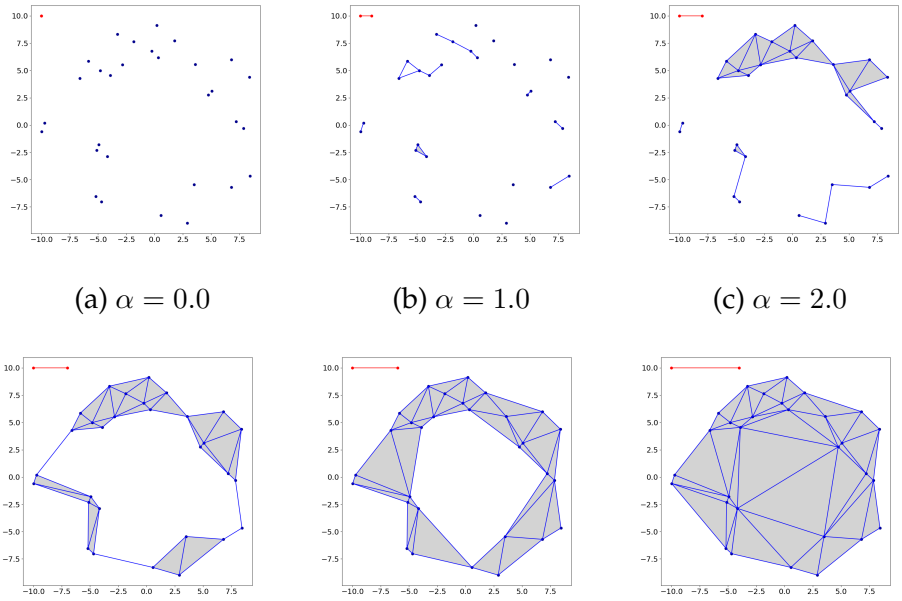
Persistent homology is used to find topological properties of provided data on different spacial scales and to distinguish the most stable and significant among them. Persistent homologies may be used on space which is already represented as a simplicial complex K with a *filtration function* $f : K \to \mathbb{R}$ specified over the simplices of the complex. This function may be considered as a "size", or a diameter, of a simplex and denotes a parameter $\alpha \in \mathbb{R}$ at which the simplex is created. For example, α parameter in an alpha complex can be used to determine filtration values of the simplices, $f(\sigma) = \min\{\alpha \in \mathbb{R} | \sigma \in A_\alpha(S)\}$.

An important rule that a filtration function must follow is that for all faces of a simplex the filtration value must be not greater than for the simplex itself: $\forall \tau \subset \sigma \in K : f(\tau) \leq f(\sigma)$. This leads to the fact that all sets in the form of $K_\alpha = f^{-1}((-\infty, \alpha])$ are also simplicial complexes, and subcomplexes of the complex K. If we vary the α value from $-\infty$ to $+\infty$, picking only the values which appear for at least one of the simplices in the complex K, and record all different subcomplexes that we obtain, we obtain finite series of complexes $\varnothing = K_{-\infty} \subseteq K_{\alpha_1} \subseteq \cdots \subseteq K_{\alpha_n} = K$ with $\alpha_1 < \cdots < \alpha_n$. This is called a *filtration* of the complex K.

Figure 3.6 gives examples of a few subcomplexes taken from the filtration of an alpha complex.

The persistence algorithm finds the segments of filtration values, during which various topological features are present in the complex. It is common to refer to the bounds of these segments as birth and death moments. The larger the segment is, the more persistent and significant the feature is said to be. On Figure 3.6 one can see a large cycle which appears after $\alpha = 2$ and gets filled up only right before $\alpha = 6$; all other features have a lot smaller persistence. This provides us with a hint that the points might have been sampled from a circular surface (or, to be more precise, from a manifold, homeomorphic to a circle). To make this statement more accurate, one could follow the findings from [24] which provide confidence values for similar inferences, with an assumption of points being sampled uniformly from the manifold.

In order to draw conclusions about topological features, persistence diagrams are used. There are several commonly used forms of the diagrams. In the thesis work, persistence is represented with



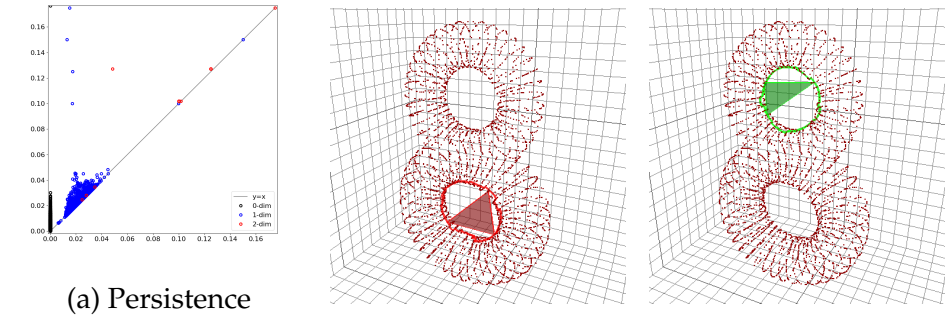
(a) $\alpha = 0.0$ (b) $\alpha = 1.0$ (c) $\alpha = 2.0$

(d) $\alpha = 3.0$ (e) $\alpha = 4.0$ (f) $\alpha = 6.0$

Figure 3.6: 2d alpha complex filtration at different values.

points on a 2d plot. Each point describes one topological feature; its x-coordinate is the birth value and y-coordinate is the death value of the feature. Because of this, all points are located above the main diagonal, and one can define significance of the features as a distance to the main diagonal.

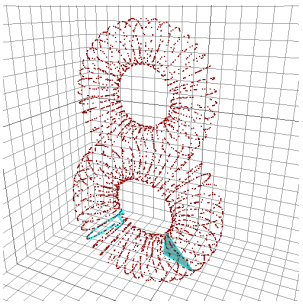
Figure 3.7 contains an example of topological space in a form of double torus, with a number of points sampled from its surface. Its persistence diagram contains 5 distinct features. 4 of those features are tunnels, two of each tori's generating circles. The last feature corresponds to the enclosed void inside of the shape.



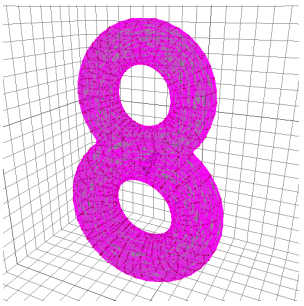
(a) Persistence diagram

(b) H_1 feature at 0.175

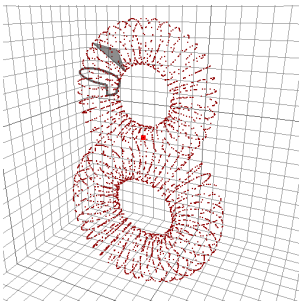
(c) H_1 feature at 0.15



(d) H_1 feature at 0.125



(e) H_2 feature at 0.125



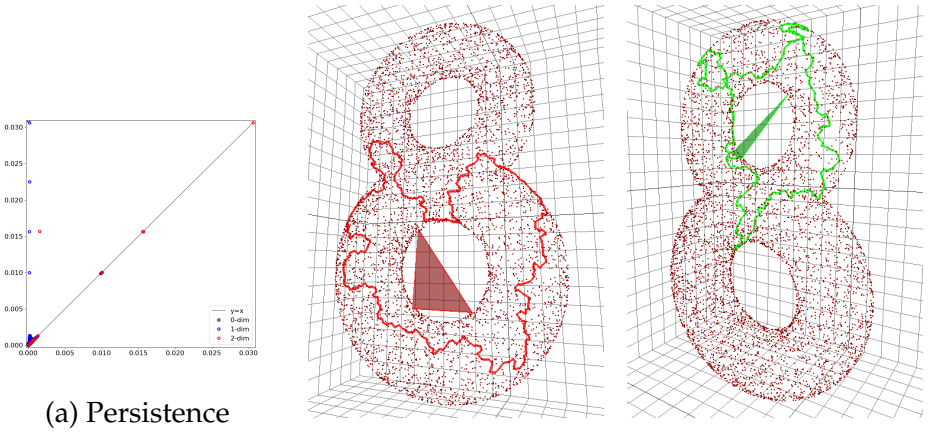
(f) H_1 feature at 0.1

Figure 3.7: Topological features of a double torus.
The diagram on Figure 3.7a contains a point at $(0, +\infty)$, representing an H_0 feature. This feature characterizes the whole connected component of the given shape, and any point of the given point cloud could be a representative for this feature.

A few words must be said about the usual shape of representatives. All cycles on Figure 3.7 look almost perfectly circular, which might not be the case in a different scenario. In this example, points were intentionally generated with a non-uniform distribution: there is a higher concentration of points in the inner holes of the tori, plus each toral part has discrete regions with generated points. Both these qualities ensure that the earliest (in terms of filtration) cycle that appears in the filtration will have a desired shape.

For a similar example with a uniformly-distributed point cloud see Figure 3.8. The persistence diagram for this data looks very similar to the one before, although this time there is a lot less noise. However, the first two representatives for the cycles look more chaotic. Also, if we dive into homology groups theory, the representative on Figure 3.8c is in fact a summation of the 2nd, the 3rd and the 4th H_1 elements

from Figure 3.7, thus the basis of the first homology group appeared to be different in the uniform case (and not as natural from observer's point of view). However, killing simplices are still pointing at the corresponding holes in topological space, and this is the reason why those might be important to look at during the analysis.



(a) Persistence diagram

(b) H_1 feature at 0.175

(c) H_1 feature at 0.15

Figure 3.8: First two topological features of a double torus (uniform distribution of points).

There are several ways to compare two persistence diagrams automatically. Naturally, the comparison can be done via the use of some defined distance function between the diagrams. One of such functions is called *a bottleneck distance*.

Consider a diagram as a set of birth-death points $\mathcal{D} = \{(b_i, d_i) \in \mathbb{R}^2 | i = 1..n\}$. The second diagram $\mathcal{D}'$ is defined in a similar way as the set $\{(b'_i, d'_i) \in \mathbb{R}^2 | i = 1..n'\}$.

Let us pick a real number δ. Define a δ-matching between the diagrams a matching between some points of two diagrams which follows the two rules for all points in the diagrams:

- If a pair of points $(p \in \mathcal{D}, p' \in \mathcal{D}')$ is in the matching, then $dist(p, p') \leq \delta$.
- If a point $p \in (\mathcal{D} \cup \mathcal{D}')$ does not belong to the matching, then $dist(p, c) \leq \delta$, where $c = \{(x, x) | x \in \mathbb{R}\}$ is the main diagonal.

An example matching is presented on Figure 3.9. In order to remove an extra condition about the diagonal, we could also say that

all diagrams naturally include all points on the main diagonal, thus simplifying the definition of a δ-matching.

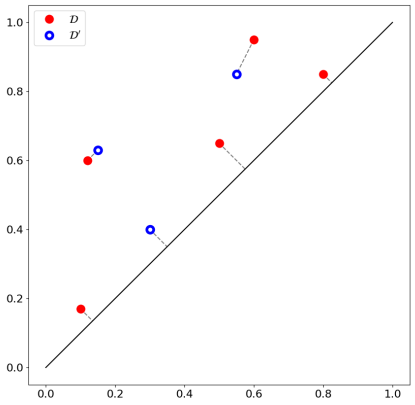


Figure 3.9: An example of a matching between two diagrams. $\delta = 0.1118$ is the minimal value for a δ-matching to exist.

Obviously, for two selected diagrams δ-matchings exist only starting from some non-negative real number δ. Also, 0-matching exists if and only if the diagrams are equal. The bottleneck distance between the diagrams is defined as the smallest δ for which δ-matching between them exists.

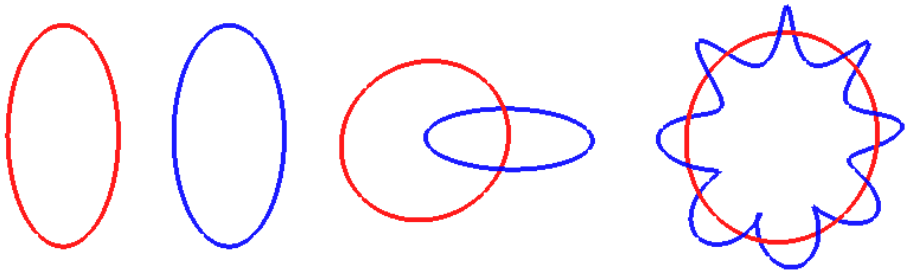
It is important that while all figures with diagrams present persistent pairs from different homology groups, the matching for bottleneck distance must be done only among points from the same homology group. The distance between mixed diagrams can be defined, for example, as a maximum of separate distances for H_0, H_1 and H_2 pairs.

3.2.5 Knot Theory: Linking Number

Knot theory is the part of topology which, in the basic case, studies embeddings of closed curves into 3-dimensional Euclidean space. In this work the concept of linking between curves is used for analysis.

Consider two non-intersecting closed curves ϕ and ψ, embedded into $\mathbb{R}^3$. $\phi, \psi : S^1 \hookrightarrow \mathbb{R}^3$. In simple words, the linking number $LK(\phi, \psi)$ characterizes how many times one curve twists around the other. If the linking number is different from 0, two curves cannot be separated from each other by a homomorphic transformation of one of the

curves without passing through the other curve. However, 0 as a linking number does not guarantee that two curves can be separated. A few examples of linking are shown on Figure 3.10. It is important to note that the linking number is signed, and its sign is dependent on directions of the curves.



(a) $LK(\phi, \psi) = 0$. (b) $LK(\phi, \psi) = 1$. (c) $LK(\phi, \psi) = 8$.

Figure 3.10: Linking number examples (absolute values).

The linking number of two curves can generally be computed by using Gauss's integral formula:

$$LK(\phi, \psi) = (4\pi)^{-1} \int_{S^1 \times S^1} \frac{det(\dot{\phi}(s), \dot{\psi}(t), \phi(s) - \psi(t))}{\|\phi(s) - \psi(t)\|^3} \, ds \, dt \tag{3.4}$$

In the thesis work all curves are represented as polygonal chains, i.e. sequences of connected segments, and are defined by sequences of vertices. Although it is possible to compute the linking number by numerical integration of the formula above, this would require significantly more computational resources than it is in fact needed. For a polygonal chain one could derive an exact formula, which is represented as a sum of $n(\phi) \times n(\psi)$ easy to compute terms. Here $n(\cdot)$ denotes the number of vertices in a chain.

Let us assume that the chain ϕ is represented by a sequence of points $\{A_i\}_{i=0}^{n_\phi}, A_0 = A_{n_\phi}$. Similarly, for ψ it would be a sequence $\{B_i\}_{i=0}^{n_\psi}, B_0 = B_{n_\psi}$. Also, $\phi_i(s)$ and $\psi_j(t)$ for $1 \leq i \leq n_\phi$, $1 \leq j \leq n_\psi$ and $0 \leq s, t \leq 1$ represent reparametrized linear parts of the curves. From here, we can split Gauss's integral into a sum of integrals over smaller segments:

$$\begin{aligned} LK(\phi,\psi) =& (4\pi)^{-1} \sum_{i=1}^{n_\phi} \sum_{j=1}^{n_\psi} \int_{[0,1]^2} \frac{det(\dot{\phi}_i(s), \dot{\psi}_j(t), \phi_i(s) - \psi_j(t))}{\|\phi_i(s) - \psi_j(t)\|^3} \, ds \, dt \\ =& (4\pi)^{-1} \sum_{i=1}^{n_\phi} \sum_{j=1}^{n_\psi} \Omega_{ij} \end{aligned} \tag{3.5}$$

Each of the integrals Ω_{ij} can be computed directly using the formulas, derived in [25] and [26], using the following equation:

$$\begin{aligned} \Omega_{ij} =& sgn(a(\overrightarrow{B_jB_{j+1}}, \overrightarrow{A_iA_{i+1}}, \overrightarrow{A_iB_j})) \cdot \\ & \cdot Q(\overrightarrow{A_iB_j}, \overrightarrow{A_iB_{j+1}}, \overrightarrow{A_{i+1}B_{j+1}}, \overrightarrow{A_{i+1}B_j}) \end{aligned} \tag{3.6}$$

$a(\overrightarrow{a_1}, \overrightarrow{a_2}, \overrightarrow{a_3})$ is a signed volume of a parallelepiped defined by three given vectors. Its calculation simply equals $(\overrightarrow{a_1} \times \overrightarrow{a_2}) \cdot \overrightarrow{a_3}$, where $\times$ is a vector product and $\cdot$ is a scalar product.

$Q(\overrightarrow{a_1}, \overrightarrow{a_2}, \overrightarrow{a_3}, \overrightarrow{a_4})$ is an area of a spherical quadrangle defined on a unit sphere, whose vertices are formed as intersections of the sphere with given vectors. It can be calculated by the following formulas:

$$\overrightarrow{n_i} = \frac{\overrightarrow{a_i} \times \overrightarrow{a_{i+1}}}{\|\overrightarrow{a_i} \times \overrightarrow{a_{i+1}}\|}, i = 1..4 \tag{3.7}$$

$$Q(\overrightarrow{a_1}, \overrightarrow{a_2}, \overrightarrow{a_3}, \overrightarrow{a_4}) = \sum_{i=1}^{4} arcsin(\overrightarrow{n_i}\overrightarrow{n_{i+1}}) \tag{3.8}$$

In equations 3.7 and 3.8 addition in indices is cyclic over the numbers $1..4$, i.e. notation like a_{i+1} should in fact be written as $a_{(i \bmod 4+1)}$.

Chapter 4

Implementation Details

4.1 Algorithms

There are many machine learning libraries which are thoroughly used at the moment. In this work `tensorflow` library[27] was used, because it provides a large amount of freedom and control in the implementation of machine learning algorithms, which could be especially useful since this study focuses more on the analysis of networks than on the training itself.

All methods of constructing the networks are standard and make use of default techniques available in the library. The only thing which at the moment is missing in the library in an easily accessible way are deconvolutional (transpose convolutional) layers with unpooling. For deconvolution the regular convolutional layers were used (`tf.layers.conv2d`), since it is essentially the same. Unpooling was modelled with image resizing method `tf.image.resize_images(·, method=tf.image.ResizeMethod.NEAREST_NEIGHBOR)`.

While everything related to machine learning is done on Python, computational topology algorithms are written using C++. The parts were connected with batch-scripts, which were also used to run several iterations of experiments.

Several libraries were considered for the construction of alpha complexes. On the first iteration, alpha shapes from `CGAL` library[28] were used. However, this approach had two major problems. One of them was that `CGAL` works only with 2- and 3-dimensional complexes, making it difficult to extend the research to higher dimensions with this

library. Moreover, $2D$ and $3D$ algorithms had many differences in the interface of usable functions. Another problem was lying in inner structures of the algorithms. It was often impossible to extract some needed additional information from the complexes. The code for alpha shapes was written in a such way, that while the algorithms worked correctly, due to various optimizations there was no guarantee of correct inner representations that are not documented or not designed for public access. For example, `filtration()` parameter of a simplex in `Alpha_shape_3` does not return the correct filtration value; often it is either equal to 0 if its computation was unnecessary, or returns some auxiliary value which happens to be a filtration value of some other related simplex.

Fortunately, a rather new library `Gudhi`[29], which is built on top of `CGAL`, doesn't suffer from the first problem, accepting the number of dimensions as a template parameter. It however still has the second problem, but this time it was easily solved by forcing some of the calculations manually. Also, `Gudhi` is a lot more powerful for topological computations, containing many other data structures for simplicial complexes (Vietoris-Rips complex was also used from this library) and some other algorithms, including bottleneck distance between persistence diagrams. So, after all, the `Gudhi` library was used to build alpha complexes on points and to compute filtration on them.

For part of the experiments we also used the `Gudhi` to compute persistent homologies. However, because some changes were later required for these algorithms, combined with high complexity of the source code of `Gudhi` which follows the complexity of `CGAL` being written fully on templates, a reduction algorithm from `Phat` toolbox[30] was applied and later modified with few additional computations. `Phat` library contains several reduction algorithms, but only implementation of the standard algorithm was used in the thesis work. `Phat`'s version of this algorithm is an efficient implementation of the matrix reduction algorithm described in [31].

The purpose of the initial version of the algorithm is to apply reduction to the given boundary matrix in order to extract all persistence pairs from it later. Unfortunately, it is only possible to extract birth and death times of homology classes from the matrix. One might be interested in obtaining a representative element of a homology class which corresponds to a chosen persistence pair.

Representatives can be easily extracted with an injection of several

lines of code into the existing algorithm. More precisely, an additional matrix R is created and initialized as an identity matrix. After that, all the operations which are applied to the boundary matrix (operations of column addition) are also applied in the same way to the matrix R. This modification is represented in the Algorithm 1, highlighted lines are the additions to the standard algorithm.

Algorithm 1 Modified standard reduction algorithm.

Input:
Boundary matrix $B \in \mathbb{Z}_2^{n \times n}$
Output:
Reduced boundary matrix B
Representative matrix $R \in \mathbb{Z}_2^{n \times n}$
1: $low(i) \leftarrow$ largest index j such that $B_{j,i} = 1$
2: $R \leftarrow \mathbb{I}_n$
3: **for** $i = 1 \rightarrow n$ **do**
4: **while** $\exists i_0 < i$ such that $low(i_0) = low(i)$ **do**
5: $b_i \leftarrow b_i + b_{i_0}$ ▷ b_k denotes kth column of B
6: $r_i \leftarrow r_i + r_{i_0}$ ▷ r_k denotes kth column of R
7: **end while**
8: **end for**

Extraction of information from the matrices is also slightly different from the algorithm in `Phat` toolbox and was rewritten. The algorithm normally seeks the columns in the reduced matrix which contain non-zero values. For each such column a candidate to persistence pairs is a pair with birth at the index of the lowest row with a number 1 in this column, and with death at the index of the column. It can be noted that after reduction the matrix is always strictly upper triangular, thus the birth index is always smaller than the death index. The main algorithm immediately adds such pair to the resulting list, but in this work we have access to the filtration values of the simplices (usually α-values from Alpha complexes), and if the birth and the death simplices have the same filtration value, then the pair is located on the diagonal and can be immediately omitted.

Another change is related to computation of representatives. They are obtained easily from the representative matrix: all rows with non-zero values in a column, for which a pair was found, altogether a set of simplices which form a representative for this pair.

Visual representation of all changes can be seen in Algorithm 2 with

highlighted lines.

Algorithm 2 Modified computation of persistence pairs.

Input:
Reduced boundary matrix $B \in \mathbb{Z}_2^{n \times n}$
Representative matrix $R \in \mathbb{Z}_2^{n \times n}$
Output:
List of persistence pairs $\mathcal{P} = \{(b_i, d_i)\}_{i=1}^{m}$
▷ b_j, d_j are indices of simplices corresponding to birth and death of a pair j
List of representatives $\mathcal{R} = \{(s_0, ..., s_{dim_i})\}_{i=1}^{m}$ ▷ s_j are indices of simplices

```
low(i) ← largest index j such that B_{j,i} = 1
filtration(s_i) ← filtration value of a simplex s_i
P ← ∅
R ← ∅
for i = 1 → n do
    if low(i) exists then
        b_i ← low(i), d_i ← i
        if filtration(b_i) ≠ filtration(d_i) then
            Add (b_i, d_i) to P
            Add the representative {s_j | R_{s_j,i} = 1} to R
        end if
    end if
end for
```

Although handling of persistence pairs with infinite death time was also implemented, it was never needed for the analysis in the thesis work, thus its implementation is omitted in the report to avoid unnecessary complications. Later in figures with persistence diagrams one can see such points, which touch the upper border of the plot. Each diagram always has at least one such point, which corresponds to an H_0 feature - the whole connected component.

4.2 Visualization

Most of the experiments which are conducted in the current work are strongly tied to the analysis of 3-dimensional data. Therefore, it becomes possible and beneficial to visualize the data in some environment to manually verify the correctness of topological derivations.

At first, all data was visualized using only `matplotlib` libraries, including 3d-projections of scatter plots from the library. However, due to the lack of necessary features and performance issues on large

amounts of data, it was decided to develop own plotting application on some 3D engine.

The first prototype was made on Python with `PyQtGraph` library. This library was chosen because it was developed with intentions for scientific use. The prototype was performing rather well in the beginning, however, the library's main direction of application was 2D-data, and it was noticeable that its 3D system was still early in the development.

After that it was decided to switch to a different visualization method once again. The choice fell on JavaScript and `three.js` 3D library, being defined mainly by the intention to make the application embeddable into a browser page.

The visualization tool is written on the basic level of WebGL, meaning that only primitives such as points, segments and polygonal meshes are manipulated. The application allows to represent more than a million points and same size H_1 features, which are represented by segmented lines, without any drops in performance on GeForce GTX 960.

The main features of the tool are the following:

- The core part of the visualization are points clouds, which are representing given datasets. Since there usually are several datasets under investigation, it is possible to switch between different scenes. At the same time, once scene can contain points with different labels, coloring all points differently, based on that principle.

- Representatives of topological features are visualized using closed polygonal chains (H_1) and polygon meshes (H_2). Usually there are several features that are represented in a dataset, so for analysis it is possible to hide some of the representatives.

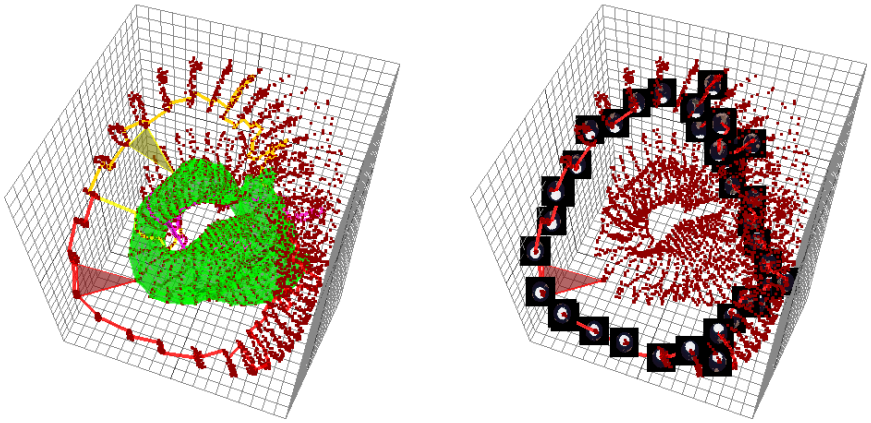
- Each representative is paired with a "killing" simplex – the first simplex which signifies death of the given topological feature in the filtration. These simplices are also depicted in the diagram, either as triangles for H_1 features, or as tetrahedra for H_2 features.

- Normally, each data point correspond to an image from an initial dataset. To see these images one can hover the pointer over a

point, and the closest one to the camera will be selected. Also, it is possible to show all images that belong to vertices on a selected topological feature representative; this way one can analyze the structure of observed cycle by looking at all its images at once.

Images are loaded asynchronously and do not affect the performance in case of large volume of files.

- If the persistence diagram is available for the data, it is shown in the corner and can be enlarged by hovering over it to see it closer.



(a) (b)

Figure 4.1: An example of data visualization.
Figure on the left shows four most persistent cycles in the depicted data. Green cycle is an H_2 feature and represents the void, while all other are H_1 features.
Figure on the right places focus on one cycle only, while also providing previews of images for the data on the cycle.

Chapter 5

Methods and Experiments

5.1 Finding Hidden Symmetries in the Data

Consider a trained multi-layer artificial neural network with some chosen fixed layer l. Let us look at the part of the network which consists of all the layers starting from the input layer and ending with the layer l, which can be viewed as the function $f_l : \mathbb{R}^{d_i} \to \mathbb{R}^{d_l}$, where d_i and d_l are the numbers of neurons in the input layer and layer l correspondingly. If we now consider $f_l(\mathcal{D})$, where $\mathcal{D} \subset \mathbb{R}^{d_i}$ is the dataset for the neural network we have, we obtain a point cloud in d_l-dimensional space. The properties of the point cloud might reveal some information about the initial data which the network was able to detect, as well as some properties about the network itself, for example how well it is trained.

In all tests which analyze the dataset representation in one of the hidden layers of the network, the dimensionality d_l of the chosen layer never exceeded 5. The cause for that lies in the chosen topological method of constructing alpha complexes over the point cloud. From Delanay triangulation properties it follows that the number of simplices in an alpha complex can be estimated as $O(N^{\lceil \frac{d_l}{2} \rceil})$, where N is the size of the point cloud, i.e. the number of elements in the dataset. Since used memory and time resources grow exponentially with the number of dimensions in the data, it is impossible to use higher dimensions without significant changes to the design of used algorithms. Moreover, most of the tests use $d_l = 3$ for the possibility of a simple visual representation of obtained point clouds.

In the first test we use the unmodified MNIST dataset, consisting of

60000 (10000 in the test dataset) 28×28 grayscale images of handwritten digits. The first neural network architecture is also rather simple – it is an autoencoder which consists of six hidden fully connected layers with the following numbers of neurons in-between:

$$(28 \times 28) \to 100 \to 20 \to \mathbf{3} \to 20 \to 100 \to (28 \times 28)$$

The highlighted layer is the one that was chosen for analysis. It is also the layer which gives us the result of encoder's operation. Function $tanh$ was used for activation between layers, and no additional regularizations or normalizations were used for this case. A batch of 128 images was passed on each training iteration, with the total number of iterations equal to 250000. These networks (as well as all other networks in the experiments section) were trained using `AdamOptimizer` algorithm with initial learning rate 0.0001.

Due to simplicity of both data and the networks themselves it was possible to train 100 autoencoders with the same structure, but with different initialization values. All autoencoders show good encoding-decoding results, as it can be seen on the Figure 5.1. The average losses on train and test datasets are 12.41 and 13.20, with standard deviations equal to 0.39 and 0.56 correspondingly.

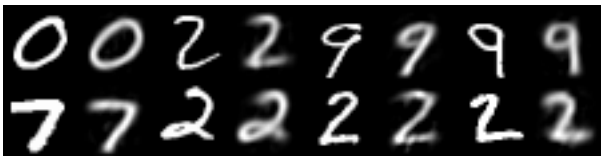
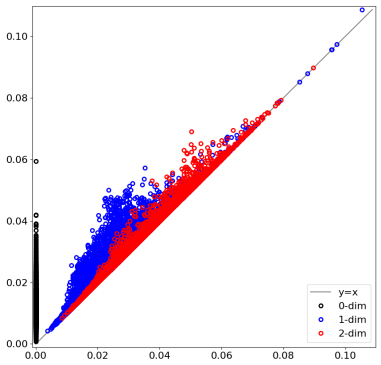


Figure 5.1: An example of digits and their corresponding outputs of the autoencoders.

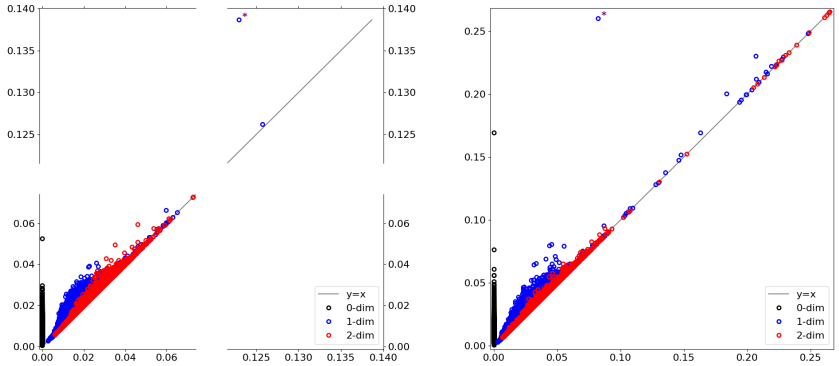
Next, for each of the trained autoencoders and for each of the labels from 0 to 9 a 3-dimensional point cloud of the images' encodings was analyzed using persistent homologies on alpha complexes. Also, *train* and *test* datasets were considered separately, so for each autoencoder 20 diagrams were constructed. The goal was to find versions of autoencoders which produce distinct topological features in the data during encoding process.

Figure 5.2 displays some of the diagrams. Most of the diagrams did not contain any noticeable features on them, although a few had persistence pairs from the group H_1 which were far from the main diagonal with distance it being more than 5 times larger than from

all other points, which could potentially mean that some interesting structure was found in the data.



(a) Regular example of a diagram.



(b) Diagrams with distinct features (marked with an asterisk).

Figure 5.2: An example of digits and their corresponding outputs of the autoencoders.

Unfortunately, after manual inspection of point clouds it turned out that all those H_1 features appeared to be created by noise and outliers in the data, whose encoding is located separately from the main clusters of other points. Figure 5.3 shows two examples of such behavior. These are the examples that correspond to the diagrams from Figure 5.2b.

The image on the left shows an example, when a cycle is obtained by a slightly curved point cluster enclosed by several outlying points.

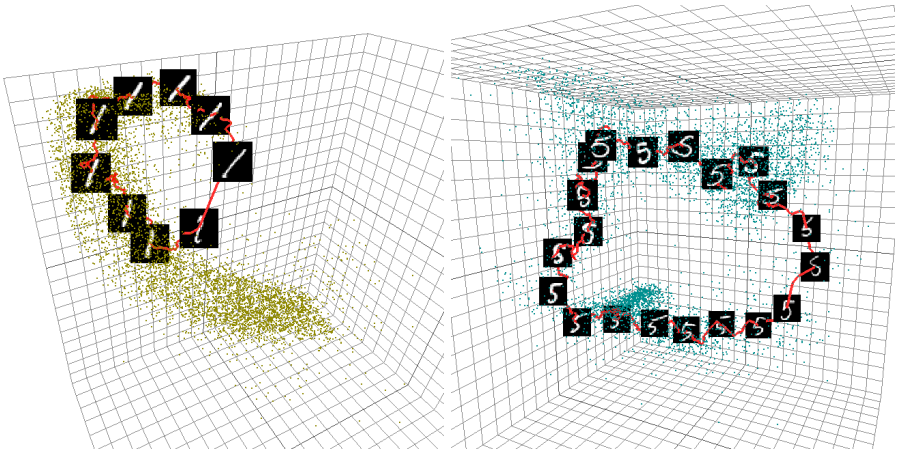


Figure 5.3: Examples of the most persistent features found in MNIST.

This is the most commonly occurring example of a visually highly persistent H_1 feature, which in fact does not look correct. The right image is more believable to represent a good circular structure. Although, if looked more closely, and especially in a 3D-view, one can see that in fact there are just two oblong clusters, by chance connected together by several outliers. There is also no clear rotational symmetry, although that of course might just be something that is difficult for a human eye to see.

Another witness to the fact that those features on the diagram were just the product of outliers in the data was the large difference between the diagrams for *train* and *test* dataset for the same autoencoders and labels. In all cases, if a *train* diagram contained one clear outlier, *test* diagram didn't have any outliers, and vice versa. This may indicate that the network was overfitted; however, it could not be the case for the networks with an architecture with so few neurons. So, perhaps the only explanation left is that the feature present on the first diagram is just there by random.

Clearly, the problem here is that topological features are quite sensitive to outliers. One way to cope with that is to remove some percentage of points with the lowest density. This approach is used in later experiments, however the negative side is that it introduces a hyperparameter to the problem, which contradicts with one of the main ideas of persistent homologies. Another solution would be to use filtering on the density of the points[32] and to construct persistent homologies based on that, but that would either restrain us from using

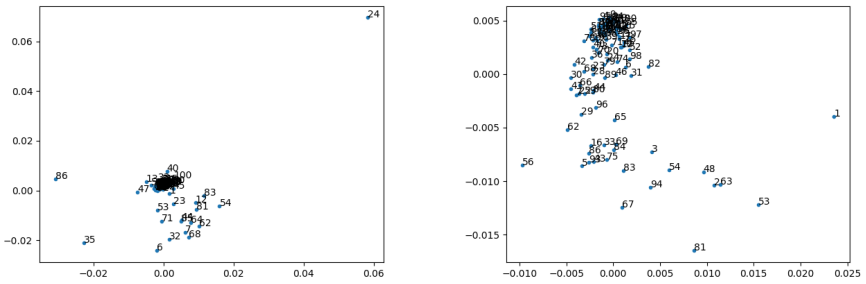
alpha complexes or require us to use multi-dimensional persistent homologies[33] based on two filtration parameters, which at the moment are extremely computationally expensive.

Another question that could be raised here is about finding distinct autoencoders among all 100 models without the need to go through all persistence diagrams manually. In order to do that, for a given dataset one could estimate pairwise distances between all diagrams and look at the clusters these diagrams are organized into.

The bottleneck distance was used to calculate distances between diagrams. Since it takes a considerable amount of time to calculate all distances with all points, all insignificant persistence pairs with lifespan below 0.0001 were removed from the diagrams, reducing the total number of points in each diagram by 5 times on average and significantly improving the performance.

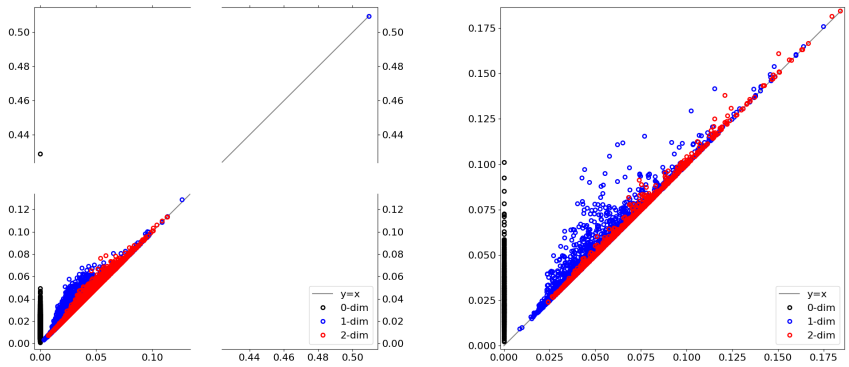
After all distances were calculated, all diagrams were embedding into 2D as points using the Multidimensional Scaling algorithm[34].

Results showed that the diagrams always form only one cluster, with one or few points lying far from the cluster center. Those diagrams could be considered significant, if only the same diagrams on the second dataset (between *train* and *test*) weren't lying in the center of the cluster, showing once again the randomness of persistent pairs. Figure 5.4 shows an example of such embedding, and Figure 5.5 compares two persistence diagrams of the 24th autoencoder for the digit 0.



(a) Train. (b) Test.

Figure 5.4: Diagram embeddings for label 0. Diagram $\#24$ is placed very far from the cluster on train datasets, but on test datasets it is somewhere in the center.



(a) Train. (b) Test.

Figure 5.5: Diagrams for the 24th autoencoder for label 0 from train and test datasets. 5.5a contains a distinct H_0 feature, which in reality is just one data point located significantly far from all other points.

A few last tests with regular MNIST were conducted on classifiers. The idea was to make the last hidden layer before the output 3-dimensional and to have a look at data representation at that layer. The only thing that was revealed is that such classifiers tend to organize 10 clusters far from each other, one for each label, at that layer. However, no structure could be found inside each cluster, all points just tend to be pushed as close into the center of the cluster as possible. A reasonable explanation for it would be that classifiers do not need to preserve any information about the initial images apart from their class, especially on a layer so close to classification output.

5.2 Analysis of MNIST with Rotational Symmetries

Since no rotational symmetries were noticed in the work of autoencoders with original MNIST data, the next step was to apply the algorithm to a dataset which is expected to contain some kind of symmetry with higher certainty, and to check whether neural networks are detecting and keeping those symmetries. For that purpose a new dataset was constructed by taking every image from the MNIST dataset and rotating it to 16 different angles, uniformly distributed on the interval

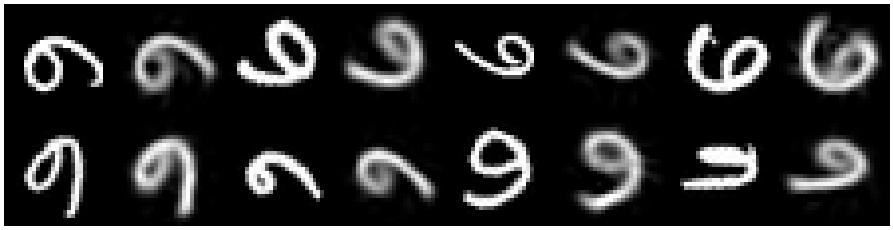
$[0; 2\pi)$.

For the first test, only images with the digit 6 were left. With this, two similar types of autoencoders were trained, 30 in one group and 10 in another. They all also has a simple structure, with only 5 fully connected layers with the following number of neurons in the layers:

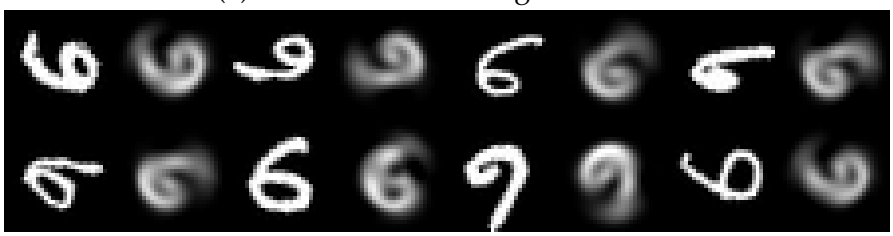
$$(28 \times 28) \to 100 \to 40 \to \mathbf{3} \to 40 \to 100 \to (28 \times 28)$$

The only difference between two types of autoencoders was that L_2-regularization was applied to the weights of the layers of the second group with the coefficient 0.1. On Figure 5.6 we can see the comparison of decoding for the autoencoders, with regularized autoencoders performing noticeably worse.

All networks were trained for 350000 iterations with 128 images in each batch on every iteration. The average L_2-loss of non-regularized networks was equal to 10.58 on train and 11.34 on test data, with standard deviations 0.39 and 0.33. Regularized autoencoders were providing average losses 17.31 and 17.69 with standard deviations 0.28 and 0.48 (plus additional 3.38 regularization loss on average).



(a) Network without regularization.

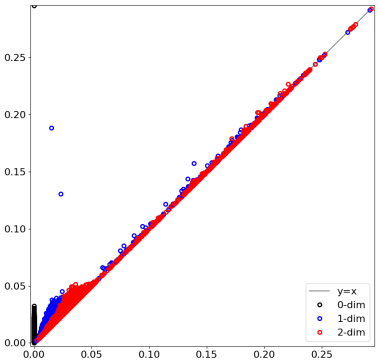


(b) Network with regularizing coefficient 0.1.

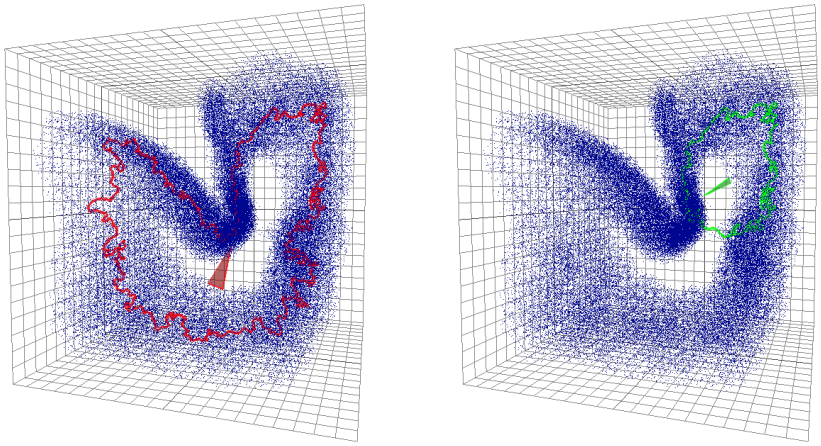
Figure 5.6: An example of rotating images with the digit 6 and corresponding outputs of the autoencoders.

However, the difference between the two groups does not only lie in the quality of decoding. Absolutely all of 30 trained non-regularized autoencoders were finding and keeping the circular structure of the

data. This can be seen on both the persistence diagrams and the 3D-views of the point clouds of the encodings. This can be seen on Figure 5.7.



(a) Persistence diagram.

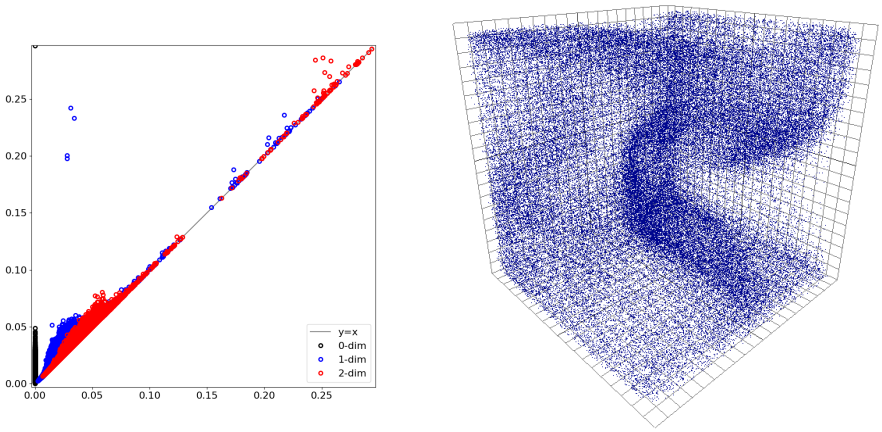


(b) 3D-view.

Figure 5.7: Label 6 rotation, non-regularized case.
In this case persistence diagram has two interesting points, however as it can be seen from 3D-views only the first one is significant.

The situation is different with regularized encoders – the circular structure of data is not visibly preserved in the trained networks. Figure 5.8 contains a typical diagram and a 3D-view of the points for the networks in this group.

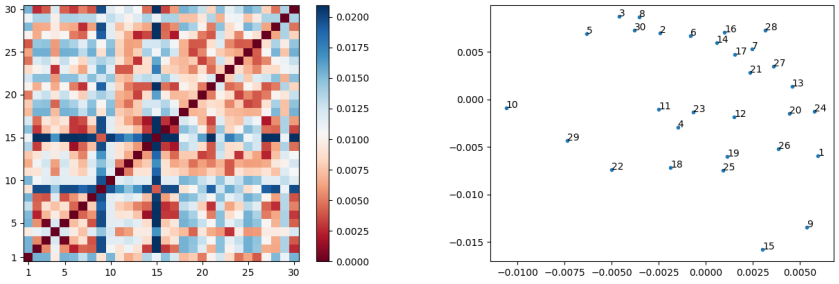
We can calculate pairwise distances between the estimated persistence diagrams. Figures 5.9 and A.3 contain results and analysis for



(a) Persistence diagram. (b) 3D-view.

Figure 5.8: Label 6 rotation, regularized case.
The diagram shows four significant H_1 features and several H_2 pairs far from the main diagonal.

regularized and non-regularized autoencoders. We observe that, in considered cases, it can be concluded that highly regularized autoencoders are less persistent in training compared to those without regularization.



(a) Distance matrix. (b) 2D embedding.

Figure 5.9: Different autoencoders without regularization. From the plots one could see that two of them – $\#9$ and $\#15$ form their own little cluster, having small distance between each other and larger distance to others. In fact, all 30 of those representations are very similar.

It is important to note that a neural network with fully connected layers and without convolutions sees an input images just as an vector

of real values. When a rotation is applied to an image, if the rotation angle is not too small, it basically means some permutation of values in the input vector. Therefore, the Euclidean distance between two images in initial input space changes drastically even between two images when one is obtained by rotation of the other (unless the rotation is small enough, which is not the case with the rotation angle $\frac{\pi}{8}$, see Figure 5.10).

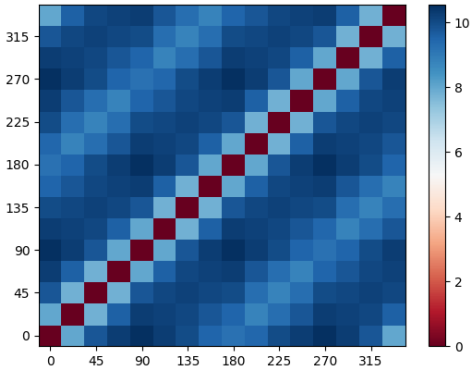


Figure 5.10: Average istances between images with digits, rotated to 16 different angles. Neighboring images are just slightly closer to each other than to other images. That means that either networks are able to detect these little differences in distance in a high-dimensional space, or they capture the rotation in another way.

Nevertheless, we can try to see if there is any detectable topological structure in the initial space of rotated images. Since the space is 784-dimensional, we cannot use alpha complexes. Instead, we can construct Vietoris-Rips (VR) complexes for the first two homology groups.

There could be different approaches which would let us use VR complexes. One of them would be about finding a proper constant to limit the maximum distance for the edges in the complex, thus potentially decreasing the number of simplices significantly. Unfortunately, it is rather difficult to apply such method for the current problem, because the distance required to find a cycle seems to be close to the diameter of the dataset, thus not providing any significant advantage.

The other approach is rather trivial: reduce the amount of data points. Following this idea, the amount of images used to construct the complex was set to 240 (also, only images with the digit 3 were

used). On top of that, 5% of the data was filtered out by distance-to-measure, leaving us with 228 images and almost 2 million simplices in the complex.

Interestingly enough, the VR complex-based approach was able to detect an H_1 feature, as it can be seen on the diagram on Figure 5.11, even though the significance of this feature might not seem very noticeable. If we look at a representative of the main cycle, we indeed get rotating images (see Figure 5.12).

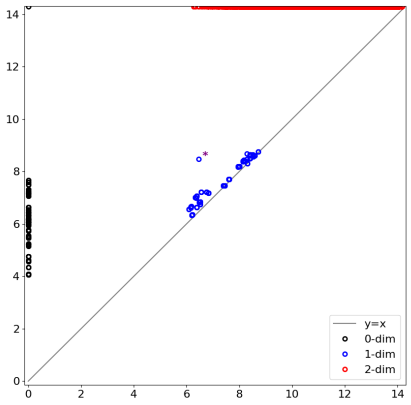
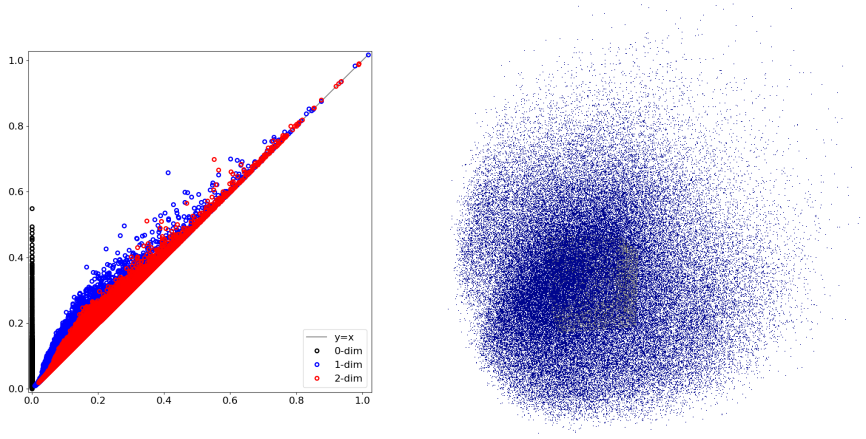


Figure 5.11: Persistence diagram for Vietoris-Rips complex on images with rotating images with the digit 6.

Figure 5.12: A representative of the most persistent cycle in Vietoris-Rips complex from Figure 5.11.

Let us step a little bit aside from neural networks and have a look at a different type of dimensionality reduction, Principal Component Analysis (PCA). To compare it with the performance of autoencoders, we simply run the projection of the input space on three main components. Persistence diagram and 3D-view with points are presented on Figure 5.13.

As we can see from the figure, PCA does not keep a cyclic structure, and the point cloud's distribution has a bell shape. Even though the cycle exists in the initial space, probably due to high dimensionality and the "noisiness" of data (the cycle is represented not with a single



(a) Persistence diagram. (b) 3D-view.

Figure 5.13: Label 6 rotation. PCA results.

curve, but with a more complex shape) it's not possible to project the cycle orthogonally into a low-dimensional space and to keep its structure at the same time without large amount of overlapping.

After training on one label only, a natural extension of the problem is to look at the neural network behavior on more complex data. Now we use all labels from MNIST, each image is rotated in 16 steps from 0 to 2π.

Since this becomes a considerably more complex task, and new visual results were not seen as passable with an old network structure, a small update to the autoencoder architecture was applied. For the current task the structure of the network is depicted on Figure 5.14. Now the network contains convolutional and pooling layers, normalization and L_2 regularization (with coefficient 0.1) on kernel parameters, as well as both `relu` and `tanh` activations. The network was trained for 1000000 iterations with batch size of 128 images.

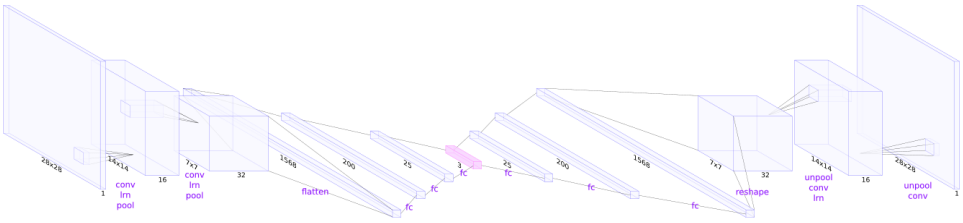


Figure 5.14: Autoencoder structure for MNIST with rotations.

Having 3 dimensions makes it more difficult for an autoencoder to learn a good compression of the given data, compared to one-labeled data that was used before, see Figure 5.15. One can see that the network is starting to lose the differences between distinct digits. However, from the output images it can be inferred that the many lines of the digits still get restored by the decoder. Total loss approaches the value 17.6, with main loss part being centered around 14.9 (14.8 on train data).

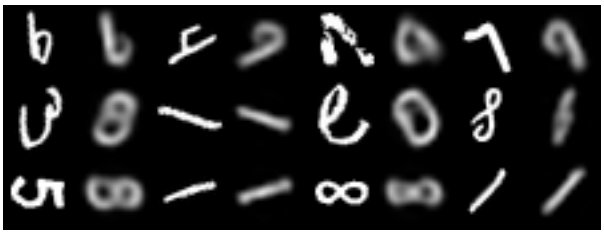


Figure 5.15: An example of rotating images with digits and corresponding outputs of the autoencoder.

At the same time, it is also quite interesting to see whether the autoencoder still captures the rotations. The persistence diagrams are presented on Figure A.1 for all labels from 0 to 9.

Clearly, the digit 0 doesn't have any detected H_1 features, since it gives almost the same image under every angle. Also, we can see that the digits 2 and 4 do not have any prominent features, perhaps, due to complexity of the symbol and also high variety in how they can be written. However, all other digits have some very distinct persistence pairs in their diagrams.

Being that we still work with a 3-dimensional space, we can look at the appearance of the point clouds (Figure A.2).

From the views we can see how well the rotations are captured, especially for the digit 1. Also interesting to note the complexity of cycles for digits 9. The same shape is taken by digits 6 (by obvious reasons) and digits 7. Another interesting things to note is that the network does not distinguish well images with number 3 from their copies rotated upside down, this specificity can be visible on Figure A.2b.

Now let us have a look at the linking between different curves in the data. An image's curve (cycle) can be defined as a cyclic sequence that is formed by encodings of the image and all of its rotations, which

are present in the dataset.

From the whole MNIST test dataset we can obtain 10000 curves, 16 points in each, 1000 curves for each label. For each pair of the curves their linking number can be computed. After sorting the curves by the label of the generating image, we can present all linking numbers as a matrix, which is depicted with a colormap on Figure 5.16.

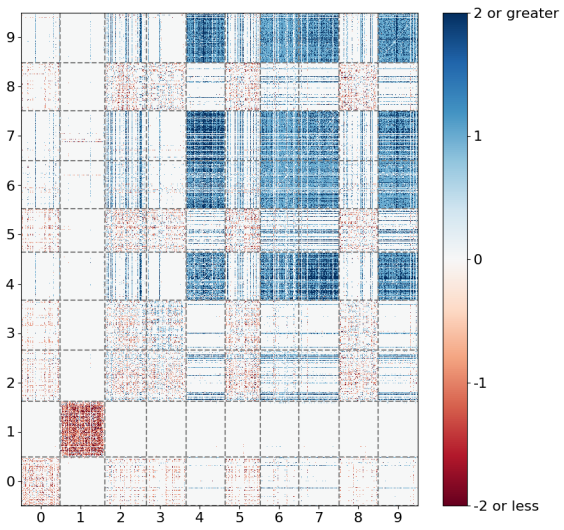
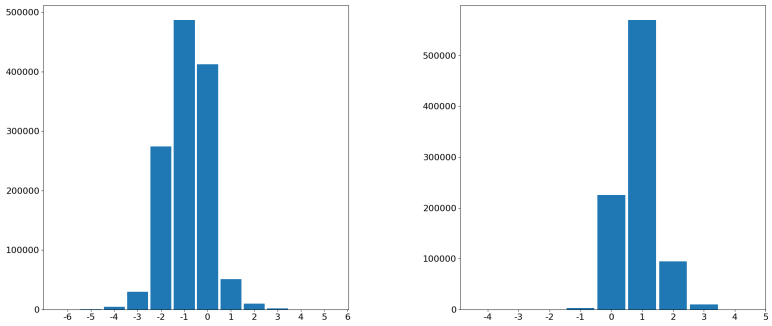


Figure 5.16: Matrix of linking numbers, grouped by image classes.

We observe non-zero linking values between curves of the same label – in R^3 their vertices are located close to each other and organize similar patterns, and high linkage can be expected between curves that are obtained from small perturbations of each other (which sounds similar to our case). What, however, looks surprising, is that one sign of the linking always dominates in each block. If we look closer at some of the distributions of those numbers (see Figure 5.17), we can see that about two thirds of curve pairs with labels 1 are linked in a "negative" direction, and curves with label 6 are mostly linked in a "positive" direction.

The reason to why 1s are linked much stronger with each other (the histogram contains a large amount of -2 linking) might be because of the how an image of a digit 1 looks. Practically all images of a digit 1 look like a straight vertical segment, which, when turned



(a) Links in the square (label 1, label 1).

(b) Links in the square (label 6, label 6).

Figure 5.17: Linking number distributions in some squares on Figure 5.16.

$180°$, looks almost the same as the initial image. Therefore, when such image makes a full $360°$ turn, the curve in fact makes two turns. This, perhaps, gives more freedom to create stronger linking between each other, compared to curves of less symmetrical digits.

It is unclear why the sign of linking is preserved in each block. That might be a signal of some structure hidden in the space of observed curves; otherwise, by intuition, the number of positive and negative links would have been roughly equal. An attempt was made to find such structure using the VR complex, which considered curves as vertices and used *Fréchet distance* as metrics between them. Unfortunately, at the moment, nothing worth of interest was revealed by the algorithm.

Note: the use of linking sign makes sense in this problem, because all curves have the logical direction which corresponds to a counterclockwise rotation of an image.

5.3 Projections of 3D-objects

The next selection of datasets that was used for experiments was constructed by rendering a 3D object from different camera positions. A textured model of Earth, which is available in `PyWavefront` library for Python, was used as an object for image generation.

The first dataset was constructed as a set of 1440 $[128 \times 128]$ images, each obtained by placing a camera vertically over the equator of the planet. In other words, each image is characterized by one parameter $\alpha \in [0; 2\pi)$ - a longitude angle. The test dataset contained 8 times more images with just a smaller rotation step.

So densely generated images are located close to each other in $\mathcal{R}^{128\times128}$ space and make up an approximation of a closed curve. To see how neural networks can handle such data, firstly, 10 autoencoders with the structure depicted on Figure 5.18 were trained for 350000 iterations each on the described dataset, with batch size equal to 16 and each training starting with a new random initialization.

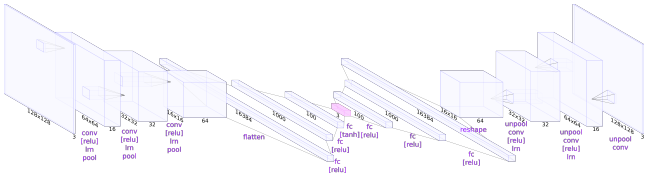


Figure 5.18: Autoencoder structure, simple Earth rotation.

As the result, all curves fit rather well into 3D space, as on Figure 5.19. The curve seems to fill the volume of the $1 \times 1 \times 1$ cube, and it was also noticed that the more complex the structure of the network is, the higher total curvature the curve has, due to a network having more degrees of freedom.

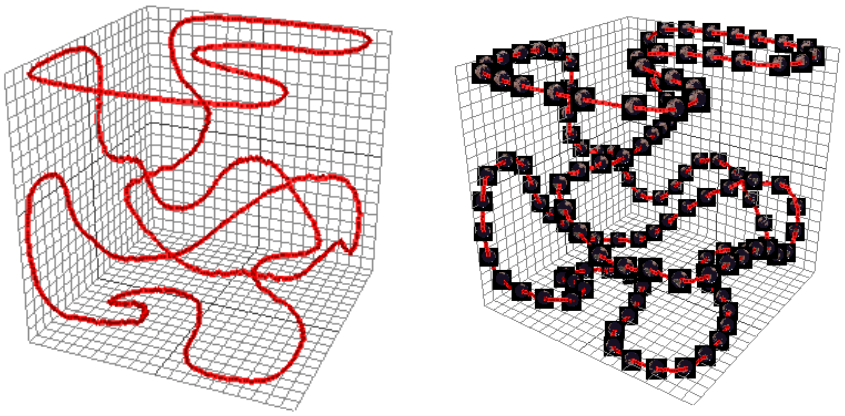
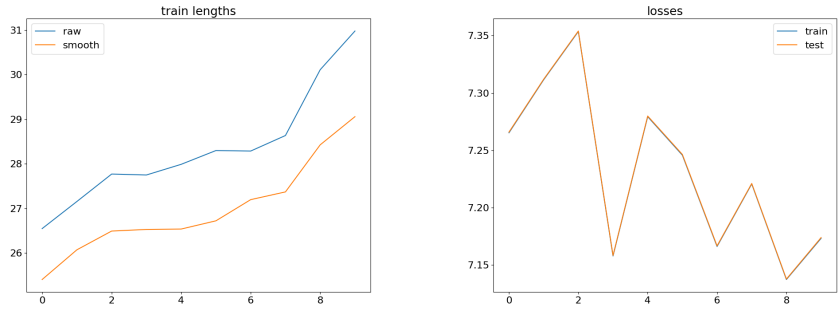


Figure 5.19: An example of a trained embedding of the curve.

It might be interesting to see the connection between how well the curve fills the given space and the error that an autoencoder gives.

The former can be evaluated as the length of the polygonal mesh. Figure 5.20 shows that longer curves tend to give lower error of the autoencoder. This is reasonable, because longer curves most probably utilize provided $3D$ space better, which lets ancoencoder have higher certainty during decoding process.



(a) Sorted lengths of curves among all autoencoders.
Smooth lengths – lengths obtained from exponentially smoothened curves with coefficient $\alpha = 0.5$.

(b) Losses (without regularization) of autoencoders, sorted by lengths of the curves.
Train and test losses are almost the same, due to nature of the dataset.

Figure 5.20: Losses of autoencoders for simple rotation.

Figure 5.21 shows a study over one autoencoder during the training process. It presents how the length of the curve and the loss value change over time. Apart from the first few iterations, the length of the curve is gradually increasing with the decrease in the loss. Interesting to note that the length curve is very smooth compared to the loss curve.

After that, more rotations were added to the dataset. Next step was the addition of second axis of rotation. After fixing a longitude angle, a lattitude angle $\beta \in [-\pi; \pi)$ was selected. All rotation all together put images on a torus. Image size was reduced to 64×64 and the number of training iterations was decreased to 100000, but everything else was kept the same. The structure of the autoencoder was the same as on Figure 5.18, with four times smaller images in convolutional layers.

Figure 5.22 contains two examples of 3D representations of the datasets after training. Firstly, they do not completely look like tori, because images of the planet taken from the south pole are rotationally symmetric and are identified practically to the same point. Secondly,

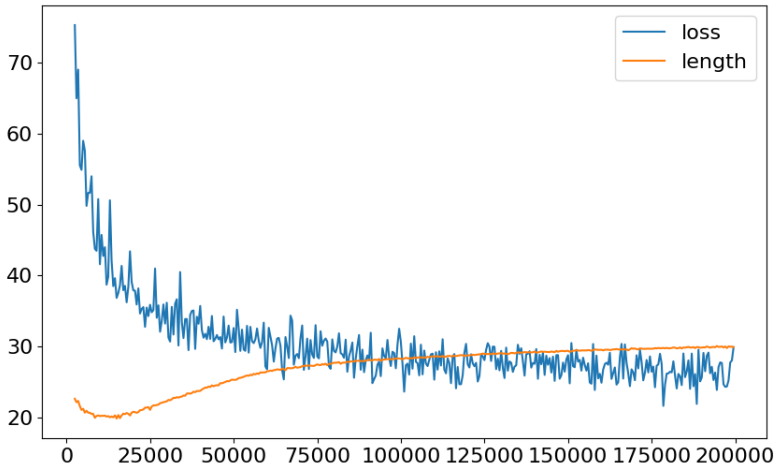
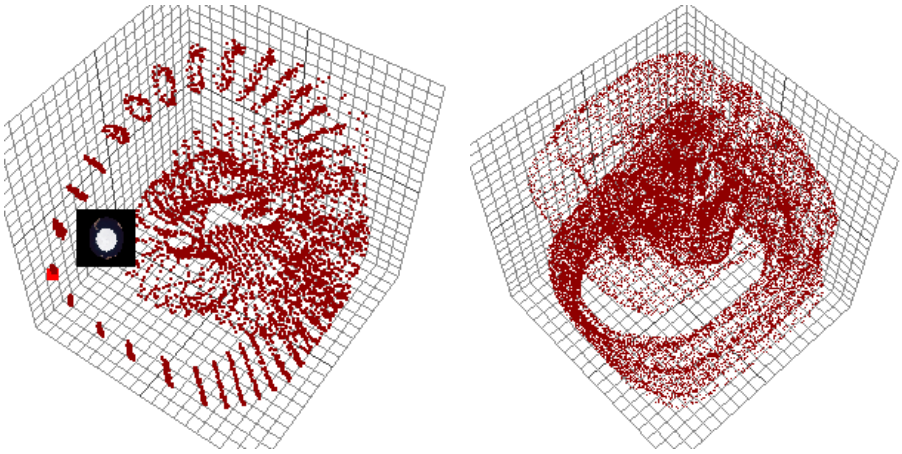


Figure 5.21: L_2 loss and curve length evolution during training, iterations 2500-200000.

there is a number of self intersections, especially when the data points were not uniformly distributed. This observation, together with the previous discussion about the correlation between curve lengths and losses, leads to an idea that it might be beneficial to somehow stimulate the appearance and strength of topological features during training.



(a) Non-uniformly distributed training data.

(b) Uniformly distributed training data.

Figure 5.22: Examples of trained embeddings of toral rotation.

Lastly, full space of rotation was tested. Usually, the group of all such rotations is denoted $SO(3)$. Each data point generated by the following algorithm:

1. Camera's center always points towards the center of the object.
2. Uniformly pick a random unit vector from $\mathcal{R}^3$. Set the camera's forward direction to this vector.
3. Uniformly pick a random angle from $[0; 2\pi)$. Rotate the camera along it's forward direction to this angle.

The obtained encoding, after an autoencoder with the same structure was trained, in on Figure 5.23. The whole space of rotations does not fit into 3-dimensional space; its 3D embedding resembles several nested spheres. However, $SO(3)$ fits into 4D, and it makes more sense to train 4-dimensional autoencoders for that. Figure 5.24 contains some persistence diagrams of the embeddings. Theoretically, $SO(3)$ is diffeomorphic to the real projective space RP^3[35] (which is a space of lines passing through the origin in $\mathcal{R}^4$), which means that it should have one 3-dimensional feature. However, the plots suggest only the presence of 1- and 2-dimensional features, which might happen due to the symmetries in the data, like the one at the south pole.

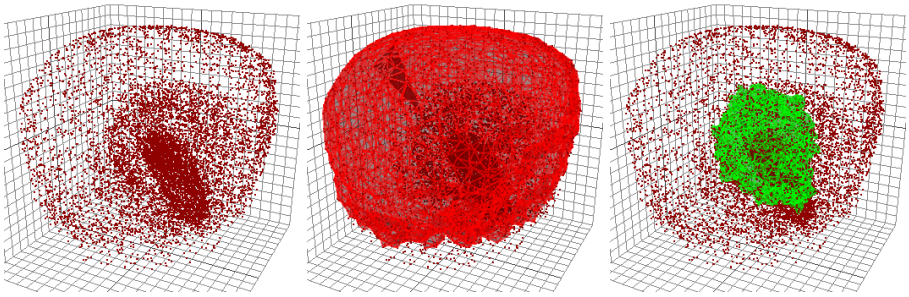


Figure 5.23: An example of a trained embedding of $SO(3)$ rotations into 3D, with two most persistent features.

5.4 Fine-Grained Rotation of CIFAR Dataset.

In the following experiments CIFAR-10 dataset, which contains 32×32 RGB images with 10 different labels, was used to investigate additional

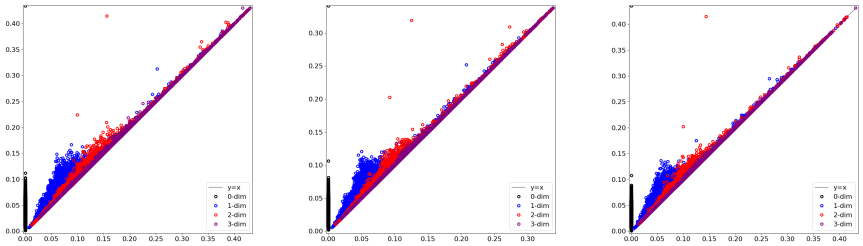


Figure 5.24: Some persistence diagrams of 4D embeddings of $SO(3)$ rotations.

properties of neural networks. The dataset for autoencoders was constructed in a similar fashion as it was done in the Section 5.2, although instead of rotating each image to 16 angles, this time the amount of turns is equal to 64, for the purpose of creating stronger connection between images of one cycle. In order not to end up with unnecessarily huge dataset, only images with label `cat` were used, thus providing us with 12592 images in total for training.

Slightly different technique was applied to the rotation of the images. MNIST images had black background and therefore it was safe to assume that pixels which appear close to corners of the images after rotation are all black. This principle does not work for more complex images that CIFAR contains. If, for example, after rotation we extrapolate unknown pixels as black, autoencoder will only encode the rotating frame of the image, not caring about the center. Therefore, circular black border was added to all images, making them insensitive to rotations. See Figure 5.25 for an example.

In this section, two structures of autoencoders are analyzed. The only difference between the architectures lies in the size of the encoding layer. The first autoencoder, similar to before, tries to learn 3-dimensional representation of the data, whereas the second autoencoder goes into 20 dimensions. In order to analyze the work of the latter, an obtained 20-dimensional point cloud was projected onto 3 dimensions using Principal Component Analysis.

Obviously, building a well-working autoencoder, especially one that goes to an extremely low-dimensional space, is significantly more difficult for a such dataset than before due to high variety in the data. Decoding results can be seen on Figure 5.27, and they show that the 20-dimensional autoencoder handles rotations much better than the 3-

Figure 5.25: Example from the dataset in the Section 5.4.
The left image is a regular image from CIFAR-10. The middle image shows how black frame appears after rotating the given image. The right image presents the black border which was added to all images in the dataset.

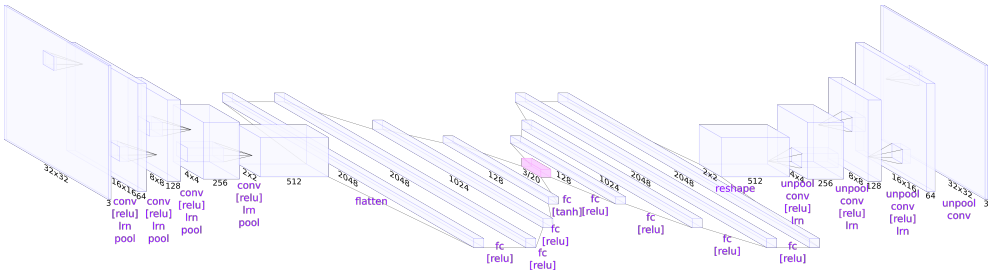


Figure 5.26: Autoencoder architecture for CIFAR with rotations.

dimensional autoencoder, because decoded images are also showing the rotation.

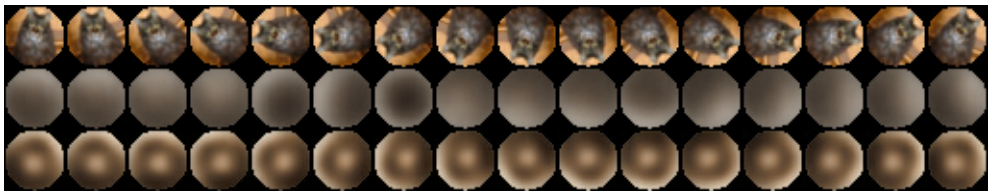
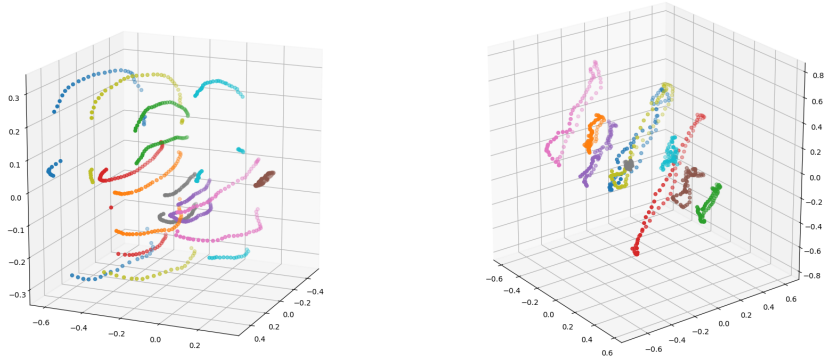


Figure 5.27: Decoding examples.
The 1st row: input images.
The 2nd row: 3-dim decoder output.
The 3rd row: 20-dim decoder output.

In addition, both autoencoders have some structuring processes inside. As we can see on Figure 5.28, autoencoders seem to align cycles in the data in a similar fashion (this, however, is not a ground truth for the 20-dimensional encoder, since we do not see the encoding itself,

but only a PCA projection).



(a) Encoding into 3-dim.

(b) Encoding into 20-dim plus projection into 3-dim via PCA.

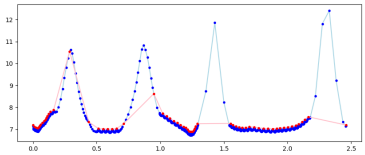
Figure 5.28: Encodings of several cycles of images in modified CIFAR, each cycle consists of 64 rotated versions of one image.

What immediately catches the eye, though, is that 3-dim autoencoder "stretches" the cycles in a strange manner, basically splitting each of them into two (or sometimes more) blocks. It is difficult to explain why exactly that happens, especially because neural networks apply only continuous transformations to the data and cannot "cut" it.

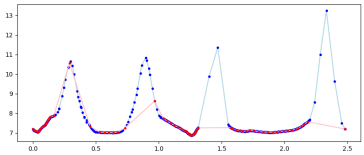
One can look at the losses that the same curves are generating in the autoencoders. In order to get more points on a cycle one could interpolate images in-between existing ones. Two types of interpolation can be applied. The first one is linear interpolation – a linear combination of two nearby images. The second one uses the knowledge about the dataset and creates more points by more rotation, similar to how a test dataset was created in Section 5.3. After looking at 3D-views, it was noticed that both interpolations give similar encoding points. The losses on interpolated images, however, act differently depending on interpolation type, and are presented on Figure 5.29.

On the figure, red dots represent train dataset and blue dots represent interpolated images. X-axis denotes accumulated distance from the first projected image till the current. For example, we can see that 3-dimensional autoencoder created 4 blocks from this curve.

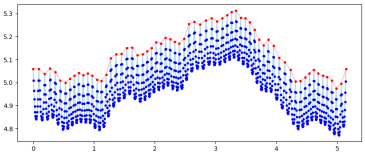
Linear interpolation makes drops in the loss function, while rota-



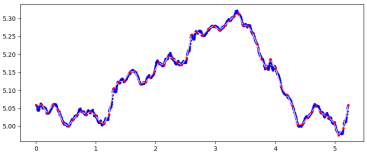
(a) 3-dim: linear interpolation.



(b) 3-dim: rotational interpolation.



(c) 20-dim: linear interpolation.



(d) 20-dim: rotational interpolation.

Figure 5.29: Losses over one selected cycle of images.

tional interpolation of images also smoothly interpolates the loss function. When similar images are linearly combined, the result appears as a bit blurred, which makes it easier to be decoded and thus reduces the error. Losses on more densely rotated images tell us that both networks are not very sensible to rotation, with the exception for several segments in case of 3-dim autoencoder, where 4 large spikes appear in the loss function.

As the final note, no work was done in this subsection related to linking. One one hand, 3-dimensional autoencoder "breaks" the curves, and it does not make much sense to consider points on the sides of the gaps connected. On the other hand, links between simple curves do not exist in 20D, and comparing them after PCA projection probably cannot reveal much information about the network.

Chapter 6

Conclusion

6.1 Summary

In the thesis work, several attempts were made to connect geometric and topological data analysis to machine learning. The area under discussion appeared to be large and rather unexplored, which opens a way to a great number of possible study opportunities. Following that, the main difficulty of the research process appears: how to determine the direction which would lead to positive results.

For this work, one basic idea was chosen and carried through the research. The focus of the analysis was applied on the data itself, and machine learning algorithms were used as data compression tools. More specifically, a class of neural networks, autoencoders, was used. In this way, some features of both used data and considered neural networks were extracted and thoroughly discussed in Chapter 5.

As the main outcome of the work done, autoencoders showed great capability of preserving the most standing-out topological features of the dataset while projecting the data into a low-dimensional space. The thesis work presents several methods which could be used for analysis, and the experiments conducted using those methods provide some interesting results, like the influence of network regularization on data representation and the shapes of embeddings.

Some of the obtained results, unfortunately, did not get a proper rigorous explanation. So for instance, the linkage episode, discussed in the later part of Section 5.2, presents strange entanglement which appears among curves in data, the reasoning for which at the moment is unclear. As an example, some methods of clustering of those curves

can be applied next, or perhaps some kind of complexes could be constructed on top of the curves; hopefully, some of these methods will be able to reveal the cause of the obtained results. Another curious result is presented in Section 5.4, where a simpler autoencoder was starting to stretch the data curves extremely at some points.

One of the biggest problems encountered in the experiments was the presence of outliers in the dataset, which were causing wrong topological inferences in many times. It was overcome by filtering away a certain amount of data points, based on their proximity to dense parts of the data cloud. This, however, introduced several hyperparameters, but other ways that were considered for this purpose were either too computationally expensive or were disallowing chosen methods of analysis.

6.2 Future Work

There are many ways to extend the current research in future work. Firstly, more complex and contemporary neural network architectures like generative adversarial networks could be analyzed instead of simple autoencoders. Secondly, different datasets can be analyzed, and this can mean either some data which is closer to real world, or more abstract data to study the properties of machine learning algorithms. Also, there are many other ways of geometric and topological analysis which were not touched in this work. In addition, some of conducted experiments still do not have the answers to all arising questions.

All of this, however, implies that several problems must be solved. Among those problems is the need to deal with outliers in the data in a better way. However, the major challenge is to come up with methods to work with large and high-dimensional data. The dimensionality problems were already encountered in the analysis of CIFAR dataset, and a lot of work must be done to extend the research on more complex and diverse data and on networks with higher complexity like AlexNet[36].

6.3 Ethical and Societal Aspects

It is always important to consider how newly developed methods of artificial intelligence might affect the society. Already at this point, ma-

chine learning algorithms are applied in many fields and their usage can be found in any sphere of life. Together with growing influence of AI on everyday live, the concern about the usage of those algorithms has risen as well. With the lack of understanding how the algorithms process the data and produce the output, many of their decisions must be taken "on trust"[37]. To introduce some regulations, for instance, "a right to explanation" might be required from AI programs, although it is argued to be not the best solution[38].

However, the methods described in the report do not suffer for the described problem. On the contrary, this work contributes to a developing area of Explainable Artificial Intelligence[39] whose aim is to lift the veil on complex AI algorithms, to make them more fair and transparent. For instance, among the most recent researches in this area, [40] provides a framework which helps to explain and visualize a variety of machine learning models without taking into account specific implementations of tested algorithms. The thesis work makes a step in the direction of explainable algorithms by providing a way to analyze the structures inside the neural networks to see the properties of data which might get learned by the AI.

Bibliography

[1] Arthur L Samuel. "Some studies in machine learning using the game of checkers". In: *IBM Journal of research and development* 3.3 (1959), pp. 210–229.

[2] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[3] Olga Russakovsky et al. "Imagenet large scale visual recognition challenge". In: *International Journal of Computer Vision* 115.3 (2015), pp. 211–252.

[4] Mark Everingham et al. "The pascal visual object classes challenge: A retrospective". In: *International journal of computer vision* 111.1 (2015), pp. 98–136.

[5] Justin Sirignano, Apaar Sadhwani, and Kay Giesecke. "Deep learning for mortgage risk". In: *arXiv preprint arXiv:1607.02470* (2016).

[6] JB Heaton, NG Polson, and Jan Hendrik Witte. "Deep learning in finance". In: *arXiv preprint arXiv:1602.06561* (2016).

[7] Travers Ching et al. "Opportunities and obstacles for deep learning in biology and medicine". In: *Journal of The Royal Society Interface* 15.141 (2018), p. 20170387.

[8] François Chollet et al. *Keras*. 2015.

[9] Dumitru Erhan et al. "Visualizing higher-layer features of a deep network". In: *University of Montreal* 1341.3 (2009), p. 1.

[10] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. "Feature Visualization". In: *Distill* (2017). https://distill.pub/2017/feature-visualization. DOI: 10.23915/distill.00007.

[11] Christian Szegedy et al. “Intriguing properties of neural networks”. In: *arXiv preprint arXiv:1312.6199* (2013).

[12] Jason Yosinski et al. “Understanding neural networks through deep visualization”. In: *arXiv preprint arXiv:1506.06579* (2015).

[13] Matthew D Zeiler and Rob Fergus. “Visualizing and understanding convolutional networks”. In: *European conference on computer vision*. Springer. 2014, pp. 818–833.

[14] GrÃŠgoire Montavon, Mikio L Braun, and Klaus-Robert MÃtller. “Kernel analysis of deep networks”. In: *Journal of Machine Learning Research* 12.Sep (2011), pp. 2563–2581.

[15] Thomas Gebhart and Paul Schrater. “Adversary Detection in Neural Networks via Persistent Homology”. In: *arXiv preprint arXiv:1711.10056* (2017).

[16] Valentin Khrulkov and Ivan Oseledets. “Geometry Score: A Method For Comparing Generative Adversarial Networks”. In: *arXiv preprint arXiv:1802.02664* (2018).

[17] William H Guss and Ruslan Salakhutdinov. “On Characterizing the Capacity of Neural Networks using Algebraic Topology”. In: *arXiv preprint arXiv:1802.04443* (2018).

[18] Karthikeyan Natesan Ramamurthy, Kush R Varshney, and Krishnan Mody. “Topological Data Analysis of Decision Boundaries with Application to Model Selection”. In: *arXiv preprint arXiv:1805.09949* (2018).

[19] Jan Reininghaus et al. “A Stable Multi-Scale Kernel for Topological Machine Learning”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015.

[20] Christoph Hofer et al. “Deep learning with topological signatures”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 1634–1644.

[21] Seth Lloyd, Silvano Garnerone, and Paolo Zanardi. “Quantum algorithms for topological and geometric analysis of data”. In: *Nature communications* 7 (2016), p. 10138.

[22] Gunnar Carlsson. “Topology and data”. In: *Bulletin of the American Mathematical Society* 46.2 (2009), pp. 255–308.

[23] Raimund Seidel. “The upper bound theorem for polytopes: an easy proof of its asymptotic version”. In: *Computational Geometry* 5.2 (1995), pp. 115–116.

[24] Partha Niyogi, Stephen Smale, and Shmuel Weinberger. “Finding the homology of submanifolds with high confidence from random samples”. In: *Discrete & Computational Geometry* 39.1-3 (2008), pp. 419–441.

[25] Konstantin Klenin and Jörg Langowski. “Computation of writhe in modeling of supercoiled DNA”. In: *Biopolymers: Original Research on Biomolecules* 54.5 (2000), pp. 307–317.

[26] T Banchoff. “Self linking numbers of space polygons”. In: *Indiana University Mathematics Journal* 25.12 (1976).

[27] Martín Abadi et al. “TensorFlow: A System for Large-Scale Machine Learning.” In: *OSDI*. Vol. 16. 2016, pp. 265–283.

[28] Andreas Fabri and Sylvain Pion. “CGAL: The computational geometry algorithms library”. In: *Proceedings of the 17th ACM SIGSPATIAL international conference on advances in geographic information systems*. ACM. 2009, pp. 538–539.

[29] Clément Maria et al. “The Gudhi library: Simplicial complexes and persistent homology”. In: *International Congress on Mathematical Software*. Springer. 2014, pp. 167–174.

[30] Ulrich Bauer et al. “Phat–persistent homology algorithms toolbox”. In: *Journal of Symbolic Computation* 78 (2017), pp. 76–90.

[31] Herbert Edelsbrunner and John Harer. *Computational topology: an introduction*. American Mathematical Soc., 2010.

[32] Frédéric Chazal et al. “Robust topological inference: Distance to a measure and kernel distance”. In: *arXiv preprint arXiv:1412.7197* (2014).

[33] “Computing Multidimensional Persistence”. In: *CoRR* abs/0907.2423 (2009). Withdrawn. arXiv: 0907.2423. URL: http://arxiv.org/abs/0907.2423.

[34] Ingwer Borg and Patrick JF Groenen. *Modern multidimensional scaling: Theory and applications*. Springer Science & Business Media, 2005.

[35] Brian Hall. *Lie groups, Lie algebras, and representations: an elementary introduction*. Vol. 222. Springer, 2015.

[36] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[37] Ian Sample. "Computer says no: why making ais fair, accountable and transparent is crucial". In: *The Guardian* (2017).

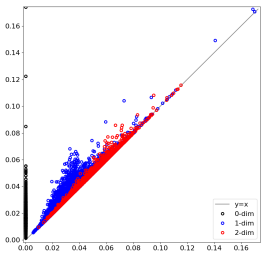
[38] Lilian Edwards and Michael Veale. "Slave to the Algorithm: Why a Right to an Explanation Is Probably Not the Remedy You Are Looking for". In: *Duke L. & Tech. Rev.* 16 (2017), p. 18.

[39] Paul Voosen. "How AI detectives are cracking open the black box of deep learning". In: *Science, July* (2017).

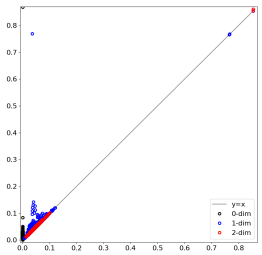
[40] Przemyslaw Biecek. "DALEX: explainers for complex predictive models". In: *arXiv preprint arXiv:1806.08915* (2018).

Appendix A

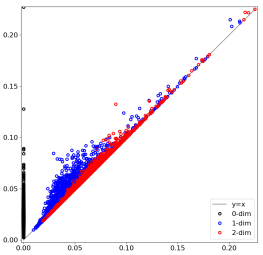
Appended Material



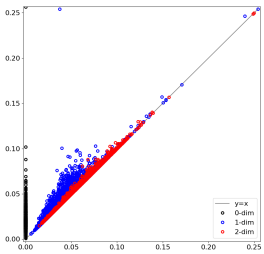
(a) Label 0.



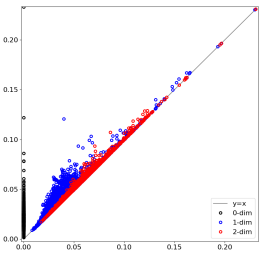
(b) Label 1.



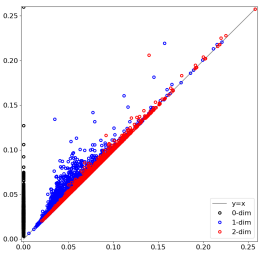
(c) Label 2.



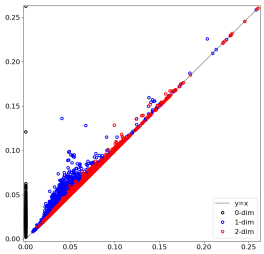
(d) Label 3.



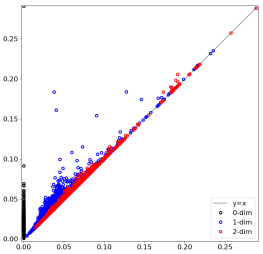
(e) Label 4.



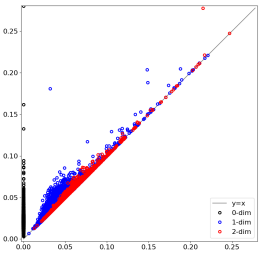
(f) Label 5.



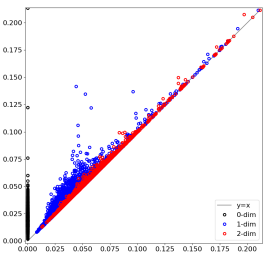
(g) Label 6.



(h) Label 7.

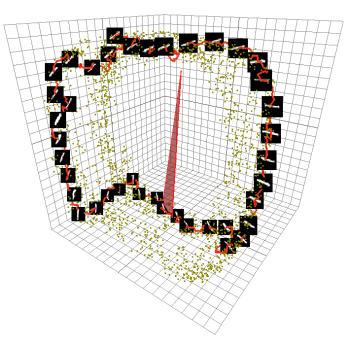


(i) Label 8.

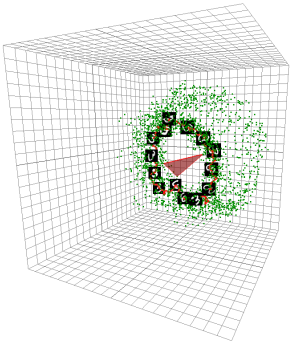


(j) Label 9.

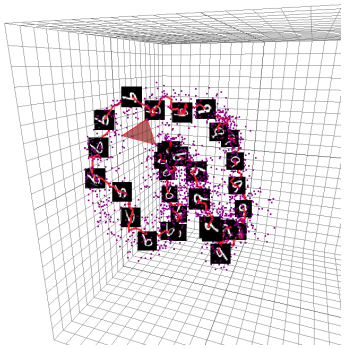
Figure A.1: Persistence diagrams for all digits separately.



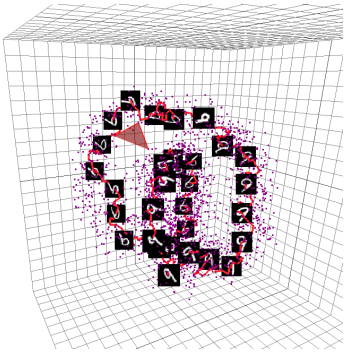
(a) Label 1.



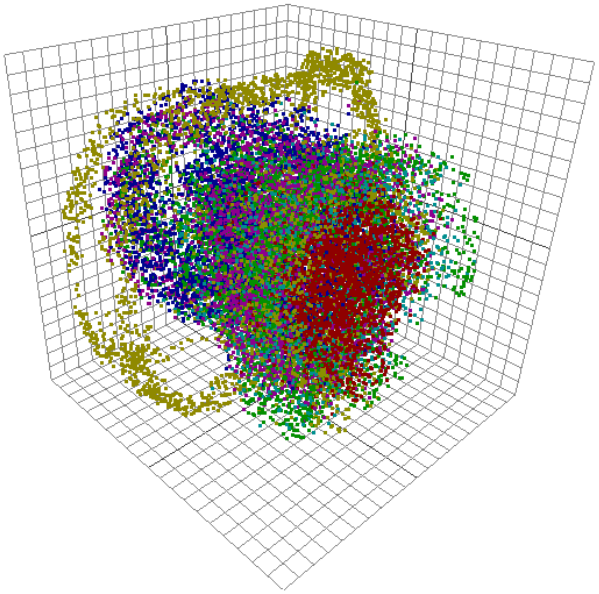
(b) Label 3.



(c) Label 9.

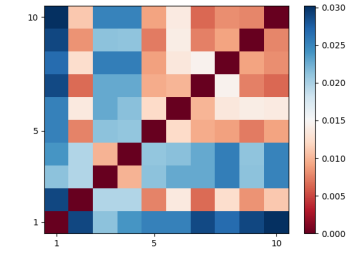


(d) Label 9, different angle.

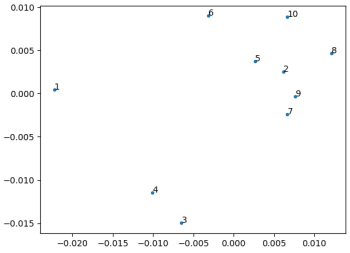


(e) All labels together.

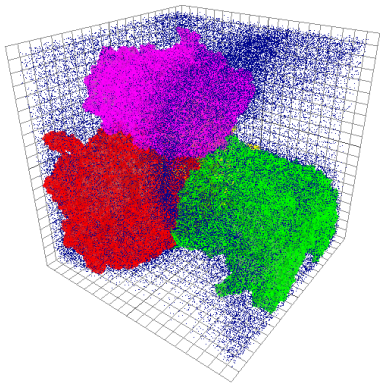
Figure A.2: Some 3D views of encoded MNIST with rotations.



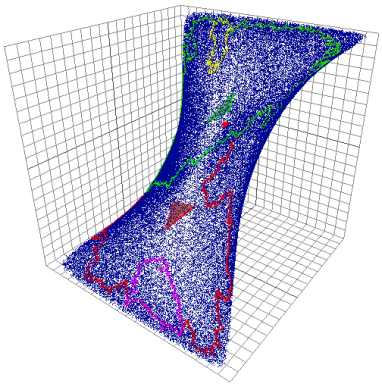
(a) Distance matrix.



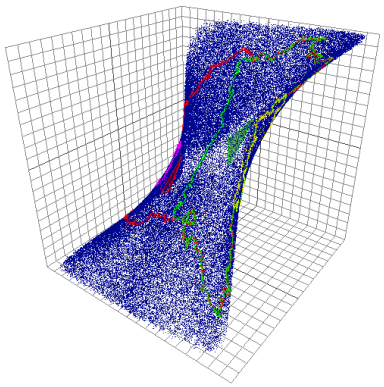
(b) 2D embedding.



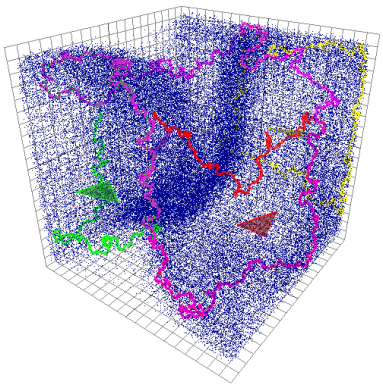
(c) Autoencoder #1.



(d) Autoencoder #3.



(e) Autoencoder #4.



(f) Autoencoder #6.

Figure A.3: Different autoencoders with regularization and distances between them.
Even though the number of trained autoencoders is not very large, we can distinguish three different clusters. For the 1st autoencoder H_2 features became the most significant. Networks number 3 and 4 formed almost planar surfaces, losing not only the rotational symmetry, but also one of the dimensions of the encoding space. The 6th diagram serves as an example of all other autoencoders.