Tor Erik Larsen

Topological Data Analysis on Convolutional Neural Networks

Master's thesis in Applied Physics and Mathematics
Supervisor: Marius Thaule
February 2020

Tor Erik Larsen

Topological Data Analysis on Convolutional Neural Networks

Master's thesis in Applied Physics and Mathematics
Supervisor: Marius Thaule
February 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

Sammendrag

I denne oppgaven presenteres et forsøk på å gjenskape den såkalte primærsirkelen av pikselmatriser i vektorrommet av vekter i et konvolusjonelt nevralt nettverk (CNN) trent på et bildedatasett, ved hjelp av åpen kildekode programvare. Totalt 100 CNN ble trent til omtrent 98 prosent nøyaktighet på et datasett av håndskrevne siffer (MNIST), og en punktsky av 6400 nidimensjonale vektmatriser ble tetthetsfiltrert til 1920 vektmatriser og Mapper-algoritmen ble brukt til å generere nerven av punktskyen. Her er det gjort funn som støtter opp om hypotesen om at et CNN observerer bilder på samme måte som det primære synssenteret hos pattedyr. Sensitivitetsanalyser ble utført for å teste styrken i det foregående eksperimentet. Enkle parameterjusteringer i tettsfiltrering og Mapper-algoritmen medførte at sirkeltopologien ble brutt. En idealisert primærsirkel ble diskretisert og påført vektene til et CNN som et utrenbart nettverkslag. Dette nettverket ble trent på MNIST-datasettet og testet på et hittil usett datasett av naturlige bilder av av reelle husnummerskilt (SVHN). Hypotesen om en dramatisk økning i testnøyaktighet gitt et påført topologisk rom har ikke blitt bekreftet. Til sist ble sammenhengen mellom en sirkulær topologi i treningsdata og testnøyaktighet testet for 100 trente CNN. Det er grunn til å stille spørsmål om tidligere observerte gunstige topologiske effekter på CNN har hatt en tidsbegrenset gyldighet, gitt den raske utviklingen innen maskinlæringsmodeller og infrastruktur generelt.

Summary

In this thesis we present an attempt to reproduce the so-called primary circle of pixel patches in the vector space of weights in a Convolutional Neural Network (CNN). In total 100 CNNs were trained to about 98 per cent test accuracy on a dataset of hand written digits (MNIST). A point cloud of 6,400 nine-dimensional weight vectors were density filtered to 1,920 points, and the Mapper algorithm was applied to generate the nerve of the point cloud. We have findings supporting the hypothesis that a CNN observes images in the same manner as the mammalian primary cortex. Sensitivity analysis were performed on that experiment. Simple parameter adjustments made the circle topology in the data vanish. An idealized discretized primary circle was enforced on a layer of the CNN and frozen from training adjustment. This network was trained on the MNIST set and tested on unseen house number sign photos (SVHN). The hypothesis of a large increase in test accuracy on the unseen data was not confirmed. Ultimately the connection between a circular topology in the CNN weights and train-test accuracy during training was tested. There was no sign of any correlation between them. There seem to be good reason to ask whether previously reported benefits on applying topological information on CNN structures has been of limited duration, given the development of the machine learning field and infrastructure in general.

Preface

This thesis represent the final work in completing the degree of Master of Science (M.Sc) at the Department of Mathematical Sciences (IMF), Norwegian University of Science and Technology (NTNU) in Trondheim.

First I would like to thank my supervisor Marius Thaule, for patiently guiding me through the process of both my thesis and the pre-project. Every time I chased wayward perspectives you pulled me back to the narrow path leading to the finish line. I would also like to thank my co-supervisor Benjamin Adric Dunn for clear and valuable perspectives on the important goals of this work.

Thanks to Jarl for getting me into topology, and for inspiration along the way. Thank you to Erik H for precious advice on the last inning. Thanks to Erik R for turning my naive questions into solid understanding.

Cheers to the open source community, who gives students and researchers the tools to discover topological data analysis for free.

Last but not least my family deserves a large amount of warm thanks for enabling and pushing me to complete my degree. This bold adventure could not have been done without your support.

Table of Contents

Chapter 1

Introduction

With Topological Data Analysis (TDA) we can qualitatively analyze the *shape* of a discrete set in Euclidean space known as a *point cloud*. The building block of TDA is the *simplicial complex*, which in turn are built from *simplices*, that is points, edges between pairs of points, solid triangles and so on. By iterating over a distance between pairs of points we can connect pairwise points together with edges, thus build nested simplicial complexes. The shape of the point cloud is determined by persistent homology, a topological tool counting the number of connected components and n-dimensional holes in the complexes.

The *Mapper* algorithm is a topological clustering method of the original point cloud. The point cloud is projected down to an open covering of a low-dimensional space via a *filter* function. The covering is then pulled back into the original data, linking nodes formed in the coverings by edges, thus forming an approximation to a Reeb graph of the point cloud.

The *mammalian primary cortex* (MPC) is connected to the pair of eyes via the optic nerve in the brain of mammals. The MPC interprets the electro-chemical visual signal as a pattern of lines and edges. Several studies suggest that the space of 3×3 pixel patches found in natural images is helpful to understand the MPC. These patches are suggested to distribute around manifolds in $\mathbb{R}^3$ in a specific manner.

A *feedforward neural network* (FNN) is a directed acyclic graph structure with weighted edges. By use of linear algebra computers can push e.g image data sets in a forward flow trough the network, and classify the images compared to a ground truth. Adjusting the parameters of the FNN is done via a backwards flow called *backpropagation*, using gradient vectors to adjust the parameters of the networks. The flow in both directions is the training process, building a model for inference on unknown data sets.

A *convolutional neural network* (CNN) shares the concept of inference and back-propagations similarly to the FNN. The main idea of the CNN is that *feature maps* identify similar spatial structures in the data set. The CNN has been successfully applied in large scale to e.g. image recognition, translation and semi-self-driving cars. There seem to be building evidence that the CNN perceives image data in a similar manner to the mammalian primary cortex.

Literature summary

The work in this thesis builds upon a trail of at least four important research results. The first result is the paper of van Hateren and colleague [36]. The authors build on the discovery that the primary visual cortex (MPC) cells in maqaque monkeys interprets the visual signal as a pattern of edges and lines, using the *receptive field* in the *simple cells* of the MPC. They compared statistical properties of the receptive fields against similar properties of pixels in natural images. The authors assembled a database of natural images, that was applied in following research.

The study of natural image patches was moved forward by amongst others Lee and colleagues [25]. The authors suggested that the most interesting statistical pattern of natural images is revealed in the space of 3×3 pixel patches. The data source of this work was the database of natural images collected by the previously mentioned authors in [36]. The space of those patches was *density filtered* to provide only high contrast patches. These patches were found to distribute in a specific pattern over the 7-sphere $\subset \mathbb{R}^8$, corresponding to a novel probability distribution.

Building on the concept of natural image patches on a manifold, Carlsson and colleagues [5] used a topological approach to reduce the dimensionality of the patch space to a topological space called the "three circle space" $(S^1)^3$, also known as the parametrization of the Klein bottle. They further partitioned the three unit circles in the primary circle and the two secondary circles. Each circle has a clear diametric pattern of patch distribution, including a pairwise intersection over two patches.

The most recent work and also the starting point of this thesis are the two papers from Carlsson and colleague [13, 6]. First the authors develop an novel perspective of the CNN [16, Chapter 9] using algebra and category theory. The motivation was seemingly to abstract away the linear algebra in order to more freely generalize the successful concept of CNN to other structures, among them the idealized primary circle. By use of proprietary software they did a number of machine learning experiments on several data sets including the MNIST [12]. The motivation of these experiments was three-fold. First to reveal the primary circle in a 9-dimensional weight vector space; second using an discretized primary circle to generalize a trained CNN to other datasets; third using the same discretization of the primary circle to gain a general increase in training times.

Note that the CNN perceives the world like the primary cortex is also recently suggested by Grace in [26]. The FNN is presented in [16, Chapter 6] and [29]. The CNN is presented in [16, Chapter 6]. TDA literature used in this thesis are for example [11, chapter VII] and [14, page 104] .

Motivation and research goals

The original work [13] is performed by use of proprietary software. Neither the source code nor detailed methods using those libraries are given. Here we attempt to reproduce parts of those results with open source software (OSS) exclusively. The main research targets are naturally inherited from the original work. The goals of this thesis are to use TDA and OSS tools to achieve the following.

- Reproduce the primary circle in a vector space of CNN weights
- Run basic sensitivity analysis of the weight data
- Apply discretized primary circle to weight space, test generalization
- Test correlation between 1-homology generators and train-test accuracy of a CNN
- Contribute to the OSS community

Outline

In Chapter 2 we present compact summaries of TDA; the neural network structures FNN and CNN; and a small introduction to data analysis methods used here. To conclude the theory chapter and validate the software libraries, computiational examples of persistent homology and Mapper are found in Section 2.1.5. In Chapter 3 we present the tools and algorithms necessary to reproduce this work. In Chapter 4 we present results and running discussion of four experiments. The first result is the main goal of reproducing the primary circle in the weight vector space; next follows the sensitivity analysis of the first experiment; the last results are those of applying the discretized primary circle for generalization and the first homology vs. accuracy test. In Chapter 5 we conclude this thesis with a general discussion and future work.

Chapter 2

Theory

2.1 Topological data analysis

This section is a brief summary of the author's survey on persistent homology [24] preceding this thesis. There is rich textbook literature on simplicial homology. We have for example followed the books of Hatcher [17, page 107], [11, chapter 3, 4 and 7]. For the Mapper routine we refer to the original work of Carlsson and colleagues in [33].

Simplicial complexes

The building block of simplicial homology is the simplicial complex, containing an abstract simplex or several abstract simplices.

Definition 2.1. Let S be a discrete set. An *abstract simplicial complex* is a finite collection of subsets $K \subseteq S$ such that if an element σ is in K, then all subsets $v \subseteq \sigma$ are elements of K.

An element σ of the abstract simplicial complex K is called a *simplex*, for which the plural is *simplices*. A simplex of cardinality $k+1$ has *dimension* k and is called a *k-simplex*. In other words the cardinality $|\sigma| = k+1$ where $k \geq 0$. A k-simplex where $k \geq 1$ has at least one *face*, that is a subsimplex v contained in σ with k or fewer elements. We can restate the definiton of a simplicial complex as a collection of simplices where all faces of a simplex σ in K is also in K. Furthermore we require that all intersections $\sigma \cap \tau$ of simplex pairs σ and τ are faces of each individual simplex. A 0-simplex is called a *vertex*, the 1-simplex is an *edge* and a 2-simplex is often called a *triangle*. The dimension of a simplicial complex is the dimension of its largest simplex.

The Čech complex

Definition 2.2. Let $\mathscr{U} = \{U_i\}_{i\in\mathbb{N}}$ be an open cover of a topological space X, such that the union $\cup^n\ U_i = X$. For each subcover U_i let v_i be its only vertex. For all $k+1$ intersecting subcovers there is a k-simplex of $k+1$ vertices. This collection of simplices form the simplicial complex $\mathscr{N}(\mathscr{U})$ known as the *nerve* of the cover.

Before stating the nerve lemma, recall that a *good cover* of a topological space contains only contractible intersections of subcovers.

Lemma 2.3. *Let $\mathscr{U} = \{U_i\}_{i\in\mathbb{N}}$ be a good cover of a topological space X. The nerve $\mathscr{N}(\mathscr{U})$ is homotopy equivalent to X.*

For a metric space we have the metric version of the nerve. Recall that a *point cloud* is a finite collection of points in a metric space.

Definition 2.4. Let X be a point cloud in a metric space M. Let $B_x(\epsilon)$ denote an open ball centered at a point x in X with radius ϵ. The associated Čech complex Čech(ϵ) is assembled in the following way. Let Čech$(\epsilon)_0 = X$. For each point $(x, y) \in$ Čech$(\epsilon)_0$ there is an edge $[xy] \in$ Čech$(\epsilon)_1$ if and only if $B_x(\epsilon) \cap B_y(\epsilon) \neq \emptyset$. For all points $(x, y, z) \in$ Čech$(\epsilon)_0$ there is a triangle $[xyz] \in$ Čech$(\epsilon)_2$ if and only if $B_x(\epsilon) \cap B_y(\epsilon) \cap B_z(\epsilon) \neq \emptyset$. In general, there is a k-simplex $[v_o, ..., v_k] \in$ Čech$(\epsilon)_k$ for all points $(x_0, .., x_k) \in$ Čech$(\epsilon)_0$ if and only if $B_{x_0}(\epsilon) \cap \cdots \cap B_{x_k}(\epsilon) \neq \emptyset$.

If X is a point cloud in a metric space M, then the Čech complex Čech(ϵ) approximates the nerve $\mathscr{N}(\mathscr{U})$ of the collection $U = B_x(\epsilon)_{x\in X}$. Therefore Čech$(\epsilon)$ is homotopy equivalent to the subset $\cup_{x\in X} B_x(\epsilon)$ of M by Lemma 2.3.

Vietoris-Rips complex

Computing the Čech complex requires checking all possible intersection of ϵ balls. The following complex is less computationally intensive whilst providing a good approximation to the Čech complex.

Definition 2.5. Let (M, ϵ) be a point cloud with real valued metric $\epsilon > 0$. The *Vietoris-Rips complex* Rips(ϵ) of M is the simplicial complex built of k-simplices $\sigma = \{v_0, ..., v_k\}$ where all pairs of points $(v_i, v_j) \in \sigma$ are within distance $|v_i - v_j| \leq 2\epsilon$.

Referring to the classical example of ϵ-balls on the nodes of an equilateral triangle,we know that Čech$(\epsilon) \subseteq$ Rips(ϵ). It can be shown that for a finite point cloud in Euclidean space, Rips$(\epsilon) \subseteq$ Čech$(\sqrt{2}\ \epsilon)$. This inclusion confirms that Čech$(\epsilon) \subseteq$ Rips$(\epsilon) \subseteq$ Čech$(\sqrt{2}\ \epsilon)$. The Rips complex is a good approximation to the Čech complex in the interval $[\epsilon, \sqrt{2}\ \epsilon)$.

2.1.1 Simplicial homology

Algebraic topology extends the concept of simplicial complexes to simplicial homology. Let K be a simplicial complex of dimension n. Let $\sigma_p = [v_0...v_p]$ be any

p-simplex in K. A *p-chain* $c = \sum_{i=0}^{p} a_i \sigma_i$ is a linear combination with coefficients a_i in the field $\mathbb{Z}_2 = \{0, 1\}$. The collection of all possible p-chains is called the chain group C_p. Given the addition operation, additive identity $c = 0$ and inverse element $-c$, we have an additive abelian group $(C_k, +)$ for all k in $\mathbb{N}$. Since we are working over a field the chain groups $C_p(K)$ are $|K_p|$-dimensional vector spaces generated by p-simplices. Let C_p be the vector space of all p-chains. The boundary operator $\partial_p : C_p \to C_{p-1}$ is a linear map, formally written as

$$\partial_p(v) := \sum_{i=0}^{n} (-1)^i \sum_{j \neq i} v_j.$$

Composing a pair of boundary operators $\partial_{p-1} \circ \partial_p$ always returns zero. We can compose the linear boundary operator to a sequence of vector spaces known as a *chain complex*

$$...C_{p+1} \xrightarrow{\partial_{p+1}} C_p \xrightarrow{\partial_p} C_{p-1}... \xrightarrow{\partial_2} C_2 \xrightarrow{\partial_1} C_1 \xrightarrow{\partial_0} 0$$

where $\ker \partial_p = Z_p \subseteq C_p$ is the subspace of *p-cycles*, and $\operatorname{Im} \partial_{p+1} = B_p \subseteq C_p$ is the *p-boundary* subspace.

Definition 2.6. The p-th homology group (or vector space if over a field)

$$H_p(K) = Z_p(K) \ / \ B_p(K)$$

is generated by the non-bounding cycles in in Z_p. All cycles in the boundary subgroup are sent to zero by the quotient.

The rank of the simplicial homology vector space H_p is called the *p-th Betti number* β_p, effectively the number of p-dimensional holes in the point cloud.

Persistence modules

A *filtration* of a simplicial complex K is a sequence of subcomplexes $K_0 \subset K_1 \subset ... \subset K$ growing in dimension and cardinality. By functoriality of H_p, the inclusion maps $i : K_i \hookrightarrow K_j$ in the *filtered complex* induce linear maps $f_i^j : H_p(K_i) \to H_p(K_j)$ between homology vector spaces. We can assemble these finite-dimensional vector spaces by taking their direct sum thus getting the *persistence module* $M = H_p(K) \oplus H_p(K_1) \oplus \cdots$ with the homology groups as submodules. The graded module M is acted upon by the polynomial ring $F[t]$. The action of $F[t]$ on M_i shifts the homology class $\alpha \in H_p(K_i)$ "upwards" in the module by $t^k \alpha = f_i^{k+j}(\alpha)$. This captures the intuition of persistence, homology classes "born" in the filtration step i and "dying" in a later step j. Alternatively, the indeterminate t^k maps all α from $H_p(K_i)$ to $H_p(K_{j=i+k})$.

Persistence intervals and barcodes

There is a structure theorem stating that a graded module over a polynomial ring is isomorhic to the direct sum of a finitely generated free module and a torsion module.

$$M \cong \left(\bigoplus_{i=1}^{n} t^{k_i} \cdot F[t] \right) \oplus \left(\bigoplus_{j=1}^{m} (t^{l_j} \cdot F[t] \Big/ t^{m_j} \cdot F[t]) \right). \tag{2.1}$$

Here the left summand $t^{k_i} \cdot F[t]$ represents a finitely generated free module of the homology generators that exist in filtration step t_i and persist infinitely. We denote this the *persistence interval* $[t_i, \infty)$. The right summand represents a finite torsion module that exist in filtration step l_j and dies in m_j. This is the persistence interval $[l_j, l_j + m_j)$. A multiset of k-th persistence intervals can be visualized in a diagram called a *barcode diagram* $B_k(K_i)$. Examples of the free modules as barcodes are "everlasting" connected components, that is the $H_0(K)$ generator for the final complex in the filtration. The alternative visualization method of persistence modules is the *persistence diagram*, which shows persistent intervals as points (x, y) in a plane with the x-axis representing "birth date" and y-axis representing "death date" of the interval. Examples will be demonstrated in Section 2.1.5.

2.1.2 Stability of persistent homology

When sampling data from point clouds, how can we trust persistent homology to return reproducible results? We need an assertion of stability as presented below.

Definition 2.7. Let D_1 and D_2 represent two persistence diagrams equipped with a L_∞ metric δ. A *δ-matching* $\mathcal{D}$ of two persistence diagrams is a bijection $\mathcal{D} : D_1 \leftrightarrow D_2$ satisfying the following conditions.

- Two points p_1 and p_2 are *matched* if the distance d between them is less than δ.
- Any points closer to the diagonal than δ are not matched.

The δ-matching filters away all the points close to the diagonal in the diagrams. These points are often considered as noise, since the k-cycles they represent only last a relatively short time before bounding. Then by definition all the remaining points assumed important need to have a close neighbour.

Definition 2.8. The *bottleneck distance* is the infimum of distances δ which maintain the δ-matching. Formally we write $d_b(D_1, D_2) = \inf\ \{\delta \leq 0 \mid \exists\ \mathcal{D}\}$.

The following definitions are used in the Rips stability theorem given by [8]. Let P and Q define metric spaces. Furthermore let the Gromov-Hausdorff distance $d_{GH}(P, Q)$ be described as "the difference between two metric spaces" [35], and small when all points in P are "close" to a point in Q.

Theorem 2.9. *Let $B_p(Rips(P))$ and $B_p(Rips(Q))$ define the p-th persistent homology barcodes of the metric spaces P and Q. Then*

$$d_b(B_k(Rips(P), B_k(Rips(Q)) \leq d_{GH}(P, Q).$$

2.1.3 Computing persistent homology

The boundary operator ∂ is the linear map between the homology vector spaces and give rise to the boundary matrix D. The following is a reminder of computing homology of a simplexwise filtration K for the classical "empty triangle" with simplices $K = \{[a], [b], [c], [ab], [ac], [bc]\}$.

$$D = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Reading the filtration order from left to right we observe the three first zero-columns indicataing the birth of the three 0-cycles. The three next columns indicate which 0-cycle is the boundaries of which 1-cycle. It can be shown [2] that reducing the boundary matrix to $R = DV$ lets us read off the Betti numbers for K. Computing p-th homology is essentially finding Rank(∂_p) for all p, also known as the Betti numbers β_p.

The basis for the essential homology classes generates the infinite persistence intervals $[j, \infty)$. We read these off of V shown here the following way.

$$R = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} = DV = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

First we identify a zero column r_j on R. If the column index j is not a pivot index i on R, this column is a free persistence module in dimension d of the simplex σ represented by the column D_j. Next we find a non-zero columnn r_j with its unique pivot index i. We read off the persistence interval $[i, j)$ that is the torsion persistence module. The cardinality of these modules is the Betti number β_k for the respective cycles. In the triangle example, we find two essential homology classes by reading off column 1 and column 6 in V. There is one "everlasting" connected component and similarly one 1-dimensional hole never dying. Note that reducing the boundary matrix to the Smith normal form allows even simpler interpretation of the complex homology.

2.1.4 The Mapper algorithm

The introduction of the Mapper algorithm was the proceeding by Gurjeet Singh et al. [33]. It seems that one of the driving question in the development of Mapper was finding an alternative method to qualitatively reveal structure hidden in the sheer size of a large dataset. One of the reference papers was the natural image patch statistics paper by Lee and co-authors [25].

The first step of the Mapper method is a function f mapping from a point cloud X to a parameter space Z. The function is called a *filter* or *data lens.* The motivation for the filtering is to project a possibly large higher-dimensional dataset down to a lower dimensional space. The next step is to choose an open overlapping cover of Z, where each cover shares possibly a large number of samples or points. The last step is pulling back the covering to a cluster of the original point cloud. The nodes sharing points are linked with an edge, thus approximating the nerve or Reeb graph [7] of the covering. The intuition of the Mapper algorithm is perhaps that topological information inherent in a dataset might help discover subtle connections that ordinary clustering or dimensionality reduction methods may fail to find.

Figure 2.1 illustrates a classical toy example for the Mapper concept. Computational examples are presenteded on Figure 2.2.

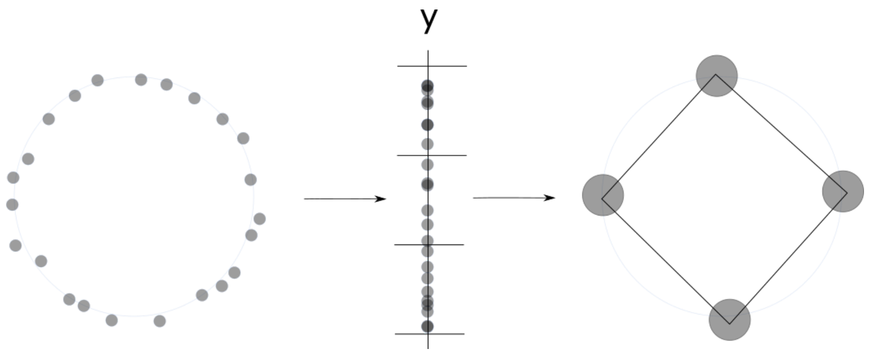

Figure 2.1: An illustration of the Mapper concept. On the left figure there is a toy sampling of the circle. In the middle figure all points of the circle are projected down to their y coordinate. The y-axis is equipped with a 3 overlapping covers. Each subcover share points by overlapping (not shown in the figure). The coverings are pulled back to the circle, and each resulting cluster node that share points is linked with an edge. The result is the Reeb graph, nerve or topological circle informing about the circular topology of the original point cloud.

2.1.5 Computational examples

To conclude the theory section on TDA, we present samplings of three different topological spaces. The samplings in the Figure 2.2 are all performed by sampling 1000 points with random noise introduced. The Mapper configuration is single linkage clustering with *relative gap size = 0.6*, $max\_fraction = 0.3$ and the covering is ten 2-cubes with 0.6 overlap. The circle persistence diagram in dimensions 0 and 1 is shown in the middle figure. A corresponding Mapper complex in the right figure.

The persistence diagram shows one first homology generator indicating the circle. The persistence diagram of the annulus indicate the circle topology whilst the Mapper complex indicates more samples surrounding the non-bounding 1-cycle. The persistence diagram for the torus shows two generators of the first homology in the persistence diagram. Note the absence of the free persistence module, ususally indicating infinite life of e.g. the first born connected component, in all persistence diagrams.

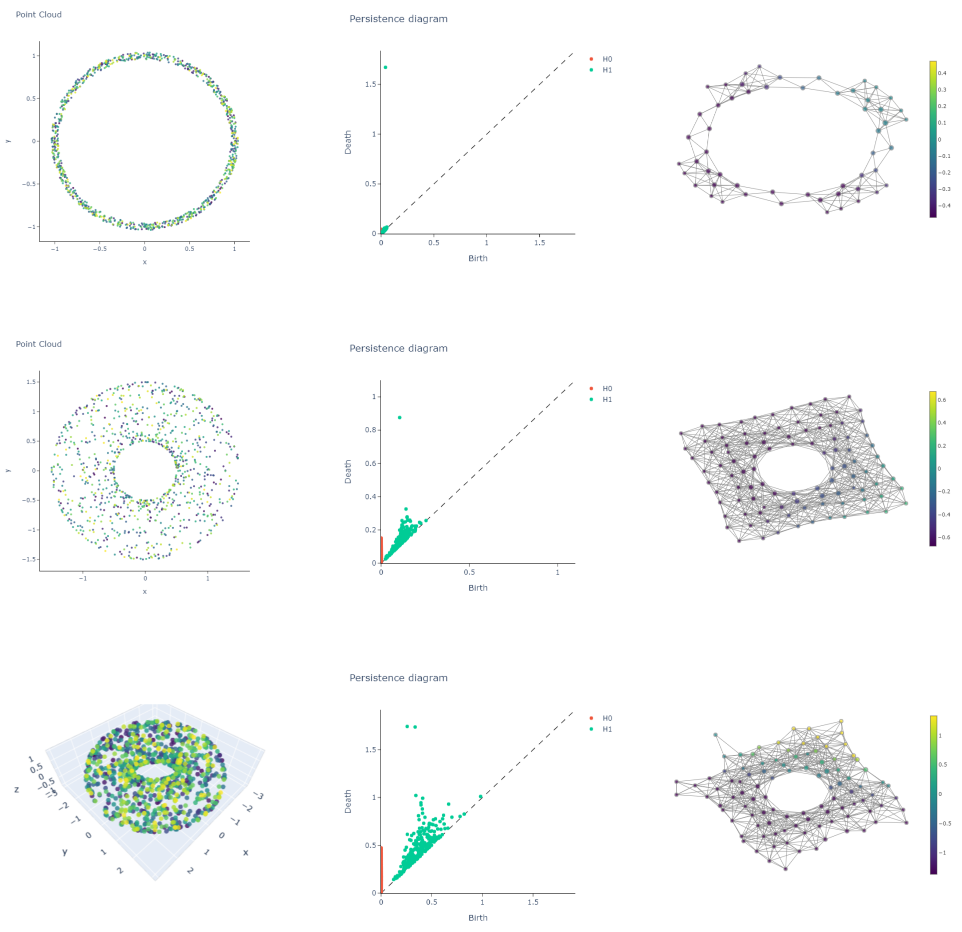

Figure 2.2: Three different topological perspectives of three different topological spaces. The general row layout for the figure is geometric plot, persistence diagram and Mapper complex read from left to right. Top row shows the radius 1 circle sample. Middle row shows an annulus with inner diameter 0.5 and outer diameter 1. Bottom row shows the sampled torus. Note that the barcodes are “hidden” in the persistence diagrams, and can be read as the horizontal lines from any point to the diagonal.

2.2 Neural networks

This brief presentation of neural networks is mainly based on the online book by Nielsen [29] and the online book from Goodfellow et. al [16, chapter 6 and 9].

2.2.1 The feedforward neural network

Starting from the input layer nodes we have a pre-determined activation value vector a^0 for an input vector of dimension n. Collecting all weights connected to the first input layer $l = 1$ in the weight matrix w^1 we can compute all activations in the first layer. By basic linear algebra we have

$$z^1 = \sum^L w^1 a^0 + b^1$$

which generalizes to

$$z^l = \sum^L w^l a^{l-1} + b^l$$

further in the network. Applying the regulating function we have the general activation value vector

$$\begin{aligned} a^l &= \sigma(\sum^L w^l a^{l-1} + b^l) \\ &= \sigma(z). \end{aligned}$$

The feedforward flow terminates with computing the final activation value vector a^L in the output layer L. In order to train the network we need to compute the *loss function*

$$C = c \; ||y_j - a^L||$$

where c is some constant and y_j is the *label* or *ground truth* of the particular training example indexed j. Minimizing the loss function corresponds to maximizing the *accuracy* of the network, a process called *training*. With a cost vector we can compute the loss gradient ∇C, enabling the backpropagation procedure for adjusting the weights.

Backpropagation

The *backpropagation* algorithm defines the way a FNN learns from the loss function gradient $\nabla_a C$. First we have assume a completed forward pass of a batch of k training examples. We define the *output error*

$$\delta^L = \nabla_a C \circledcirc \sigma'(z)$$

where the right side is the elementwise multiplication of the loss gradient vector $\nabla_a C$ and the output activation error gradient vector $\sigma'(z)$. With the output error we can backpropagate and find the error

$$\delta^l = ((w^{l+1})^T \; \delta^{l+1}) \;\circledcirc\; \sigma'(z^l)$$

for each layer $L-1, L-2, .., l_1, l_0$. The principle is taking the known error vector δ^{l+1} and map this "backwards" one layer by the transposed weight matrix $(w^{l+1})^T$. Scaling the backwards step with the activation gradient we find the total error for that layer.

After computing the error for all layers we update all weight matrices w^l and bias vectors b^l with the rule

$$w^l \rightarrow w^l - \hat{c} \sum \delta^l (a^{l-1})^T$$
$$b^l \rightarrow b^l - \hat{c} \sum \delta^l.$$

This is known as the *gradient descent* algorithm, where we minimize the loss by incrementally moving in the direction of the negative gradient i.e "down the hill-side" of the loss function landscape. The constant $\hat{c}$ is proportional to the *learning rate*, that is the stride taken for each gradient descent step.

The cross-entropy cost function

The gradients of the cost function with regards to the weights and biases is the foundation of the learning process of a neural network. When using e.g. the basic quadratic cost function the first gradients are

$$\frac{\partial C}{\partial b^L} = (a^L - y)\sigma'(z^L)$$
$$\frac{\partial C}{\partial w^L} = (a^L - y)\sigma'(z^L)x,$$

where the x represents a training example batch and y is the corresponding label vector. The dependency of the differentiated sigmoid can cause of the network learning rate decreasing. Large values of z cause small changes in the sigmoid of z. This is known as the *vanishing gradient* problem . The *cross entropy cost function*

$$C = -\frac{1}{n} \sum_x [y \ln a + (1-y) \ln(1-a)]$$

over n training examples x compared to the respective labels y, improves this situation by not depending on the sigmoid gradient. The paper of de Boer et. al. [10] and [19] seem to give an early representation of the subject.

Softmax activation function

While the cross entropy cost function is without dependency of the differentiated sigmoid neuron, the *softmax activation function* does not depend on the sigmoid at all. The formal definition is

$$S_j = a_j^L = \frac{e^{z_j^L}}{\sum_k e^{z_k^L}},$$

where $z_j$is the weighted value of the output neuron j, and $\sum_k^n z_k$ is the sum of all neurons in the output layer. What makes the softmax activation function special is that the sum over all output neurons $\sum_j a_j^L = 1$. This makes the softmax in practice a discrete probability distribution.

Dropout

The dropout procedure can be described as dropping half of the hidden neurons for each minibatch. Therefore the training process is done on a number of N/n "different" networks, where N is the number of training examples, and n is the number of inputs in each minibatch. The first references are the papers from NIPS and Hinton et. al. [23, 18].

Alternatives to the sigmoid output

The *RELU* neuron has the formula $max(0, z)$, effectively the absolute value of z. The origin of using the RELU can be traced to e.g. the papers [15, 28].

2.2.2 Convolutional neural networks

The first noticable difference between the feedforward neural network and the CNN is the shape of the input data. Instead of a $(n, 1)$-dimensional input matrix, we have an (m, n)-dimensional input matrix representing each training example. For example the input data of the MNIST hand written digit dataset is of dimension $(28, 28)$. A principal illustration is shown in Figure 2.3.

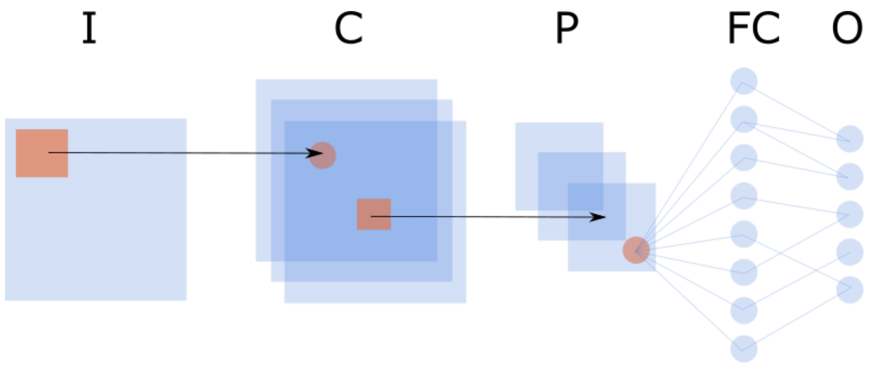

Figure 2.3: A principal illustration of a convolutional neural network. The layer I denotes the input data matrix. The letter C denotes a convolutional layer. The letter P denotes a pooling layer; FC is a fully connected layer and O is an output layer.

Local receptive fields

The *local receptive field* is a $a \times a$ field convolving over the elements in the input matrix. E.g a 3×3 field of grayscale values between 0 and 255 in an MNIST image. This field is convoluted over the whole input matrix by the *stride* parameter, or step length if one wish. For each step or stride the sum $\sum g \circledcirc w$ of that particular patch p and a weight matrix w is sent to one *hidden neuron* in the first *hidden layer.* The first hidden layer is now of dimension $(m-a+1, n-a+1)$. An example is the (28,28) MNIST matrix sent to a (24,24) hidden input matrix by a 3×3.

Shared weights

The CNN uses *shared weights* from every input matrix to all hidden neurons. One weight matrix for each hidden sublayer. The reason for calling these particular weights *feature maps* is that each map supposedly represents a specific spatial *perception* of the input data. Perhaps an intuitive perspective is that each weight vector "reads" different geometric features in the point cloud. Shared weights also represents a dramatic decrease in the total number of parameters used in the total network. If we use a 3×3 sized field of shared weights with three feature maps, we have a total of $3 \times 3 \times 3 = 27$ weights plus one bias for the first two layers of our network. Compare that to $784 \times 30 = 23,520$ weights in the first two layers of an FNN.

Pooling

If the hidden neurons represents one kind of dimensionality or perhaps parameter cardinality reduction, the concept of *x-pooling* is another. In a similar fashion to the receptive fields, the pooling algorithm traverse a hidden layer with a $b \times b$ lens. The algorithm used in this thesis is the *max-pooling*which chooses the largest activation value out of 4 for each patch. This algorithm effectively reduces the next layer by a factor of 4. If our first hidden layer is $24 \times 24 = 576$, the pooling layer is $576 \times 1/4 = 144 = 12 \times 12$.

The fully connected layer

From the last hidden or pooling layer, all receptive fields in all sublayers must wire to at least one fully connected layer. This layer is similar to an "ordinary" hidden layer of stacked nodes in a FNN. The difference is that this layer is connected to all receptive fields in all feature or pooling maps of the previous layer. It might help to think of this summation as "flattening" the last hidden stack of feature maps.

The output layer

There are many parameter choices when it comes to the output layer of a CNN. Here we will use *dropout* [18, 34], which is leaving a certain fraction of nodes out for each forward pass. This will mimic variations by introducing a so called *thinned* networks. We will also use cross entropy loss function and Softmax activation, in addition to a variation of gradient descent called ADAM, founded in [22].

2.3 Data analysis

Density filtration

In the original work of Carlsson et. al. [13] the nine-dimensional point cloud of spatial information from the CNN is first reduced by a density filtration. This is in accordance with the inspirational work by Mumford et al [25] where the authors only cared about 20 per cent of the "densest points" in the set of natural images. Here we use the notation $\rho(k, p)$ to indicate the density estimator k-th nearest neighbor, and p as the decimal number proportional to the percentage of the most dense points one wist to filtrate. Example of a density filtration used in this thesis is $\rho(200, 0.3)$ which denotes using the 200-th nearest neighbor and filtering out 70 per cent of the less dense points. The pseudocode in Algorithm 1 hopefully serves as an explanation.

Algorithm 1: Finding the p-th densest points of a point cloud

```
input  : A point cloud $X$ with $m$ data points
output: A point cloud $\hat{X}$ with $n = p \times m$ data points
For all $m$ points, compute the condensed distance matrix $D$;
Compute redundant distance matrix $\hat{D}$ from $D$ ;
For all $m$ points in $\hat{D}$, find index $j$ of its $k$-th nearest neighbor;
Append index $j$ to a list ;
Identify $k$-th nearest distance $d_j$ in $\hat{D}$, add to list $d$ ;
Sort distances $d_j \in d$ in ascending order;
Let $\hat{d}$ be the $p \times m$ first indices of distance list $d$;
Select $X[\ \hat{d}\ ]$
```

Clustering

The clustering algorithm used in the original paper is a version of agglomerative clustering with a *single-linkaged* criteria [21]. The agglomerative clustering is tree-based, which means a dendrogram can illustrate the clusters in a schematic manner. All samples start as individual nodes from the same *height* of the dendrogram, eventually ending up with the same distance to the root node r. The single linkage term comes from choosing the minimum distance in the distance matrix D when choosing which node pair to cluster, for each clustering step. A pseudocode for assembling the single linkage clustering is shown in Algorithm 2.

Principal Component Analysis

As discussed in the theory section, a Mapper lens projects high dimensional data down to something humanly comprehensible. In the original work the authors have chosen to consistently use the two first components of Principal Component Analysis (PCA) as a lens. This method is derived for example in the book of Jolliffe

Algorithm 2: Agglomerative clustering with single linkage

```
Result: A dendrogram representing a node tree with all leaves equidistant
        from the root node.
input: A point cloud $X$ with $m \times n$ data points
Initial condition: all points is a node ;
Compute a distance matrix $D_1$ for the point cloud ;
Find point pair $(a, b)$ with least pairwise distance $d_1$ ;
Collapse $(a, b)$ to a node $u$ equidistant between the point pair ;
Reduce distance matrix to $D_2$, where $u$ replace $(a, b)$ ;
Find point(s) $c$ with least pairwise distance $d_2$ to $u$ ;
Collapse $(u, c)$ to a equidistant point $v$ ;
Reduce distance matrix to $D_3$ ;
Continue until all points in $D_i$ have a corresponding node. ;
```

[20]. Mainly the method fits the data onto an ellipsoid, where the axis represent principal components eigenvectors scaled by their eigenvalues. The intention of the ellipsoid representation is to read off the ratio of variance in the data explained by each mutually orthogonal eigenvector.

Chapter 3

Method

In this chapter we present the research methods used in this report. The goal is that a competent student could reproduce this work with minimal own interpretation. The caveat of methods including software is that the software most likely will have evolved or expired between this experiment and the eventual reproduction. Version numbers are provided for the sake of determining the functionality available for these experiments.

Introducing the software libraries

Here we aim to briefly introduce the software libraries used in the report. The computational examples of the theory presented in Figure 2.2 will serve as validation of the topological tools.

The software used for all neural network computations was *Pytorch* version 1.4.0 [30]. The main software library for computing topology in the report was *giotto-tda* [1] version 0.1.4. This is an open source software (OSS) solution under development, itself building on other OSS libraries e.g. the *Ripser* library developed by Ulrich Bauer [3].

Quite some time was spent during this work getting familiar with the *Scikit-TDA* library [31]. The Mapper complex in the first experiment is done with the *Kepler Mapper* library from *Scikit-TDA*. Due to consistency concerns and time constraints, *giotto-tda* was preferred for both persistent homology as well as Mapper computations for the remaining experiments.

Based on the plots in Figure 2.2 on Page 10, the software library seem to return the basic topological results we would expect. The exeption is that *giotto-tda* returns a memory error when attempting to compute the second persistent homology dimension. This way we can not for example "confirm" the void of the torus in Figure 2.2h.

3.1 Analysing prior work

In this section we describe the steps necessary to reproduce the original work [13]. We describe training a convolutional neural network, apply density filtration and compute persistence diagrams and Mapper complexes.

Training the CNN

As described in [13, Table 1] we train a CNN with the settings given in Table 3.1. The network returns about 99% accuracy on the MNIST data table [12]. The data set has 60.000 training images and 10.000 test images. Numerous tutorial exist on the internet for training CNN on the MNIST data set, each one corresponding to the preferred deep learning library.

Table 3.1: The structure of the CNN used in the first experiment. This network architecure is denoted $M(64, 32, 64)$ troughout the report. Note that the pooling layers are not defined separately. The filter notation $a \times b \times c$ simply indicates the height and width of the receptive fields, and the height of the convolution layer. Note the use of softmax activation and cross entropy loss function for the output layer. Also note the use of both dropout and a specific stochastic gradient descent variant on the output.

Convolution layer 1	Convolution layer 2	FC layer	Output
$3 \times 3 \times 64$ filters	$3 \times 3 \times 32$ filters	64 nodes	10 nodes
ReLU	ReLU	ReLU	Softmax, Cross entropy
2×2 max-pooling	2×2 max-pooling		Dropout 0.5 ADAM

The network parameters was set to learning rate $lr = 0.001$, and other *Pytorch* parameters $betas = (0.9, 0.999)$ and $eps = 1e - 08$. Each network was trained over 86 *epochs*, that is complete passes of all 60,000 training example images. The 60,000 training images was partitioned into 469 batches of 128 images in each batch. A running training accuracy has been computed for each epoch and batch, then added to a log file. For each completed network training, the *Pytorch* model was saved to a file. Also after training each network, 10,000 test images partitioned in batches of 128 was sent to the model, and a test accuracy between 0 and 1 was logged to file. In total 100 networks was trained, a procedure that took approximately 17 hours with a *NVIDIA RTX2080* graphics card with 8 GB video memory.

The useful output of the networks was the weight matrices used for generating the hidden neurons in the 64 feature maps of the first convolutional layer. With a receptive field of 3×3, this is a vector space in $\mathbb{R}^9$. There was 64 shared weights for each network, in total $64 \times 100 = 6400$ 9-dimensional weight vectors. The resulting point cloud was stored as a *Numpy* array and saved to file.

Data analysis of the weight vectors

Each point i.e row in the $(6400, 9)$ matrix was mean-centered and normalized with the *scale* function in the *Scikit-learn* library. Explicitly this is the operation $z = (x - \mu)/\sigma$ where μ is the mean and σ is the standard deviation of the sample.

A density filter $\rho(200, 0.3)$ based on the k-nearest neighbour density estimator was applied with $k = 200$ and $p = 0.3$ to return 1920 points. This filtration is not a dimensional reduction, rather a cardinality reduction of the point cloud. First we estimate density by finding the k-th nearest neighbor of each data point. By measuring the pairwise distance d between these points we achieve a inverse density estimator ρ. Here $n = 6400$, percentage $p = 0.3$ leading to the reduced point cloud of $0.3 \times 6400 = 1920$ nine-dimensional points. Note that the *condensed* distance matric is sparse, and not readily inspectable. The *redundant* distance matrix is the humanly readable distance matrix with a set of redundant distance observations. The computations are performed on the redundant matrix, since this representation allows intuition over rows and columns.

We validate the density filtration algorithm by applying it to a point cloud of 1000 points in a circle sampled with noise. A correct implementation should by intuition preserve the topology of the original point cloud. We test the algorithm on row standardization as given in the original work and compare with column standardization. The density estimator used was the 30-th nearest neighbor, from which the 30 per cent most dense points was kept. The persistence diagram of the same topology to the right, shows a relative long-lived first-homology generator. The first result is shown in Figures 3.1. By inspection both algorithms seem to capture points in dense clusters.

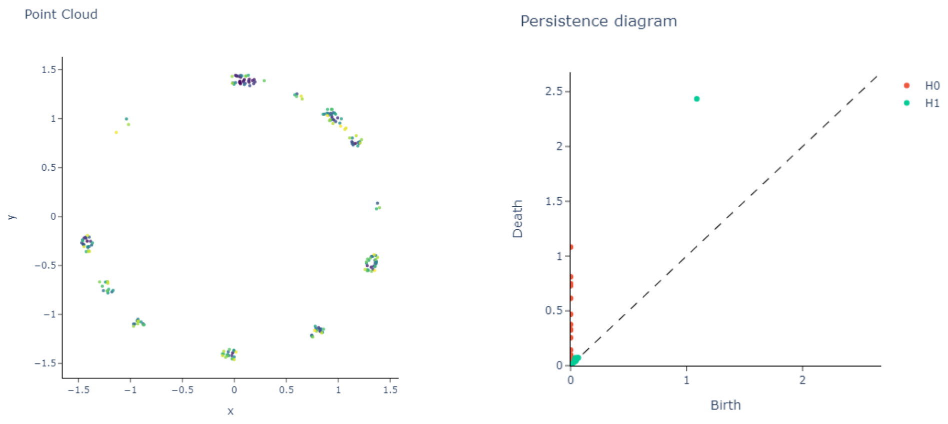

Figure 3.1: The circle with 1000 sampled points, density filtered with column standardization.

The row standardization algorithm seen in Figure 3.2 does not seem to preserve topology, while column standardization does.

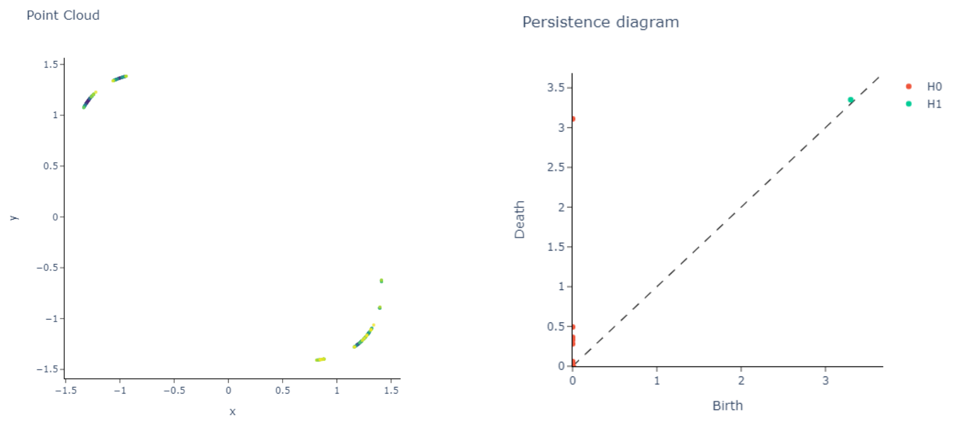

Figure 3.2: The circle with 1000 sampled points, density filtered point cloud with row standardization, the method used in the original work an this report.

Mapper lens and clustering

As in the original work we have chosen the first two components of PCA as the data lens. The cover of the parameter space is 30 2-cubes with 0.66 overlap. The clustering method used is a hierarchical method with a single linkage ward. There are two parameters set for the *Kepler Mapper* library. These are $distance\_threshold = 15$ and $min\_samples = 10$. The distance treshold determines the dendrogram distance treshold for not merging clusters. A larger distance treshold results in fewer clusters, which was necessary to form the circular topology in the Mapper complex. The minimum number of samples was hard coded in the Mapper source code in order to reduce a lot of nodes with just 1 or 2 samples. The combination of these two settings made the circular topology appear in the Mapper complex.

As in the original work [13], we have applied *Variance Normalized Euclidean (VNE)* clustering metric. This is the ordinary Euclidean metric applied after dividing each column by its variance.

A short recipe for finding the mean weight patches to plot adjacent to the Mapper complex follows. After the Mapper complex is generated, use the *HTML* visualization to find nodes distributed on the complex. Then use the element indices of the samples in the node to identify data points in the point cloud. Next we compute the mean of those 9-vectors. Plot that mean as a 3×3 matrix and assemble them on their identified place on the image of the Mapper complex. Note that several attempts were tried to automate a kind of weight simulation. These were abandoned due to the complexity of the source code knowledge necessary to modify dependencies. Therefore the process was to pick a random node on the complex, identify sample indices and then send those indexes to the “patch plot” algorithm.

3.2 Sensitivity analysis of prior work

These experiments mainly reuse the algorithms from the main experiment. Any difference in parameters are stated in the results chapter. The Mapper libraries used in these experiments came from the *giotto-tda* library. The clustering method was a version of agglomerative clustering with two parameters *relative_gap_size* and *max_fraction*. The relative gap size is a fraction of “the largest linkage distance in the full dendrogram”. This is an upper bound on how large a linkage distance can be before breaking into a new tree or cluster. The maximal fraction parameter sets an upper bound of the number of samples as $max\_fraction * (n\_samples - 1)$ clusters before starting from the “bottom” with a new leave node.

3.3 Topological spatial effects on the CNN

In this section we test the second half of the motivation for the original work. The first hypothesis is that applying a topological space to the first convolutional layer in a CNN trained on MNIST, will double the test accuracy for unseen data set.

Generalizing by topology

The following procedure is based on our interpretation of the derivation in Section 5.2 of [6, page 17] and the implementation details found in the first paragraph of [13, Section 6]. For testing generalization of the topology to be discovered, we apply a *idealized primary circle* to the weight vectors mapping from the first convolutional layer. We interpret a 3×3 pixel patch as the subset $\mathscr{L} = \{-1, 0, 1\} \times \{-1, 0, 1\}$ Next we recognize the formula

$$q_{m,n,\theta}(p) = \sum_{(i,j)\in\mathscr{L}} p(m+i, n+j) \cdot f_\theta(i,j)$$

where

$$f_\theta(i,j) = i\ cos(\theta) + j\ sin(\theta) \tag{3.1}$$

as the sum of the elementwise product of the individual pixel grayscale values in the patch, and one element of a weight 9-vector. Recall the mapping from an input image to a hidden neuron in one of the feature maps in the first convolutional layer. This mapping is the function q.

The issue is how to implement this as a discretization of the primary circle found in Figure 2 of [6, page 10]. Especially since the paper states [6, page 19, (5.5)] that the discretization is the set μ_{16}, that is the 16th roots of unity. That number makes seemingly no correspondence to the eight patches of the primary circle, nor the 64 feature maps in the first convolutional layer. The solution that makes most sense is to keep the network structure to $M(64, 32, 64)$ and compute one weight patch for each of the 64th roots of unity angles as described in Algoritm 3.

```
Algorithm 3: Computing the weight patches for discretizing the primary
circle as the set of 64th roots of unity
  input: Parameter $n = 64$
  Result: 16 different 3 ×3 patches
  for each $\theta$ in $\mu_{64}$, the 64th roots of unity do
      compute Eq. 3.1 over $\mathscr{L}$ ;
      save result to a $3 \times 3$ matrix $w_\theta$;
      append matrix $w_\theta$ to array $w$ ;
  Save $w$ as a Pytorch tensor;
```

Implement the test procedure as described in Algorithm 4 with *Pytorch* using the network structure $M(64, 32, 64)$. The layer freezing was done with self .conv1.weight.requires_grad = False in *Pytorch*.

```
Algorithm 4: Testing generalization of CNN
  Result: 3 CNN models from different initialization
  Set the weights of the first convolutional layer to $w$ from Algorithm 3;
   Freeze the first convolutional layer from changing during training;
  Train 3 networks with 86 epochs each on the MNIST dataset;
  Save the trained model;
  Test accuracy for the SVHN dataset rescaled to $28 \times 28$;
  Log results;
  Set the weights of the first convolutional layer to random Gaussian;
  Freeze the first convolutional layer from changing during training;
  Train 3 networks with 86 epochs each on the MNIST dataset;
  Save the trained model;
  Test accuracy for the SVHN dataset rescaled to $28 \times 28$;
  Log results;
  Train 3 networks with nothing fixed or frozen layers;
  Log results;
```

Testing homology versus CNN training

Use the M(64, 32, 64) network structure. Recall that $\rho(100, 0.1)$ denotes a density filtration based on the 100-th nearest neighbor density estimator, filtering 10 per cent of the densest points. The pseudocode of this experiment is given in Algorithm 5.

```
Algorithm 5: Testing 1-homology versus testing accuracy
  input: The MNIST training and testing data sets
  Result: Longest lived 1-homology vs testing accuracy during training
  Train 100 networks on 25 epochs each ;
  For each epoch log training accuracy ;
  For each epoch send model to testing set and log accuracy ;
  For each epoch save the 64 weights 9-vector to matrix ;
  After training, collect weights into $25 \times (6400 \times 9)$ matrix ;
  Run density filtration $\rho(100, 0.1)$ on point cloud ;
  Run persistent homology computation on each of the 25 $(640, 9)$ point
    clouds For each 1-dimensional persistence interval, find length;
  Record maximal 1-dimensional persistence interval for each epoch ;
  Plot log scale of longest $H_1$ and testing accuracy during training ;
  Repeat for SVHN dataset
```

Chapter 4

Results

We present the computation result using the methods described in Chapter 3. In Section 4.1 we analyse the prior work of revealing the primary circle. In Section 4.2 we attempt a brief parameter sensitivity analysis on the original work. In Section 4.3 we test for the generalization effects of the discretized primary circle. At last we test if first homology generators correlate with train-test accuracy of the CNN.

4.1 Analysing prior work

The hypothesis of this experiment is to find the primary circle in the 9-dimensional weight vector space of weights applied from input examples to the feature maps of the first convolutional layer *C1*.

In total one hundred $M(64, 32, 64)$ networks was trained on the MNIST training data set over 86 epochs each. Unfortunately the running training accuracy was not logged. Only test accuracy i.e accuracy on the MNIST testing dataset per network was logged for this experiment. The mean test accuracy for the 100 trained networks was 98.77 per cent, with a standard deviation of 0.11. The running training loss mean for all 100 networks is shown in Figure 4.1.

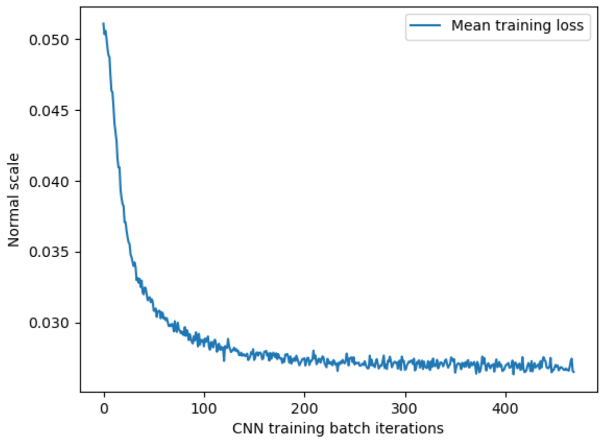

Figure 4.1: The mean training loss logged for each batch iteration for each epoch for each of the 100 networks. Recall that this represents 86 epochs of training on 60,000 images in batches of 128 images.

Two kinds of $\rho(200, 0.3)$ density filtration was applied to the point cloud of 6400 9-dimensional weight vectors as decribed in Section 3.1. The first step in the experiment was to try dropping the sample standardization of the point cloud before density filtration. In Figure 4.2 we can see both the two first components of PCA and the persistence diagram in 0 and dimension 1 as a result. The PCA plot is dense and centered and the persistence diagram show no persistence relatively far from the diagonal. On inspection there are no circular topology in the point cloud of 1920 9-dimensional points *without* sample standardization.

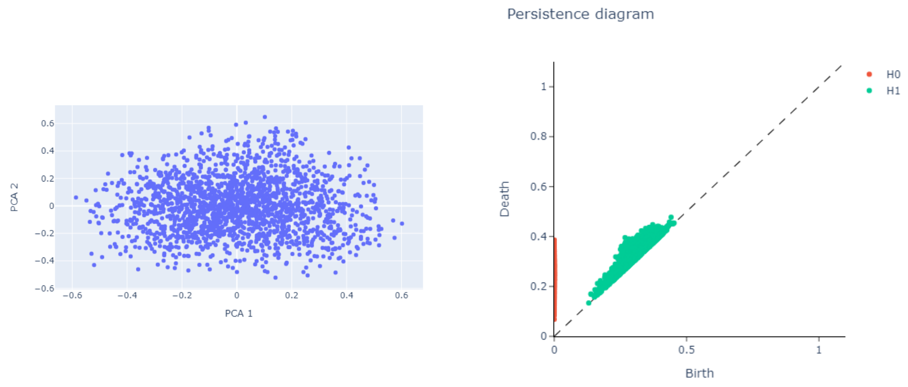

Figure 4.2: Left figure shows the first two PCA components of the density filtered point cloud of 1920 9-dimensional points with no sample standardizing prior to density filtration Right figure shows the persistence diagram for homology dimensions 0 and 1.

Next we apply sample standardization before density filtering and thereafter compute PCA 1 and 2, and then compute persistence diagrams of the density filtered point cloud. The variance ratio of the first two principal components are $(0.28, 0.21)$ respectively. That is, the variance explained by the first two PCA components are 28 per cent and 21 per cent. From Figure 4.3 we see the PCA plot which shows that the points are more dense towards the edges. The persistence diagram of the 2-dimensional point cloud of two principal components have a relatively long-lived generator of H_1. Based on the two plots there seem to be a circular topology in the point cloud of 1920 points after applying sample standardization, density filtration and PCA.

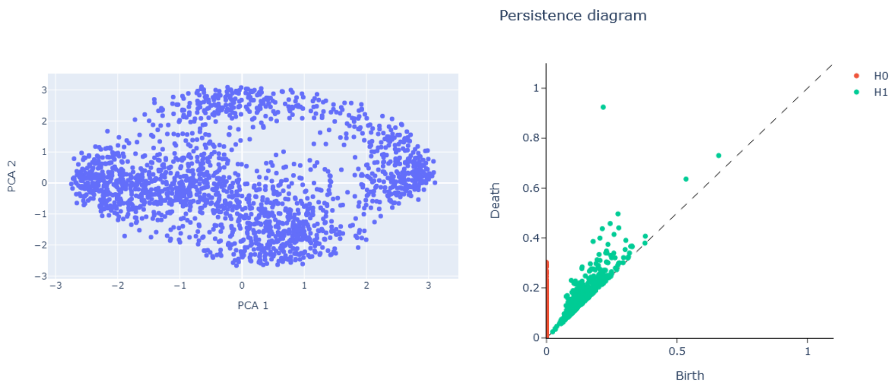

Figure 4.3: Left figure shows PCA 1 and PCA 2 for the density filtered point cloud of 1920 2-dimensional points after sample standardizing. Right figure shows the persistence diagram for homology dimensions 0 and 1.

The previously mentioned PCA decomposition also served as the data lens used to generate the Mapper complex. The complex was computed by applying the NVE clustering metric described in Section 3.1 and the PCA data lens. The Mapper complex of the 9-dimensional point cloud of cardinality 1,920 is seen in Figure 4.4a. The complex clearly has a 1-dimensional hole indicating a circular topology. However we could not adjust parameters to produce the more circular shape found in [13, Figure 2, page 3]. As mentioned in the method section certain attempts were made to reduce "noise" and enforce more samples per node. Still this was the final result.

Also seen in the same figure are imposed mean valued patches of weight vectors. The result is not the obvious primary circle as in the original work [13, page 3]. There is a similarity perhaps in the pattern, but no clear pointer to discovering "opposite" edges and lines on the diagonal. There could be numerous reason for the discrepancy. First there is the choice of "interesting nodes" as described by Carlsson et al. Which node, and only one node at the time? As mentioned there was several abandoned attempts to automate the visualization of weights for the individual nodes. The sheer number of nodes could also play a role, in the generated Mapper complex we have more than 500 nodes. An estimated count of nodes in the original could be about 250 nodes, based on a point cloud of 1920 points. Without the original source code and a more clear method for the discovery this is how close we get to the coveted primary circle.

The persistence diagram of the point cloud with 1920 9-dimensional points do indicate a 1-dimensional hole by presenting a relatively large 1-dimensional barcode in Figure 4.4b.

The hypothesis of this experiment is partially fulfilled. We manage to generate similar weight patch pattern as the primary circle, and succeed in reproducing a circular topology in the density filtered point cloud.

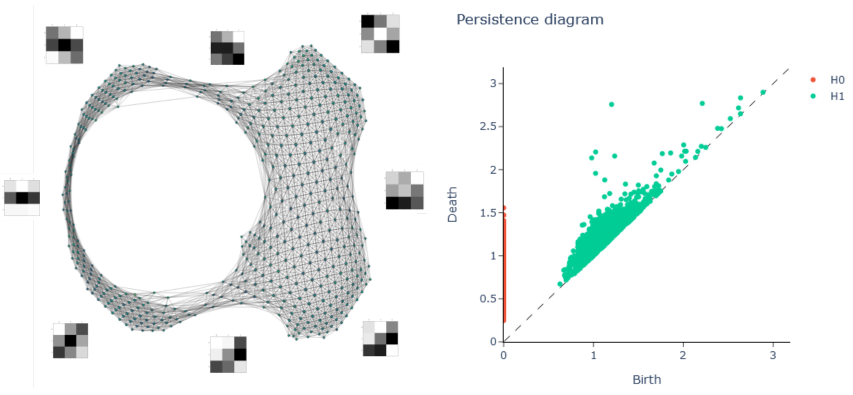

Figure 4.4: The left figure presents the Mapper complex for the density filtered 9-dimensional point cloud and VNE metric clustering. The right figure shows the persistence diagram for the same point cloud for homology dimensions 0 and 1.

4.2 Sensitivity analysis

We do a basic sensitivity analysis of the weight point cloud extracted from the weight space. There are no concrete hypothesis attached, other than twisting some parameter knobs and report the eventual effects. Due to practical restrictions of access to the computer lab, and the fact that one run of 100 networks with 86 epochs took 17 hours, that training attempt was considered sufficient as the source of these experiments.

In Figure 4.5 we present the result of the topological information after choosing 10 per cent of the densest points. This is a $\rho(200, 0.1)$ density filtration. With a point cloud of only 640 9-dimensional points it is perhaps no surprise that the circular topology breaks. The PCA plot shows a cluster like formation of the most dense points. The persistence diagram show no relatively large barcodes, and the Mapper complex has several connected components. This result is perhaps not so interesting in its own right, but it does show that there is a sensitivity of the number of points “required” for the circular topology to appear.

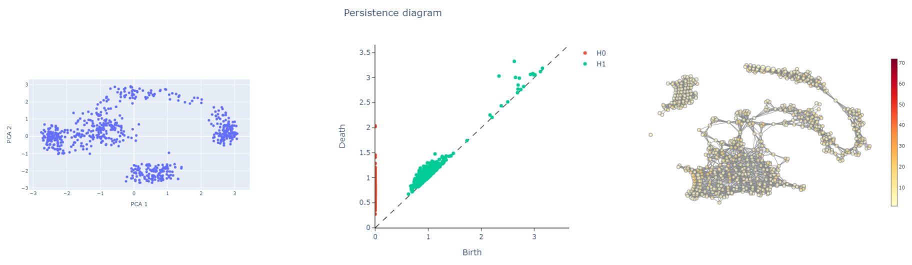

Figure 4.5: Results from choosing 10 per cent of the 200-th nearest neighbors density estimator filtration. The figure layout is PCA plot to left, persistence diagram of data lens in the middle, right column is the Mapper complex of point cloud.

In Figure 4.6 we see the result of changing the density estimator to the 100-th nearest neighbor and keep the 30 per cent most dense points. The PCA looks nearly identical to Figure 4.4a, while the persistence diagram appear to show several 1-dimensional interesting barcodes. The Mapper complex is quite dense, with several holes corresponding seemingly to the persistence diagram. There are more nodes in the complex, that means more nodes have common samples, thus they are closer or denser in the covering. Looking at the data with a more greedy eye for density makes the circle less apparent.

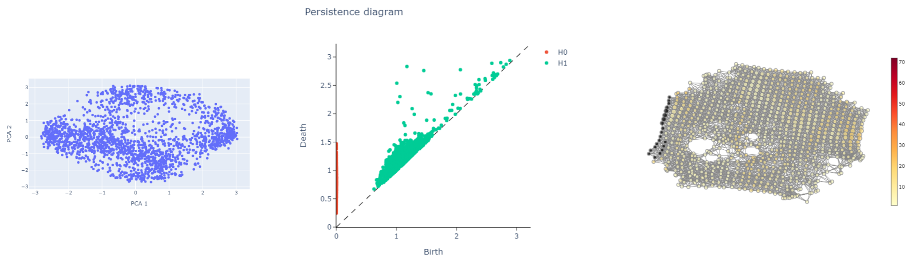

Figure 4.6: Results from choosing 30 per cent of the 100-th nearest neighbors density estimator filtration. The figure layout is PCA plot to left, persistence diagram of data lens in the middle, right column is the Mapper complex of point cloud.

In Figure 4.7 we present the result of choosing only one component of the PCA as the Mapper data lens. The PCA plot does not show any particular topology, while the persistence diagram of the data lens suggest several 1-dimensional holes. The Mapper complex is quite uninteresting, a graph where the nodes contain several hundred samples each. This result is contrary to the classical toy example of Mapper, where the 1-sphere is collapsed onto the line, but by “Mapper definition” restores the circle after pulling back the cover. The covering was 1D, and it should pull back a circle if there was one.

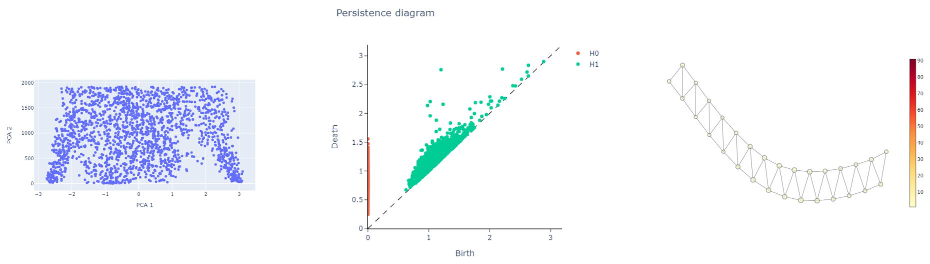

Figure 4.7: Results from choosing the first component of the data lens with the $\rho(200, 0.3)$ density filtration. The figure layout is PCA plot to left, persistence diagram of data lens in the middle, right column is the Mapper complex of point cloud.

4.3 Topological spatial effects on the CNN

We present the result of testing how enforcing the discretized primary circle on *C1* affect generalization of the CNN on unseen data, as described in Section 3.3. Even though two of the networks was trained with the first convolutional layer frozen, the training accuracy started around 90 per cent and levelled rapidly as seen in Figure 4.8. Perhaps this is an indication that training time has decreased considerably since the experiments in the original work? The 3 networks are trained to between 98 and 99 per cent test accuracy on the MNIST data set with 40 epochs, approximately half of the epochs used in the original work.

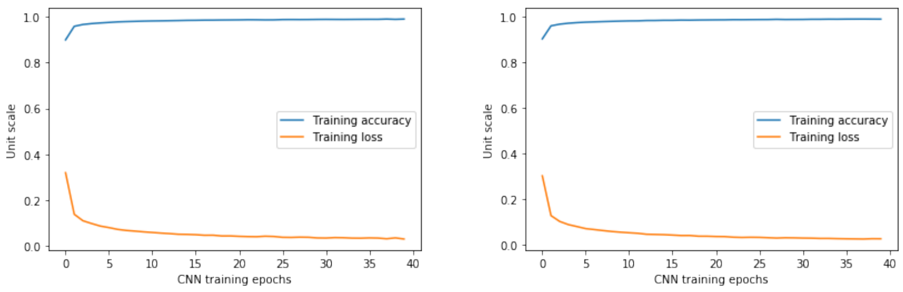

Figure 4.8: Training 3 networks $M(64, 32, 64)$ on the MNIST dataset with two different weight configurations for *C1*. Then tested on the rescaled SVHN data set. The left figure shows *C1* initialized with the discretized primary circle but left free for adjustment during training. Right figure shows *C1* initialized with the discretized primary circle and frozen from adjustment. Running training accuracy is shown in blue, running training loss in orange.

We also train the same network configuration with random Gaussian weights applied to *C1*. The running training accuracy and training losses of the frozen and free layer is shown in Figure 4.9. Based upon these plots alone the networks seem to train in the same manner. Yet an indication that the CNN models of today are robust against perturbation.

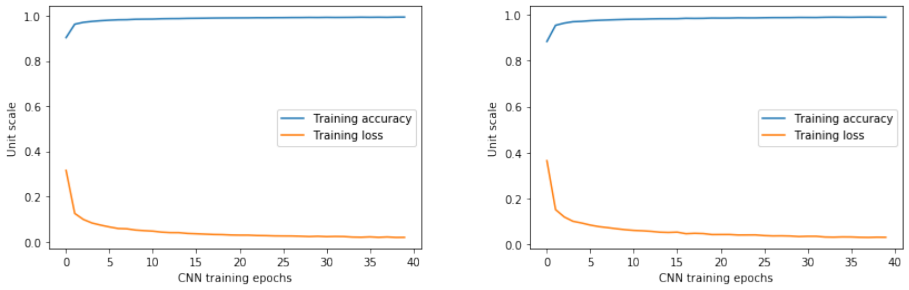

Figure 4.9: Training 3 networks $M(64, 32, 64)$ on the MNIST dataset with two different weight configurations for *C1*. Then tested on the rescaled SVHN data set. The left figure shows *C1* initialized with 64 random Gaussian weights but left free for adjustment during training. Right figure shows *C1* initialized with 64 random Gaussian weights and frozen from adjustment. Running training accuracy is shown in blue, running training loss in orange.

There is quite an amount of interpretation behind performing this experiment. What does it really mean to “fix the first convolutional layer”? Is it initializing weights and then let the gradient descent adjust the weights, or is it freezing the layer so that it is protected from training? And what is the “perfect discretization of the primary circle”? A discretization usually gets more accurate the finer the partition. That is why we decided on using the set of 64 roots of unity to 64 feature maps instead of the 16 roots of unity suggested in the paper [6, page 19, (5-5)]. Could we compute even more n-th roots of unity to achieve an even finer circular discretization? Yes, but we leave that to future work.

In Table 4.1 we see the result of the experiments compared to the results in [13, page 7]. Recall that this process is about training the CNN on one data set, then testing on another data set. For sake of comparison we ran two configurations of the prior weight initializations. We guess that the original work means that *C1* is frozen from training adjustment. The first we note is the near doubling of testing accuracy of the vanilla network on the SVHN data set. It is also interesting to observe that the discretized circle shows the largest effect when not frozen. The Gaussian initialization has also increased by a factor of 5/3. Is this all due to more efficient CNN implementations today than for approximately two years ago? Note that we did not achieve the same testing accuracy from enforcing the primary circle. A final note is that the testing accuracies the SVHN set achieve could variate with $\pm 2 - 3\%$ per cent between networks. Perhaps a more precise experiment could use for example 100 networks as previously applied.

In this experiment we do not confirm the hypothesis. We fail to see that the enforced topological information dramatically increases testing accuracy on unseen data compared to the vanilla and random weight initializations.

Table 4.1: The results of testing how different weight initializations perform when training on MNIST and testing on SVHN dataset. Run 1 has no initialization of weights and *C1* is free for gradient adjustment. Run 2 has *C1* initialized with random gaussian weights and free for the gradient adjusting. Run 3 has *C1* initialized with random gaussian weights and frozen for gradient adjusting. Run 4 has *C1* initialized with weights from the discretized primary circle and free from gradient adjusting. Run 5 has *C1* initialized with weights from the discretized primary circle and frozen from gradient adjusting. *NA* means "Not Applicable". All accuracies are mean values of 3 trained networks.

Author	Run 1	Run 2	Run 3	Run 4	Run 5
Carlsson et. al	11 %	*NA*	12 %	*NA*	28 %
This author	20.4 %	21.5 %	20.0 %	24.6 %	23.7 %

Testing first homology generator vs. running testing accuracy

As the last experiment, we present the result of attempting to discover a correlation between testing accuracy and 1-st homology as described in Section 3.3.

The result is presented in Figure 4.11. There seem to be no correlation between the longest lived first homology and testing accuracy. Note that the accuracy starts already at 95 per cent with *Pytorch*, while the accuracy in the original work starts at approximately 0.3 per cent. Is this experiment still valid with todays' network architecture?

There are several concerns with this experiment. The size of the point cloud is $100 \times 25 \times 64 = 160000$ 9-dimensional points. During the sampling standardization prior to density filtration, there are consistent numerical errors due to standard deviation close to 0. Debugging is complicated with these complex accumulations, while the author has given this algorithm several rounds of quality control.

We do not meet the expectation of seeing a correlation between 1-st homology generator and train-test accuracy in this experiment.

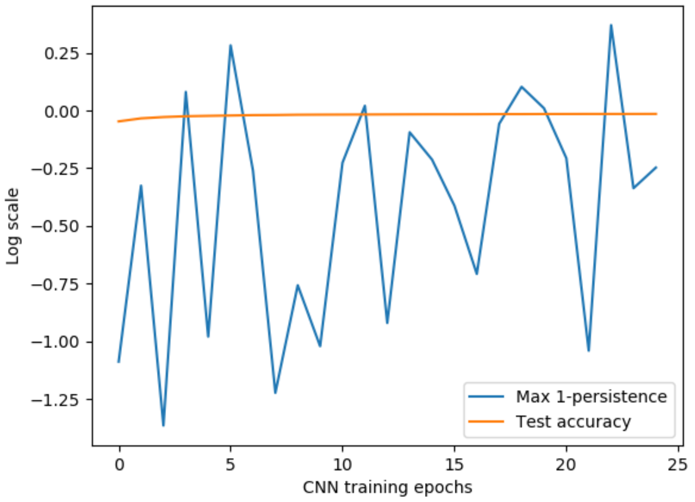

Figure 4.10

Figure 4.11: Log scale plots of the longest persisting first homology generator and testing accuracy for each epoch

Perhaps out of curiosity more than research value. How does the 1-cycles evolve during training of the CNN? By inspecting the algorithm during run time we note that the 1-cycles has large birth values with near zero persistence from the start. We also note that the persistence diagram return 700 0 and 1-cycles for each of the 25 epochs. Is the data too similar or is the persistence computing library not sufficiently tested for edge cases?

Chapter 5

Discussion

The research goals of this thesis has evolved over the time spent studying and implementing the two papers [13, 6] preceding this thesis. The main goal after successfully reproducing the original work was initially to find parameters in the network architecure or e.g. the density filtration to make the eventual primary circle disappear. After some experimental progression was made the full motivation of Carlsson and colleague appeared gradually. Realizing that the authors had used topological features to achieve gains in both training performance and generalization was intriguing and became the main focus towards the end of the project period.

Discussing theory

The theory section on TDA 2.1 is perhaps too brief, and nearly stripped for illustrations. This report is written for a reader comfortable with persistent homology, and perhaps less so for neural networks. The section on neural networks is also short. Hopefully the interested reader is interested enough to seek knowledge on the vast literature on both subjects. The authors' intention was to keep the total amount of reproduction to an infimum.

A short note on the matching criteria in Definition 2.7 for the stability theorem for persistence diagrams. The stability Theorem 2.9 seem to build upon the notion of "points near the diagonal are noise". By solely focusing on the "outliers" in the diagram there seem to be of concern that actually interesting homology generators could "hide" in the presence of larger generators of the same dimension.

Open source software

Open source software (OSS) is often a fruit of academic labor. Public funding of research returns to society in the form of free tools for students and researchers. An emerging trend is that commercial companies are adjusting business models towards OSS for both transparency and profit. The fact that OSS enables for example students to reproduce part of work originally performed by closed source companies is laudable.

The authors of the *Scikit-TDA* library deserves praise for developing during their time as graduate students. Some contribution by the author was made to the *GitHub* repository [32], but is seems that active development unfortunately has stalled as of January 2020. Creating and maintaining OSS is time and focus consuming. If there is no large community around a OSS project they tend to be vulnerable when the founders leave academics or pursue less free positions. Luckily there are newcomers to the OSS scene.

At the time of writing this report, the *giotto-tda* crew seem to make an extraordinary and praiseworthy effort to serve TDA for free to the data science community. Least we do not forget the shoulders upon whom the mentioned developers stand. Contributions from e.g. Ulrich Bauer creating libraries such as *Ripser* and *PHAT* [4] are invaluable to anyone involved in TDA. There are also more up and coming stars in TDA, for example Leland McInnes attracting considerable attention for his *UMAP* clustering algorithm [27]. Also noteworthy are the authors of the *Mapper based classifier*, claiming classification performance on digit data sets near equal to neural networks [9].

The goal of both using and contributing to OSS has been fulfilled. Not so many pull requests has been delivered to the repositories, while delivering several bug reports and running dialogue with the maintainers. In practice the author has switched between a dormant repository and one currently under development i.e. beta status. This has caused quite a few dead ends in time and progress. In retrospect, all of those challenges has added value to the learning experience.

Interpreting prior work

During this investigation, the author has been required to interprete and extrapolate methods and results, sometimes extensively. This is to be expected when reproducing other work, one can not list every detailed step in the method sections. Still there was one experiment that could particularly benefit from more implementation details. The discretization of the primary circle in Section 3.3 and 4.3 included an uncomfortable amount of guesswork, also described in the results.

The hypothesis of revealing the primary circle in the weight vectors of the CNN can quite easily be discussed, based on simple sensivity analysis. That could be missing the point as previously mentioned. One could take an alternative perspective. It seems more reasonable to infer that the original motivation was to show that *if* a neural network is tuned to a circular topology in the feature space, *then* we unlock the potential benefits.

Discussing results

The main discussion of the results in the experiments are subsequently written next to the respective results.

Future work

There are a number of experiments to be further explored. For instance one can attempt to reproduce the Klein bottle topology of the other data sets in the experiment. There is also room for many more variants of applying the discretized primary circle in 4.3. For example, run tests of 100 networks to reduce uncertainty of mean results. There could also be several more attempts to test the discretization itself, how many or how few roots of unity and still achieve some better performance? Also, there is the concept on classifying data based on Mapper complexes. This represent the most interesting route for the author, perhaps one could compete with neural networks for classification? Note that Carlsson and colleague started on the algebraic perspective on the CNN structure [6, page 3]. This could perhaps trigger the interest of the student more inclined towards pure math.

Bibliography

[1] Giotto AI. Giotto-tda, 2019. `https://github.com/giotto-ai/giotto-tda`.

[2] Ulrich Bauer. Computation of persistent homology, 2018. `https://www.ima.umn.edu/2017-2018/SW8.13-15.18/27414`.

[3] Ulrich Bauer. Ripser library, 2019. `https://github.com/Ripser/ripser`.

[4] Ulrich Bauer, Michael Kerber, and Jan Reininghaus. Phat (persistent homology algorithm toolbox). `https://bitbucket.org/phat-code/phat/src/master/`.

[5] G. Carlsson, T. Ishkhanov, and V. et al de Silva. On the local behavior of spaces of natural images. *Int J Comput Vis*, 76:1–12, 2008.

[6] Gunnar E. Carlsson and Rickard Brüel Gabrielsson. Topological approaches to deep learning. *CoRR*, abs/1811.01122, 2018, 1811.01122. `http://arxiv.org/abs/1811.01122`.

[7] Mathieu Carrière, Bertrand Michel, and Steve Oudot. Statistical analysis and parameter selection for mapper, 2017, 1706.00204.

[8] Frederic Chazal, Vin de Silva, and Steve Oudot. Persistence stability for geometric complexes, 2012, 1207.3885.

[9] Jacek Cyranka, Alexander Georges, and David Meyer. Mapper based classifier, 2019, 1910.08103.

[10] Pieter-Tjerk de Boer, Dirk P. Kroese, Shie Mannor, and Reuven Y. Rubinstein. A tutorial on the cross-entropy method. *Annals of Operations Research*, 134(1):19–67, Feb 2005. `https://doi.org/10.1007/s10479-005-5724-z`.

[11] H. Edelsbrunner and J. Harer. *Computational Topology: An Introduction*. Applied Mathematics. American Mathematical Society, 2010. `https://books.google.no/books?id=MDXa6gFRZuIC`.

[12] Yann LeCun et. al. The mnist datatable of handwritten digits. http://yann.lecun.com/exdb/mnist/.

[13] Rickard Brüel Gabrielsson and Gunnar E. Carlsson. A look at the topology of convolutional neural networks. *CoRR*, abs/1810.03234, 2018, 1810.03234. http://arxiv.org/abs/1810.03234.

[14] Robert Ghrist. *Elementary applied topology.* 2014.

[15] Xavier Glorot, Antoine Bordes, and Y. Bengio. Deep sparse rectifier neural networks. *Proceedings of the 14th International Conference on Artificial Intelligence and Statisitics (AISTATS) 2011*, 15:315–323, 01 2011.

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

[17] A. Hatcher and Cambridge University Press. *Algebraic Topology.* Algebraic Topology. Cambridge University Press, 2002. https://books.google.no/books?id=BjKs86kosqgC.

[18] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012, 1207.0580. http://arxiv.org/abs/1207.0580.

[19] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jürgen Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

[20] I.T.Jolliffe. *Principal Component Analysis.* Springer-Verlag New York, 1986.

[21] Stephen C. Johnson. Hierarchical clustering schemes. *Psychometrika*, 32(3):241–254, Sep 1967. https://doi.org/10.1007/BF02289588.

[22] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014, 1412.6980.

[23] Alex Krizhewsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutionalneural networks. 2012. https://bit.ly/2S2gp9r.

[24] Tor Erik Larsen. An introduction to persistent homology, 2019. Pre-project to Master's thesis.

[25] Ann B. Lee, Kim S. Pedersen, and David Mumford. The nonlinear statistics of high-contrast patches in natural images. *International Journal of Computer Vision*, 54(1):83–103, Aug 2003. https://doi.org/10.1023/A:1023705401078.

[26] Grace W. Lindsay. Convolutional neural networks as a model of the visual system: Past, present, and future, 2020, 2001.07092.

[27] L. McInnes, J. Healy, and J. Melville. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. *ArXiv e-prints*, February 2018, 1802.03426.

[28] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pages 807–814, 2010. `https://icml.cc/Conferences/2010/papers/432.pdf`.

[29] Michael Nielsen. Neural networks and deep learning, 2015. `https://neuralnetworksanddeeplearning.com/index.html`.

[30] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019. `https://bit.ly/31u0W1F`.

[31] Nathaniel Saul and Chris Tralie. Scikit-tda: Topological data analysis for python, 2019. `https://doi.org/10.5281/zenodo.2533369`.

[32] Nathaniel Saul and Chris Tralie. Scikit-tda: Topological data analysis for python, 2019. `https://github.com/scikit-tda`.

[33] Gurjeet Singh, Facundo Memoli, and Gunnar Carlsson. Topological methods for the analysis of high dimensional data sets and 3d object recognition. pages 91–100, 01 2007.

[34] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[35] Alexey A. Tuzhilin. Who invented the gromov-hausdorff distance?, 2016, 1612.00728. `https://arxiv.org/abs/1612.00728`.

[36] J. H. van Hateren and A. van der Schaaf. Independent component filters of natural images compared with simple cells in primary visual cortex, 1998. doi:10.1098/rspb.1998.0303.