# Topological Data Analysis of Weight Spaces in Convolutional Neural Networks

Adam Wagenknecht
*University of Missouri-St. Louis*, ajwkc2@umsystem.edu

# Topological Data Analysis of Weight Spaces in Convolutional Neural Networks

Adam Wagenknecht

M.A Mathematics, University of Missouri-St. Louis, 2019
B.S. Mathematics, Truman State University, 2015
B.S. Computer Science, Truman State University, 2015

A Dissertation Submitted to The Graduate School at the University of Missouri-St. Louis in partial fulfillment of the requirements for the degree Doctor of Philosophy in Mathematical and Computational Sciences

May 2023

## Abstract

Convolutional Neural Networks (CNNs) have become one of the most commonly used tools for performing image classification. Unfortunately, as with most machine learning algorithms, CNNs suffer from a lack of interpretability. CNNs are trained by using a training data set and a loss function to tune a set of parameters known as the layer weights. This tuning process is based on the classical method of gradient descent, but it relies on a strong stochastic component, which makes the weight behavior during training difficult to understand. However, since CNNs are governed largely by the weights that make up each of the layers, if one can gain an understanding of the space in which these weights lie, then much can be learned about the structure of the CNN and how it calculates its output. Topological Data Analysis (TDA) is a recent addition to the field of data science, which uses ideas from geometry and algebraic topology to create a novel methodology for analyzing high-dimensional datasets. Specifically, TDA offers a mathematically rigorous method for studying the structure of CNN weight spaces. In this thesis, we use TDA to study the weights of a binary classification CNN model trained on a large dataset known as Dogs vs Cats. Our analysis reveals that, during training, the 3x3 convolutional filter weights of the CNN model in question exhibit non-trivial homological properties. Namely, persistent 1-cycles occur within the first homology groups. This structure is similar to the structure that is found in 3x3 high-variance image patches of natural images [47, 3] demonstrating that a CNN built on this data set learns features of the ambient structure of the image data. This demonstrates the validity of the CNN and, along with work done by Carlsson and Gabrielsson [18], furthers the hypothesis that convolutional layer weights arising from training a CNN on natural image data lie on a space with non-trivial geometry, in particular a non-empty first homology group.

# Contents

## Acknowledgments

First and foremost I would like to thank my advisor Dr. Adrian Clingher for his support, advice, and the copious amount of time he put in to helping me with this work. I would also like to thank the University of Missouri Science and Technology for use of their computing cluster ”The Foundry” which is supported in part by the National Science Foundation under Grant No. OAC-1919789. Lastly, I would like to thank my wife Emily for her patience and loving support during all of my work.

# 1 Introduction

## 1.1 General Background: CNN Models in Deep Learning

Over the last two decades, the study and development of machine learning methods for natural image classification have become an important interdisciplinary field, at the confluence of science and engineering. Machine learning and statistical learning models seek to replicate the remarkable ability of humans and many other animals to make sense of and classify natural images. Hypotheses on how this is achieved have long been advanced by neuroscientists. Of particular importance is the 1962 work of Hubel and Wiesel [29] on the function of the primary visual cortex (V1). The primary cortex belongs to the larger mammalian visual cortex (MVC) and is the lowest level image-processing layer, beyond the retina. Hubel and Wiesel showed that the primary visual cortex of monkeys and cats contains two special types of neurons, referred to as simple and complex, that tend to individually respond to small regions and features in the visual field. In particular, these cells are responsible for detecting edges and lines. Higher MVC levels are known to perform more abstract tasks. Later on, Fukushima [17] developed the first functional mathematical model for the biological visual system. This model, called Neocognition, consists of layers of artificial neurons (non-linear statistical models for classification), referred to as S-cells and C-cells, whose activation seeks to replicate the behavior of the Hubel-Wiesel specific neurons. The S-cell and C-cell layers, as computational systems, are arranged in a self-organized hierarchical architecture, governed by numerical weights that change over repeated exposure to labeled digital images. Over the next decade, the Neocognition model ideas have been further improved and refined, culminating, in the early 1990s, with the development of a class of models called Convolutional Neural Networks (informally referred to as CNNs or ConvNets). This class of models have their origin in computer vision and their methodology belongs to the larger class of feed-forward neural networks.

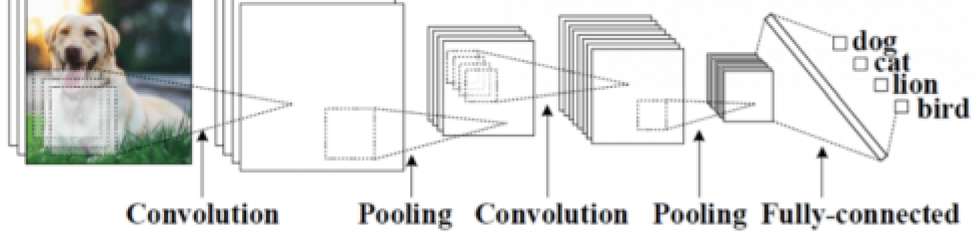


Figure 1.1: *Generic CNN Architecture (Source: KDnuggets.com)*

The CNN architecture consists of a hierarchical succession of layers, organized as 2D or 3D grids (or tensors, in machine learning lingo) that encode the pixel array format of digital pictures. Information circulates in a sequential manner, each layer being activated from the previous one, according to several possible computational patterns and via a choice of numerical weights. The two most important layer types are: convolutional and pooling. Convolutional layers attempt to replicate the behavior of S-cells in Neocognition. Pooling layers are akin to the C-cells. In addition, a fully connected layer is traditionally added at the end of the sequential hierarchy, where the final image classification is performed. The accuracy of a CNN model depends on an optimal choice of weights. Such a choice is obtained via a process called training - a mathematical optimization process inspired by the method of gradient descent, but with an added stochastic component. This procedure uses the training data - given digital images labeled according to the desired classification - and involves a repeated updating of the weights, via a sequence of forward-propagation and back-propagation algorithms.

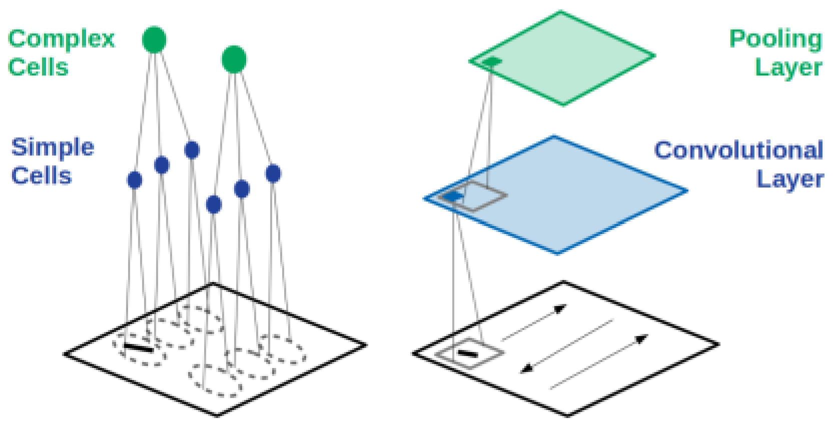


Figure 1.2: *Hubel-Wiesel Pathway vs. CNN model [40]*

In recent years, benefiting from the dramatic increase in available computing power, CNNs and neural network models, in general, have become powerful tools for solving problems associated with large and complex datasets. They have produced outstanding classification results on images, text, time series, etc. The neural network methodology has permeated the fields of machine and statistical learning, leading to the appearance of deep learning - a new field at the interplay of mathematics, statistics and computer science.

While strongly anchored in mathematics and statistics, the field of deep learning is nevertheless driven by practical applications. CNN models are nowadays crucial components in a wide array of technologies: object detection in self-driving cars, banking fraud detection, natural language recognition, medical image analysis, healthcare risk assessment, social media, etc. However, the interpretability of CNN models is far from ideal. Models are in a continuous process of transformation and their efficiency is routinely judged rather experimentally, via a testing and validation process. A model is considered satisfactory if its accuracy consistently meets a pre-agreed validation threshold. A model is good if it largely works as expected, when given independent new data. This methodology has led to a significant and continuously growing gap between practically successful applications and the theoretical understanding of their underlying deep learning models. In short - practice is far ahead of theory.

This prevalent lack in theoretical understanding has consequences. For instance, it limits the usefulness of these models in many key domains, such as regulated industries like the financial sector or healthcare. But from a more practical point of view, the absence of a good theoretical understanding makes it very difficult for one to predict a CNNs ability to generalize on unseen new data. For instance, it has been observed [22] that certain standard CNN models tend to be vulnerable to what is called *adversarial behavior* - they overfit on particular datasets that are slightly modified from their core training data. Intuitively, some image recognition CNN models can be made to fail by making small changes to the image data, changes that would be almost imperceptible to a human. For these reasons, it is necessary to develop new methods for gaining a better understanding of the internal states of a CNN. Due to the very large number of nodes and the highly stochastic nature of the training mechanism, this is a problem in data analysis.

## 1.2 Motivation: the Carlsson-Gabrielsson Hypothesis

A key to understanding CNNs is an understanding of the statistical properties of their weights and the space in which they lie. Thus, studying how the CNN weights evolve during the training process is a topic of great interest in machine learning research. Several studies [63, 58, 43] have been done in this direction, from a machine-learning point of view, with interesting qualitative results. Most of these works involved a direct inspection of the weights, using traditional statistical analysis techniques. The difficulty in drawing rigorous conclusions here mainly stems from the high-dimensionality of the weight data and the fact that the weight effect on the behavior of the network is simultaneous. Attempts at reducing the dimension of the data, via traditional dimension-reduction techniques, like Principal Component Analysis (PCA), lead to significant information loss.

A relatively new methodology for studying this area is Topological Data Analysis (TDA). TDA is a data analysis methodology that combines classical geometric ideas with modern machine learning algorithms, to identify shape features in data. Pioneered about a decade ago by Gunnar Carlsson, a Stanford University mathematician and founder of the data analytics company Ayasdi, TDA's theoretical foundation is based on a branch of pure mathematics called algebraic topology, which studies the notion of shape and builds topological invariants. Algebraic topology takes on two main tasks: the measurement of shape and the representation of shape. Both tasks are meaningful in the context of large, complex, high-dimensional datasets. They provide one with a way to measure shape-related properties within the data, such as the presence of loops. These measurements are then recorded in an algebraic structure called persistent homology. TDA uses the theory of persistent homology to create so called persistence diagrams - compressed representations of datasets that retain features and reflect the relationships among data points in the form of a combinatorial graph, with the purpose of identifying hidden structures, trends, and critical patterns. The TDA methods have proved to be quite useful for a plethora of machine learning technologies.

The application of TDA to study shape features of the data begins by constructing a simplicial complex, such as a Rips-Complex or Čech Complex, on top of the data being considered. These are discussed in greater detail in Section 2, but in general both the Rips and Čech Complexes are built by defining a distance metric $d(x, y)$ and a threshold $\rho$. These then determine the simplices based on distances between the points compared to the threshold. From this there are two general paths that are most commonly used in TDA. The first uses an algorithm known as Mapper which constructs a simplicial complex from a projection of the data (often referred to as a lens) into a low-dimensional subspace which can be visualized. This allows for a visual inspection of the data to determine its structure. Mapper is explained in detail in Section 4.2. This method is quite useful in that it produces a visualization of the data; however, this visualization is heavily dependent on several hyper-parameter choices and only shows the structure for a specific lens. Without a broader understanding of the structure this makes the visualization susceptible to misinterpretation.

The other TDA method used is that of persistent homology. Using the simplicial complex one can build out the Homology Groups $H_k$ and subsequently determine the Betti numbers $\{\beta_i\}$ where $\beta_k = Rank(H_k)$ (i.e., rank of each Homology Group). These Betti numbers are topological invariants that describe the structure of the data. This process, like Mapper, is dependent on the threshold value $\rho$, used to build the simplicial complex. As such, the structure that is found is true of one given scale or coarseness (see def 2.16) defined on the data which may not give much information about what the true structure of the data is. If the threshold is chosen too coarse or not coarse enough the true structure of the data set may be hidden. For example, consider a sine wave with noise on it. If the threshold is equal to the period, then one cluster will exist for every period and thus the structure will look like a straight line. However, if the threshold is too small then every feature created by a variation in the noise will be modeled which will cause the true smooth sine wave to be hidden.

The solution to this is to examine the data under a set of increasing threshold values. This gives rise to a set of nested simplicial complexes

$$\emptyset = K^0 \subseteq K^1 \subseteq K^2 \subseteq ... \subseteq K^m = K$$

referred to as a filtration. This is the same concept that Carlsson refers to as functoriality [4], which is the idea that topological invariants should be studied not solely in relation to the objects being inspected but also in relation to maps between the objects. From [4]: "Functoriality is central in algebraic topology in that the functoriality of homological invariants is what permits one to compute them from local information, and that functoriality is at the heart of most of the interesting applications within mathematics. Moreover, it is understood that most of the information about topological spaces can be obtained through diagrams of discrete sets, via a process of simplicial approximation."

The TDA method for studying topological invariants, more specifically in our case Betti numbers, over a filtration is via Persistent Homology. As we will describe in detail in Section 2, using a filtration $f$, a 2-dimensional structure, referred to as a persistence complex, can be built consisting of a family of chain complexes

$$C_*^0 \xrightarrow{f^0} C_*^1 \xrightarrow{f^1} \cdots \xrightarrow{f^{p-1}} C_*^p$$

which are produced by the filtration (one for each level of the filtration). Each complex is a set of chain groups which can be traversed via the boundary operator

$$\partial_p, \cdots \xrightarrow{\partial_{p+2}} C_{p+1}^* \xrightarrow{\partial_{p+1}} C_p^* \xrightarrow{\partial_p} C_{p-1}^* \xrightarrow{\partial_{p-1}} \cdots$$

This complex gives rise to a map of homology groups at each dimension. The filtration allows homology groups to be defined together now with the number of filtration levels they span, $H_p^{i,j} = Z_p^i/(B_p^{i+j} \cap Z_p^i)$. That is, we can denote $i$ as the filtration level for which $p$ a non-bounding cycle first appears and $j$ as the filtration level in which $p'$ a boundary homologous to $p$ exists and thus derive the persistence as $j-i-1$. Using this persistence value, we can talk about the topological invariants that persist for some time $t > M$, as the true structure of the data, invariant of the choice for clustering threshold value.

Turning back now to CNNs, we seek to gain a better theoretical understanding of how CNNs characterize a given problem space. As outlined in the prior section, it is known in practice that CNNs perform very well in Supervised Learning for Classification, that is, they are able to use a training data set to derive a description of the problem space in order to determine correct classification of input data taken from the same space as the training data. It is, however, not fully understood from a theoretical perspective why or how CNNs are able to accomplish this. Thus, there is a need for deeper and more theoretical studies of how CNNs function. It should be noted that we said "derive a description" rather than "approximate the distribution" because while it is assumed in practice that CNNs approximate the distribution of the problem space there is no theoretical property that ensures that this is true. That being said, the work of many in this field as well as our work in this paper lend further credence to the idea that CNNs do in fact approximate the distribution of the problem space.

In our case, we seek to further the understanding of CNNs applied to the problem of natural image classification, a problem at which, in practice, CNNs are known to perform well. Now, a CNN, after training, is described primarily by the weights of each of its layers (see Section 3 especially Section 3.10) and most of the computational power in a CNN comes from the convolutional layers. Therefore, if a CNN is correctly able to classify a set of images it would be expected that the convolutional weights from that CNN would resemble the structure of the images it is seeking to classify. As such, before seeking to understand the structure of a CNN built to classify natural images it is important to understand if there is a structure to the space of natural images and if so, what it is.

An image is made up of an $m \times n$ grid of pixels each of which has either a single grayscale value (typically 0-255) or a 3-tuple of color values - typically red, green, and blue with each value being 0-255 denoting the hue of each color, in this way any color can be represented. For the sake of the present discussion, we assume only grayscale images. This assumption is valid since in a CNN RGB values are often separated into their color channels in which case the hue of a single color acts the same as a grayscale value. Using this, we can say that an image lies in the space $\mathbb{R}^p$ where $p = m \times n$. The question then becomes, what is the nature of the collection of all images in $\mathbb{R}^p$. For instance, does the collection appear as a manifold modeling a p-dimensional multivariate gaussian distribution or is there some sort of submanifold with lower dimension than $p$ and if so, what is the dimension and topology of this submanifold.

It seems fairly obvious that natural images are non-Gaussian. Intuitively by looking at a picture if one knows what part of the image is they can guess to some extent what the rest of the image contains, thus making it non-Gaussian. For example, consider an image with half a tree in the middle of the image, if the other half of the image is missing the observer can guess with fairly high confidence that the pixels next to the visible half of the tree will depict features of a tree instead of some random set of grayscale or RGB values. Of course, from a mathematical point of view, intuition is not enough and so many studies have been done to show that natural images are not random. Take as an example the work of Daniel Ruderman [54]. This then begs the question, if not Gaussian, what is the structure of natural images?

While some effort had been made in this area, David Mumford was the first to fully tackle this problem. The bulk of his work in this area was published in a 2003 paper with co-authors Ann Lee and Kim Pedersen [47]. In this work, Mumford et. al. came to the conclusion that although the entire manifold of images is not characterizable in any meaningful way, the space of small image patches does appear to have a structure which can be at least partially described. Additionally, Field [15] and van Hateren [25] found that an understanding of local statistics within an image provides a lot of information about the global statistical properties of the image. This means that we can restrict our study to small local image patches and still be able to make some claim about the overall structure of images.

In order to study this space of small image patches, Mumford et. al. utilized a set of black and white images published in conjunction with a paper by van Hateren and van der Schaaf [26]. They then examined 3x3 patches within the image, yielding vectors in $\mathbb{R}^9$. Amongst these images the authors restricted their research to "high contrast" (high variance) image patches (call this set $\mathcal{M}$) as most of the patches lie within solid color portions of the image which do not carry any interesting structure to them. By restricting to high-contrast image patches, they restricted their study to those patches which were interesting and non-trivial. This aligns with human visual processing as Reinagel and Zador [52] found that humans tend to focus on regions with high spatial contrast when presented with a natural image scene. To determine those patches which are high contrast, the authors developed a measure of contrast which they referred to as the D-norm. The developed D-norm is a logarithmic measure of the differences in the values calculated by summing the differences between 4-connected neighbors ($i \sim j$) in the 3x3 patch and then taking the square root. Formally, $\|x\|_D = \sqrt{\sum_{i \sim j}(x_i - x_j)^2}$, or in matrix form:

$$\|x\|_D = \sqrt{x^T D x} \text{ where } D = \begin{bmatrix} 2 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & 0 & 0 & -1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 3 & -1 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & -1 & 4 & -1 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & -1 & 3 & 0 & 0 & -1 \\ 0 & 0 & 0 & -1 & 0 & 0 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & -1 & 3 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 2 \end{bmatrix}$$

Using this, the authors focused on the top 20% of high-variance patches in the data set as evaluated by the D-norm. These values were then mean centered, meaning that the mean over the 9 pixels in the patch was subtracted from each of the pixels. This has the effect of mapping patches with the same relative brightness between the pixels but an overall brighter patch to the same point. This also reduces the data to an 8-dimensional subspace within $\mathbb{R}^9$. The data was also normalized using the D-norm which has the effect of mapping patches that differ only in overall contrast to the same point. Formally for $x \in \mathbb{R}^9$, then:

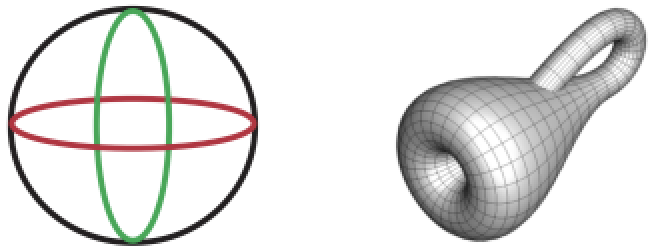
$\hat{x} = \frac{x - \frac{1}{9}\sum_{i=1}^{9} x_i}{\|x - \frac{1}{9}\sum_{i=1}^{9} x_i\|_D}$ is the normalized form of $x$.

The result of these two operations projects the data to the surface of a 7-dimensional ellipsoid in $\mathbb{R}^8$. The author's work was then to study how the data $\mathcal{M}$ was arranged within $S^7$. From this, the authors surprisingly found that the data was concentrated around an annulus. Their conclusion was then that high-density, high-contrast patches from images lie along a high-codimensional manifold within the space of patches and that this submanifold has a non-trivial geometry meaning that $H_1$ (the first homology group) was non-zero. Codimension refers to the dimensional difference between a manifold and its containing space, so formally, if $N$ is a submanifold in $M$, then $codim(N) = dim(M) - dim(N)$. Understanding this, the fact that the space of high-contrast image patches is a high-codimensional manifold is a non-obvious and particularly interesting result.

Building on the work of Mumford et. al., Carlsson, Ishkhanov, de Silva, and Zomorodian, in their 2007 paper [3], sought to apply TDA to the study of high-contrast image patches, $\mathcal{M}$, in order to develop a topological model of $\mathcal{M}$. In particular, they found that the 2-dimensional submanifold of $\mathcal{M}$ has the topology of the Klein bottle. More specifically the authors showed that the topology is that of the three-circle model $C_3$ (see Figure 1.3) which is topologically equivalent (i.e., homotopy equivalent) to the Klein Bottle. In $C_3$, two of the circles (pictured as red and green in Figure 1.3) intersect the third circle in precisely two points while they themselves do not intersect. This is possible because $C_3$ lies inside of $\mathbb{R}^8$ so such a geometry is feasible. The authors show that the topology of each of the components is in fact topologically a circle. The authors also found that the value $k$ used to determine the density of the filtration (larger values of $k$ allow for less dense points to be considered, see Section 4.1.2 K-Nearest Neighbors Density Filtration) has a large effect on the topology of the submanifold. When $k$ is smaller the 3 circle/Klein bottle topology is found; however, when k is increased the topology collapses to a single circle of linear gradients rotating around the circle. This is referred to as the Primary Circle. The authors used a TDA software package called PLEX to verify their results by using it to compute the Betti numbers of each of the data sets.

The authors also give a potential rendering of the layout of the image patches on the $C_3$ model (see Figure 1.4); however, no work within the paper appears to prove that this is in fact how the patches are arranged within the submanifold.

In 2018, Carlsson, in collaboration with computer scientist R. Gabrielsson, used this theoretical background to propose a novel TDA-inspired approach on the CNN weight analysis problem [18] (expanded in [5]) using a program called Ayasdi which implements a TDA algorithm known as Mapper [60]. Mapper produces a visual representation of the structure of a point cloud



(a) Three-circle configuration (b) Klein bottle

Figure 1.3: *3 circle configuration and Klein Bottle representation*

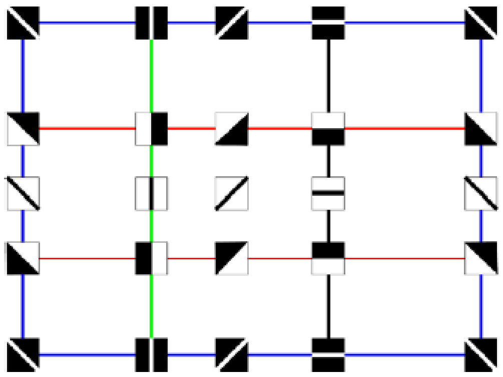


Figure 1.4: *Potential rendering of the layout of the image patches on the $C_3$ model [3]*

set (see Section 4.2.1). Carlsson and Gabrielsson noted statistical similarities between the weights in a given layer of a CNN model and small 2D digital image patches extracted from a large dataset of natural images (e.g. the same data studied by Mumford et. al.). Given that a CNN will attempt to learn the structure of a data set, it would make sense that the topology of 3x3 weights in a CNN would be similar to that of high density, high-contrast 3x3 patches of natural images.

Extrapolating the above observations to CNN models, Carlsson and Gabrielsson then advanced the hypothesis that CNN weights form high-dimensional point-clouds that evolve over the training process, according to geometric manifold patterns similar to the ones found by Lee, Mumford and Pedersen. They also proposed that one should be able to understand the evolution of these geometrical shapes using TDA methods. In their work, Carlsson and Gabrielsson show that for several different CNNs the weights of the convolutional layers approximate a similar topological structure as found in high density 3x3 image patches found by Carlsson et. al. in [3]. The authors show that for grey-scaled images the network weights take on non-trivial geometry, namely that there is a high persistence cycle in $H_1$. This circle consists of rotating gradients (i.e., Primary Circle). For color images, the weights approximate a geometry with a similar 1st homology group to that of $C_3$ i.e., the three-circle model where each color is represented by one of the circles. The authors also found the three-circle model in a few of the greyscale convolutional layers, but, while not discussed in their work, it is assumed that the single circle shows up as opposed to $C_3$ in grayscale because the density value $k$ is set high enough, just as observed in [3].

Building on all of this we can now attempt to state the hypothesis of the work performed by Carlsson and Gabrielsson. This hypothesis is not explicitly stated anywhere, although the summation of it is posited in [3] and verified in [18] and [5]. As such, we will explicitly define the Carlsson-Gabrielsson Hypothesis [18] as follows:

**Carlsson-Gabrielsson (C-G) Hypothesis**: The point-cloud of 3x3 weights arising from training a CNN on natural image data lie on a manifold with non-trivial geometry. In particular, one in which $H_1$ is non-zero. This is the same structure found in high-density, high-variance, normalized 3x3 patches taken from natural images [3].

The Carlsson-Gabrielsson hypothesis carries significant implications in several directions. First, if validated, the hypothesis would shed light on how CNN models learn during the training process, and give a theoretical model for their understanding. It would also provide evidence for the believed similarity between the CNN image recognition learning and the biological Hubel-Wiesel visual pathway. Additionally, if the hypothesis holds, this gives a defined geometry for the weight space. This would allow for the validation of trained CNNs by checking to see if, after training, the weights have taken on this geometry. A lack of the expected geometry would then point to an issue with the model. Lastly, the hypothesis indicates a possible way of accelerating the training process of a CNN model, an aspect of fundamental importance in deep learning. The CNN training procedure is a stochastic version of gradient descent optimization, with the starting weight set initialized either randomly or according to a specific distribution (e.g. Glorot initialization). One would then assume that initializing the starting weights according to the learned geometrical global structure predicted by the Carlsson-Gabrielsson hypothesis would result in faster training.

To summarize, we have the following set of observations leading up to the Carlsson-Gabrielsson Hypothesis and subsequently our work.

- In order to understand the structure of the weight space of a CNN built to classify natural images one must first understand the structure of natural images

- The structure of pixels in natural images is known to be highly non-gaussian (e.g. [54])
- Building on this non-normality Mumford et. al. showed that the set of high-variance local patches from natural images can be modeled by a high-codimensional submanifold [47]
- Carlsson, Ishkhanov, de Silva, and Zomorodian used TDA methods to show that this same set of high-variance local patches from natural images takes on the structure of a 2-dimensional submanifold [3]
- Carlsson and Gabrielsson [5] showed that the weight data from convolutional layers of CNNs seems to resemble the set of high-variance local patches from natural images studied by Mumford and also Carlsson, Ishkhanov, de Silva, and Zomorodian
- Carlsson and Gabrielsson thus applied TDA to discover whether the convolutional weights had the same structure as the high-variance local patches from natural images [18, 5]
- Carlsson and Gabrielsson were able to show for several datasets (MNIST, CIFAR-10, and SVHN) that the convolutional weights did in fact approximate a similar structure [18]
- Thus Carlsson and Gabrielsson posited the Carlsson-Gabrielsson Hypothesis [18]

## 1.3 Our Work: Objectives and Outline

Given the implications of the Carlsson-Gabrielsson hypothesis and the promise shown in their published results as well as the larger body of work in characterizing local image patches of natural images, our goal is to further the application of CNNs by studying the structure of weight spaces in CNNs. In particular, we will study the structure of weight spaces in CNNs built to classify a previously unstudied complex image set known as Dogs vs. Cats. This structure will be studied by applying TDA methods to analyze the weight spaces of convolutional layers of CNNs trained using the Dogs vs Cats data set, to determine what the underlying structure of the weight space for a CNN applied to this set is. In this work, a secondary objective will be to validate the C-G Hypothesis by extending it to a more realistic data set to demonstrate the legitimacy of the hypothesis.

To do this, the first step will be to replicate the work in [18]. Using the MNIST data set, a CNN of the same architecture will be created and trained on the data set. The results will then be compared to the work in [18]. Since

Carlsson and Gabrielsson used a proprietary software suite (Ayasdi) to achieve their results and did not publish any of their code their results have been left unverified. An attempt was made by Larsen [39] to verify the results of Carlsson and Gabrielsson using Python packages with only limited success. By replicating the work of Carlsson and Gabrielsson we will verify their results and will also demonstrate the validity of our methods and subsequent results.

After replicating the results from [5], the next step and real work will be to extend the results to a more realistic data set. To do this we will use the Dogs vs. Cats data set. This data set comes from a 2013 Kaggle competition [34] used to test the robustness of Asirra (Animal Species Image Recognition for Restricting Access) [11] which is a form of CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) developed by Microsoft in conjunction with petfinder.com. The Kaggle competition was intended to test whether Asirra was safe from attack and "benchmark the latest computer vision and deep learning approaches" [34]. As opposed to the 28x28 pixel monochromatic images of the MNIST data set the Dogs vs Cats data set consists of 200x200 pixel colored images making it much more representative of actual images.

Using this data set, we will use TDA to show that there is an underlying structure to the CNN weight space after training and that this structure approximates a manifold with non-trivial geometry. In particular, we will seek to show that this structure includes the presence of a high persistence cycle in $H_1$ i.e., the "Primary Circle", which is the same structure found in high-variance local image patches of natural images. This structure is also analogous to the arrangement of neurons in the primary visual cortex which perform edge detection.

The remaining structure of the paper is as follows. Sections 2-4 will cover the necessary theoretical background needed to understand and interpret the results. This includes a refresher of Algebra and Topology and a look at the necessary theory of Homology and Persistent Homology in Section 2, an introduction to Convolutional Neural Networks in Section 3, and in Section 4 we will cover some of the methods of Topological Data Analysis which will be used. Section 5 will give a description of the methods used before jumping in to the results in sections 6-7. In Section 6 we will give an overview of the results found by Carlsson and Gabrielsson on the MNIST data set [18] and then will show our work recreating their results and furthering the study of MNIST. Section 7 will detail the analysis of the Dogs vs Cats data set and the results of the TDA study of CNNs built for this data set. In Section 8 we will discuss the results and relevant findings with a conclusion in Section 9.

# 2 Mathematical Theory

## 2.1 Algebra Refresher

Before delving into the primary work of this paper, it is necessary to review some concepts from Abstract Algebra which will be foundational to the ideas presented later. For a more complete introduction to algebra the reader is directed to [14].

**Definition 2.1.** ***Ring***: *A ring $R$ is a set, together with two binary operations, addition (+) and multiplication ($\cdot$), defined such that the following are satisfied:*

*1. $\langle R, + \rangle$ is an abelian group.*
*2. $\cdot$ is associative*
*3. For $a, b, c \in \mathbb{R}$, $a \cdot (b+c) = (a \cdot b) + (a \cdot c)$ and $(a+b) \cdot c = (a \cdot c) + (b \cdot c)$*

**Definition 2.2.** ***Integral Domain***: *An Integral Domain $D$ is a commutative ring with unity $1 \neq 0$ (multiplicative identity) such that if $a$ and $b$ are two elements of a ring $R$ with $a \cdot b = 0$, then at least one of $a, b$ must be zero. (If neither are 0 then $a$ and $b$ are referred to as zero divisors of $R$).*

Integral domains are important because the cancellation laws for multiplication apply. In other words, if one has a polynomial equation with coefficients from an Integral Domain which can be factored into linear factors then it can be solved by setting each term equal to 0. This is not always true in a more general ring.

Taking this one step further we arrive at the concept of a field.

**Definition 2.3.** ***Field*** *A Field is a commutative ring $R$ with unity $1 \neq 0$ such that for $u \in R$ there exists $u^{-1} \in R$ where $u \cdot u^{-1} = u^{-1} \cdot u = 1$.*

Every Field is an Integral Domain but the converse does not hold. However, every Integral Domain $D$ can be enlarged to, or embedded in, a Field $F$ made up of quotients from $D$ [14]. Such a field $F$ is referred to as a field of quotients of D.

**Definition 2.4.** ***Ideal***: *If $N$ is an additive subgroup of a ring $R$ such that $aN \subseteq R$ and $Nb \subseteq R$ for $a, b \in R$, then $N$ is called an Ideal.*

If $R$ is a commutative ring with unity and $a \in R$, then the set $\{ra | r \in R\}$ is an ideal and is referred to as the **principal ideal generated by** $a$ and denoted $\langle a \rangle$. If an ideal $N = \langle a \rangle$ for some $a \in R$, then $N$ is referred to as a **Principal Ideal**.

**Definition 2.5.** ***Principal Ideal Domain (PID)*** *$R$ is a principal ideal domain if $R$ is a ring with no zero divisors and every ideal in $R$ is principal.*

A polynomial $f(t)$ with coefficients from a ring $R$ is the formal sum $\sum\limits_{i=0} a_i t^i$ where $a_i \in \mathbb{R}$, $k \in \mathbb{N}$, and only a finite number of $a_i$ are non-zero. The set of all polynomials $f(t)$ with coefficients from a ring $R$ forms a commutative ring with unity denoted $R[t]$. (Note the distinction between f(t) a single polynomial with coefficients from $R$ and $R[t]$ the set of all polynomials $f(t)$ whose $a_i \in R$). This applies to fields as well. The set $F[t]$ of polynomials with coefficients from a field $F$ is an Integral Domain - and more precisely - is a Principal Ideal Domain.

**Definition 2.6.** ***Graded Ring:*** *A graded ring is a ring $\langle R, +, \cdot \rangle$ containing a direct sum decomposition of Abelian groups $R \cong \bigoplus_i R_i, i \in \mathbb{Z}$ such that multiplication is defined by pairings $R_m \otimes R_n \to R_{m+n}$.*

**Definition 2.7.** ***Module:*** *A Module $M$ is a commutative group which can be acted on by elements from a ring $R$ satisfying the following properties, if $r, s \in R, x, y \in M$, then:*

1. $r \cdot (x + y) = (r \cdot x) + (r \cdot y)$
2. $(r + s) \cdot x = (r \cdot x) + (s \cdot x)$
3. $(r \cdot s) \cdot x = r \cdot (s \cdot x)$
4. $1 \cdot x = x$

$M$ is also referred to as an $R$-module. This is the same idea as a vector space except that the scalars are only required to be from a ring as opposed to a field in the case of the vector space.

As an example of a graded ring we can take the standard polynomial ring $R[t]$ and grade it using the *standard grading* such that $R_n = Rt^n, n \geq 0$. Clearly $R_n \otimes R_m = Rt^n \otimes Rt^m = Rt^{n+m} = R_{n+m}$, so this is a graded ring.

Similar to the ring we can also talk about a *graded module.*

**Definition 2.8.** ***Graded Module:*** *A Graded Module $M$ over a graded ring $R$ is a module with a direct sum decomposition $M \cong \bigoplus_i M_i, i \in \mathbb{Z}$ such that the action of $R$ on $M$ is defined by pairings $R_m \otimes M_n \to M_{m+n}$.*

One may recall that the Fundamental Theorem of Finitely Generated Abelian Groups gives a description for the structure and composition of every finitely generated abelian group. There is a generalization of this theorem referred to as the *Structure Theorem for Finitely Generated Modules Over a Principal Ideal Domain* [23] (henceforth referred to as the Structure Theorem). This theorem describes the structure of finitely generated $D$-modules over a PID $D$.

**Theorem 2.1.** ***Structure Theorem***: *If $D$ is a PID then every finitely generated $D$-module $M$ decomposes into the following form:*

$$M \cong D^{\beta} \oplus \bigoplus_{i=1}^{m} D/d_iD \qquad \text{for } d_i \in D, \beta \in \mathbb{Z}, \text{ such that } d_i | d_{i+1}.$$

By corollary the Structure Theorem can also be extended to a graded module over a graded PID [23].

**Corollary 2.1.1.** *: The Structure Theorem can be extended to describe $M$ if $M$ is a graded module over a graded PID $D$. In this case:*

$$M \cong (\bigoplus_{i=1}^{n} \Sigma^{\alpha_i} D) \oplus (\bigoplus_{j=1}^{m} \Sigma^{\gamma_j} D/d_jD) \text{ for } d_j \in D \text{ such that } d_j|d_{j+1}, \alpha_i, \gamma_j \in \mathbb{Z},$$

*and $\Sigma^{\alpha}$ denotes an $\alpha$-shift upward in grading, i.e., $\Sigma^{\alpha} D_m \to D_{m+\alpha}$.*

Similar to the Fundamental Theorem of Finitely Generated Abelian Groups the structures described by the Structure Theorem and its Corollary above consist of two parts, the free portion on the left and the torsional portion on the right. If $D$ is a field rather than a PID then the torsional portion disappears and we are left with only the free portion on the left. When $D$ is a PID the torsional elements on the right describe generators which may only generate a finite number of elements so the structures look like vector spaces, as they would with a field, except that some of the dimensions may appear "finite".

## 2.2 Topological Background

The primary topological structure used in TDA is the Simplicial Complex. This section contains its definition and other relevant definitions and results which will be used. For a more complete introduction to Algebraic Topology the reader is directed to [48].

### 2.2.1 Simplicial Complexes

**Definition 2.9.** ***Simplex:*** *A $k$-simplex is a convex hull of $k+1$ affinely independent points, $\sigma = conv\{u_0, u_1, ...u_k\}$. More formally,*
*Given a set $\{u_0, u_1, ...u_k\}$ of points in $\mathbb{R}^N$, such that for real scalars, $t_i$, if $\sum_{i=0}^{k} t_i = 0$ and $\sum_{i=0}^{k} t_i u_i = 0$ then $t_i = 0$ $\forall i = 0..k$. Then the $k$-simplex $\sigma$ is the set of points $x \in \mathbb{R}^N$ such that $x = \sum_{i=0}^{k} t_i u_i$ where $\sum_{i=0}^{k} t_i = 1$.*

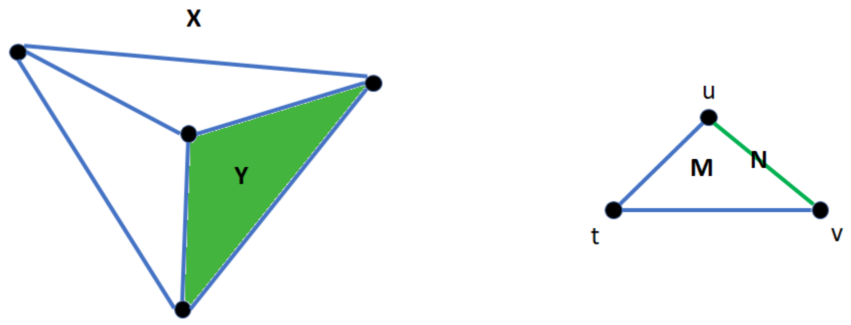


Figure 2.1: *On the left $X$ is 3-simplex and $Y$ is a face of $X$. On the right $M$ is a 2-simplex consisting of the points $t, u, v$ and $N$ is a face of $M$ containing $u$ and $v$*

If $\sigma$ is a simplex then a *face* of $\sigma$ is the convex hull of a non-empty subset of the $u_i$ (See Figure 2.1).

Using this definition, a *Simplicial Complex* is just a finite collection of simplices $K$ with the following properties.

**Definition 2.10.** ***Simplicial Complex*** *A finite collection $K$ of simplices such that*

1. *$\sigma \in K$ and $\tau$ a face of $\sigma$ implies $\tau \in K$*
2. *$\sigma, \tau \in K$ implies $\sigma \cap \tau$ is either empty or a face of both.*

Above is an example of a Simplicial Complex $K = \{\{t\}, \{u\}, \{v\}, \{t, u\}, \{u, v\}, \{v, t\}, \{t, u, v\}\}$. As one can see the face of every simplex in $K$ is also in $K$, so $\{t, u, v\} \in K \Rightarrow \{t, u\} \in K$. Now it is not the case that every point must be connected. For instance, we can remove the simplex $\{t, v\}$ from $K$; however, in this case we no longer have a Simplicial Complex because $\{t, v\}$ is a face of $\{t, u, v\}$. If we also remove $\{t, u, v\}$ from $K$ then we once again have a valid Simplicial Complex $K' = \{\{t\}, \{u\}, \{v\}, \{t, u\}, \{u, v\}\}$ (pictured in Figure 2.3).

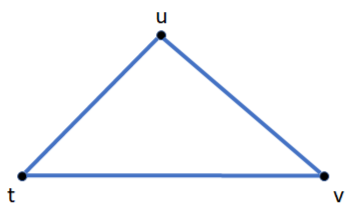


Figure 2.2: *A simplicial complex* $K = \{\{t\}, \{u\}, \{v\}, \{t, u\}, \{u, v\}, \{v, t\}, \{t, u, v\}\}$*, i.e., the Simplex from Figure 2.1 and its faces.*

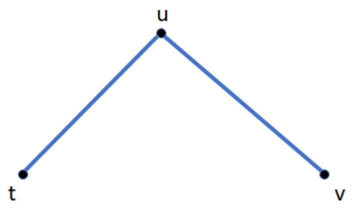


Figure 2.3: *The simplicial complex* $K$ *from Figure 2.2 with the simplices* $\{t, v\}$ *and* $\{t, u, v\}$ *removed.*

One problem with the use of Simplicial Complexes is that it requires a geometric realization. To get around this we can define an *Abstract Simplicial Complex* which can be defined and used in a more general topological space without having to worry about its geometric representation.

**Definition 2.11.** ***Abstract Simplicial Complex:*** *An Abstract Simplicial Complex is a finite collection of sets* $A$ *such that* $\alpha \in A$ *and* $\beta \subseteq \alpha$ *implies* $\beta \in A$.

This definition allows the use and description of Simplicial Complexes without the need for a geometric realization. That being said we can essentially use these definitions interchangeably since the Geometric Realization Theorem [8] states:

**Theorem 2.2.** ***Geometric Realization Theorem*** *Every abstract simplicial complex of dimension* $d$ *has a geometric realization in* $\mathbb{R}^{2d+1}$.

So even if a specific geometric realization is not known we can talk about the simplicial complex in terms of a general geometric realization since one is known to exist.

### 2.2.2 Constructing A Simplicial Complex

With this idea in hand we turn now to a discussion of constructing simplicial complexes. This is a complex and broad topic that depends, among other properties, on the type of data being used to build the complex and the space in which that data lies. For the sake of simplicity and application to the results in this paper the discussion is restricted to point cloud sets in a metric space.

**Definition 2.12.** ***Metric:*** *A metric $d$ on a set $X$ is a function $d : X \times X \to \mathbb{R}$ satisfying the following properties:*

1. *$d(x, y) \geq 0 \ \forall x, y \in X$ and $d(x, y) = 0$ iff $x = y$*
2. *$d(x, y) = d(y, x) \ \forall x, y \in X$*
3. *$d(x, y) + d(y, z) \geq d(x, z) \ \forall x, y, z \in X$*

A metric is often referred to as the distance between two points. Given a metric $d$ we can define the set known as the $\epsilon$-ball.

**Definition 2.13.** *Given a metric $d$, a point $x$ and real value $\epsilon > 0$, the $\epsilon$-ball centered at $x$, denoted $B_d(x, \epsilon)$, is defined as:*

$$B_d(x, \epsilon) = \{y \in X | d(x, y) < \epsilon\}$$

**Definition 2.14.** ***Metric Space:*** *If $X$ is a topological space, $X$ is said to be metrizable if there exists a metric $d$ on the set $X$ that induces the topology of $X$. A Metric Space $(X, d)$ is a topological space $X$ together with a metric $d$ such that the topology of $X$ is induced by $d$.*

**Definition 2.15.** ***Point Cloud:*** *A point cloud is a finite set of vectors $\{v_i\} \in \mathcal{M}$ a metric space. Almost always, as will be the case with the results throughout this paper, the metric space is $\mathbb{R}^n$.*

To start with, let $X = \{x_i : i = 1..n\}$ be our finite point cloud data set. Using this data set, a simplicial complex can be constructed using the data. There are many different ways to construct the complex, a discussion of which is beyond the scope of this paper; however, below are two examples of ways in which a simplicial complex can be constructed from a point cloud.

**Čech Complex**

**Definition 2.16.** *Given a collection of points $X = \{x_i\}$ in a metric space and $\epsilon > 0$, the Čech Complex $C_\epsilon$ is the abstract simplicial complex where for each $\sigma \subseteq X, \sigma \in C_\epsilon$ iff*

$$\bigcap_{x \in \sigma} B(x, \tfrac{\epsilon}{2}) \neq \emptyset.$$

**Definition 2.17.** ***Nerve:*** *Given a finite collection of sets $F$, the Nerve of $F$, $\mathcal{N}(F) = \{X \subseteq F | \bigcap X \neq \emptyset\}$. That is, the Nerve consists of all non-empty subcollections of $F$ whose sets have a non-empty intersection. The Nerve of $F$ is an abstract simplicial complex whose vertices are elements of $F$ and whose simplices are the subcollections with non-empty intersection.*

**Theorem 2.3.** ***Nerve Theorem:*** *Let $F$ be a finite collection of closed, convex sets in Euclidean space. Then the nerve of $F$ and the union of the sets in $F$ have the same homotopy type.*

The Čech Complex is the Nerve of the point cloud $X$ (i.e., $C_\epsilon = \mathcal{N}(X)$). So by the Nerve Theorem, $C_\epsilon$ is homotopy equivalent to the union of the $\frac{\epsilon}{2}$ balls around $X$. Because of this the Čech Complex behaves nicely. For $X = \{x_i | x_i \in \mathbb{R}^d\}$ while the complex can have dimension much higher than $d$ the Nerve Theorem tells us that the complex behaves like a subset of $\mathbb{R}^d$. However, despite this nice property, the computation of the Čech Complex is quite difficult and cumbersome [19]. As a more computationally efficient alternative we can compute the Rips Complex.

**Rips Complex**

**Definition 2.18.** *Given a collection of points $X = \{x_i\}$ from a metric space, the Rips Complex $R_\epsilon$ is the abstract simplicial complex where for each $\sigma \subseteq X, \sigma \in R_\epsilon$ iff for $x, y \in \sigma, d(x, y) < \epsilon$.*

The reader is directed to [19] for a more in-depth comparison of the Čech and Rips Complexes, a summary of which is presented below. The important take away is that Rips Complexes are both easier to compute and require less storage than Čech Complexes. The Rips Complex is a flag complex meaning that a set of $k + 1$ vertices spans a $k$-simplex if and only if any two span a 1-simplex, so it is defined by its 1-skeleton. Due to this, the complex can be stored as a graph and the entire boundary operator is not needed like it is for the Čech Complex. Thus, even though the Rips Complex typically has more simplicies than the Čech Complex (since the Čech Complex is a subcomplex of the Rips Complex) it is more efficient to compute and store. However, unlike the Čech Complex it does not necessarily behave like an $n$ dimensional space [19].

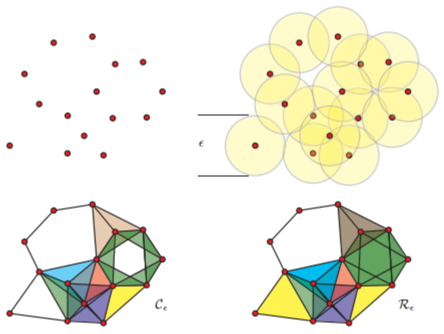


Figure 2.4: *On the left we show the Cech Complex and the corresponding simplicial complex and on the right the Rips Complex and its simplicial complex.(Image taken from [19])*

### 2.2.3 Homology

Let $\sigma$ be a simplex, then it has a vertex set $\{v_0, v_1, ...v_k\}$. We say that two orderings of the vertex set are equivalent if they differ from one another by an even permutation. So if we assume $\dim\{\sigma\} \neq 0$ ($\dim\{\sigma\} = |\{v_i\}_{i=0..k}| = k$) then the orderings fall into two equivalence classes, each of which are called an *orientation* of $\sigma$. A simplex together with its ordering is referred to as an *oriented simplex*. In Figure 2.5 $[t, u, v]$ is an oriented 2-simplex.

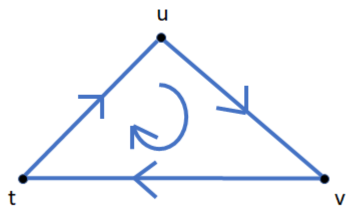


Figure 2.5: *An oriented simplicial complex* $K = \{[t], [u], [v], [t, u], [u, v], [v, t], [t, u, v]\}$

**Definition 2.19.** *Given a simplicial complex $K$ and $p \in \mathbb{N}$, a **p-chain** on $K$ is a function $c$ from the set of oriented $p$-simplicies of $K$ to a Ring $R$ such that:*

*(1) $c(\sigma) = -c(\sigma')$ if $\sigma$ and $\sigma'$ are opposite orientations of the same simplex.*

*(2) $c(\sigma) = 0$ for all but finitely many oriented $p$-simplices of $K$.*

An example of a $p$-chain is the *Elementary Chain* corresponding to $\sigma$.

**Definition 2.20.** ***elementary chain:*** *If $\sigma$ is an oriented simplex, then the elementary chain $c$ corresponding to $\sigma$ is defined by*

$$c(\tau) = \begin{cases} 1 & : \tau = \sigma \\ -1 & : \tau = \sigma' \text{ where } \sigma' \text{ is the opposite orientation of } \sigma \\ 0 & : \text{otherwise} \end{cases}$$

$p$-chains can be added together by adding their values. So, if $c, c'$ are $p$ chains, then $(c + c')(\sigma) = c(\sigma) + c'(\sigma)$. This results in abelian groups $C_p(K)$ for each $p \in \mathbb{N}$, which are referred to as the $p$-th chain groups. For $p = 0$ and $p > \dim\{K\}$ we define $C_p(K)$ as the trivial group. Using this we can then define the *Boundary Operator.*

**Lemma 2.4.** *A basis for $C_p(K)$ can be obtained by orienting each $p$-simplex and using the corresponding elementary chain as a basis.*

For proof see [48]

**Definition 2.21.** ***Boundary Operator:*** *The boundary operator $\partial_p : C_p(K) \to C_{p-1}(K)$ is a homomorphism defined as follows. Given an oriented simplex $\sigma = [v_0, v_1, v_2, ...v_p]$ with $p > 0$*

$$\partial_p\sigma = \partial_p[v_0, ...v_p] = \sum_{i=0}^{p}(-1)^i[v_0, ...\hat{v}_i, ...v_p]$$

*where $\hat{v}_i$ means that the vertex $v_i$ is to be deleted from the array.*

As an example, consider the oriented 2-simplex $[t, u, v]$ from Figure 2.5. Then: $\partial_2[t, u, v] = (-1)^0[\hat{t}, u, v] + (-1)^1[t, \hat{u}, v] + (-1)^2[t, u, \hat{v}] = [u, v] - [t, v] + [t, u]$ This is the set of oriented 1-simplices pictured in Figure 2.6.

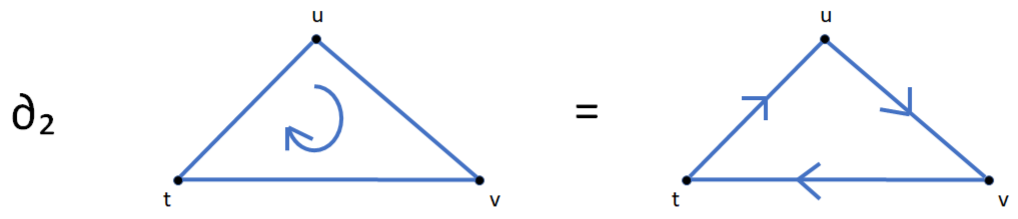


Figure 2.6: *The $\partial_2$ boundary operator acting on the oriented simplex $\sigma = [t, u, v]$*

Using the boundary operator on each of the chain groups we obtain the following sequence of groups:

$$\begin{tikzcd} \cdots \arrow[r, "\partial_{p+2}"] & C_{p+1} \arrow[r, "\partial_{p+1}"] & C_p \arrow[r, "\partial_p"] & C_{p-1} \arrow[r, "\partial_{p-1}"] & \cdots \end{tikzcd}$$

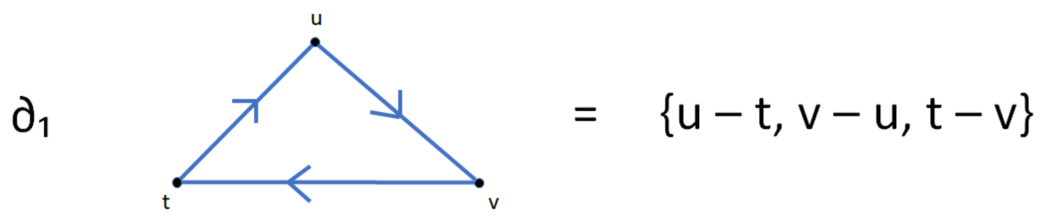


Figure 2.7: *The $\partial_1$ boundary operator acting on the oriented simplices* $\{\sigma_1, \sigma_2, \sigma_3\} = \{[t, u], [u, v], [v, t]\}$

This sequence is referred to as a *chain complex* and is denoted $C_*$.

This boundary operator gives rise to the two groups we need to build the *Homology Groups*. First, we have the kernel of $\partial_p : C_p(K) \to C_{p-1}(K)$ which is referred to as the group of $p$-cycles and denoted $Z_p(K)$. Secondly, we have the image of $\partial_{p+1} : C_{p+1}(K) \to C_p(K)$ which is called the group of $p$-boundaries and is denoted $B_p(K)$. Using these two groups we arrive at the construction of the $p$th *homology group.*

**Theorem 2.5.** *The boundary of a $p+1$-chain is automatically a $p$-cycle (i.e., $B_p(K) \subseteq Z_p(K)$)*

*Proof.*
Let $[v_0, v_1, ..., v_p]$ be a $p+1$ chain. Then $\partial_{p+1}[v_0, v_1, ..., v_p]$ is a $p$-cycle if $\partial_p \partial_{p+1}[v_0, v_1, ..., v_p] = 0$.

$$\begin{aligned}\partial_p \partial_{p+1}[v_0, v_1, ..., v_p] &= \sum_{i=0}^{p} (-1)^i \partial[v_0, ...\hat{v}_i, ..., v_p] \\ &= \sum_{i=0}^{p} (-1)^i \sum_{j=0}^{i-1} (-1)^j [v_0, ..., \hat{v}_i, ...\hat{v}_j, ..., v_p] \\ &\qquad + \sum_{i=0}^{p} (-1)^i \sum_{j=i+1}^{p} (-1)^{j-1} [v_0, ..., \hat{v}_j, ...\hat{v}_i, ..., v_p]\end{aligned}$$

For any pair $(i, j)$ in the first sum there exists a pair $(j, i)$ in the second sum and so $(-1)^i(-1)^j[v_0, ..., \hat{v}_i, ...\hat{v}_j, ..., v_p] + (-1)^j(-1)^{i-1}[v_0, ..., \hat{v}_i, ...\hat{v}_j, ..., v_p] = 0$. Similarly every pair $(i, j)$ in the second sum has a corresponding pair $(j, i)$ in the first sum.

Thus:

$$\begin{aligned}&\sum_{i=0}^{p} (-1)^i \sum_{j=0}^{i-1} (-1)^j [v_0, ..., \hat{v}_i, ...\hat{v}_j, ..., v_p] \\ &\qquad + \sum_{i=0}^{p} (-1)^i \sum_{j=i+1}^{p} (-1)^{j-1} [v_0, ..., \hat{v}_j, ...\hat{v}_i, ..., v_p] \\ &\qquad = 0\end{aligned}$$

Therefore the boundary of a $p+1$-chain is automatically a $p$-cycle. □

**Definition 2.22.** ***Homology Group:*** *The pth homology group is defined as* $H_p(K) = Z_p(K)/B_p(K)$.

Since each boundary of a $p+1$-chain is automatically a $p$-cycle (i.e., $B_p(K) \subseteq Z_p(K)$) this is well defined.

As an example, consider the simplicial complex $K$ pictured in Figure 2.5. We have already shown in Figures 2.6 and 2.7 the results of the $\partial_1$ and $\partial_2$ operators. We know that $H_2(K) = Z_2(K)/B_2(K)$, and we know that $\partial_2[t,u,v] \in Z_2(K)$. Since there are no 3-simplices then $B_2(K) = \{0\}$, so $H_2(K) = Z_2(K) \cong \mathbb{Z}$. Additionally for $H_1(K)$ since $B_1(K) = C_1(K)$, then $Z_1(K) = B_1(K)$ and thus $H_1(K) = \{0\}$.

As another example consider the simplicial complex pictured in Figure 2.5 where we add a point $\{d\}$ and directed simplices $\{[c,d],[d,a]\}$ as pictured below in Figure 2.8. Then:

$$\partial_1\{[a,b],[b,c],[c,a],[c,d],[d,a]\} = \{b-a, c-b, a-c, d-c, a-d\}$$
$$\partial_2\{[a,b,c]\} = bc - ac + ab = \{[a,b],[b,c],[c,a]\}$$

So

$$B_1(K) = \{[a,b],[b,c],[c,a]\}$$
$$Z_1(K) = \{[a,b],[b,c],[c,a],[d,a]\}$$

Thus $H_1(K) = Z_1(K)/B_1(K) = \mathbb{Z}_2$

And similarly to the previous example there are no 3-simplices and there is at least one 2-cycle so $H_2(K) = Z_2(K) \cong \mathbb{Z}$.

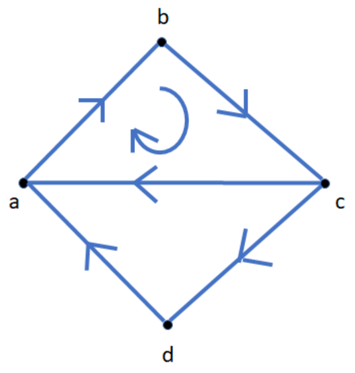


Figure 2.8: *Oriented simplicial complex* $K = \{[a],[b],[c],[d],[a,b],[b,c],[c,a],[c,d],[d,a],[a,b,c]\}$

Recall that the Fundamental Theorem of Finitely Generated Abelian Groups tells us that for any finitely generated abelian group $G$, $G$ can be decomposed into a free abelian subgroup $H$ with finite rank $\beta$ (the Betti number of $G$) and its Torsion Subgroup $T$ (where $G = H \oplus T$). In our context then, the $p$th Betti number is the rank of the $p$th homology group, i.e., $\beta_p = \text{rank}\{H_p\}$. This turns out to be quite important as Poincare' proved that the Betti numbers are topological invariants, thus in the context of data analysis they provide important information about the underlying structure of the data.

A few more relevant definitions:

**Definition 2.23.** *Two cycles $c, c' \in Z_p(K)$ are* ***homologous*** *if they differ by a boundary, i.e., there exists a $p+1$-chain $d$ such that $c - c' = \partial_{p+1} d$.*

**Definition 2.24.** *A* ***homotopy*** *is a family of maps $f_t : X \to Y, t \in [0,1]$, such that the associated map $F : X \times [0,1] \to Y$ given by $F(x,t) = f_t(x)$ is continuous. Then $f_0, f_1 : X \to Y$ are homotopic via the homotopy $f_t$, denoted $f_0 \simeq f_1$.*

**Definition 2.25.** *A map $f : X \to Y$ is called a* ***homotopy equivalence*** *if there is a map $g : Y \to X$ such that $f \circ g \simeq 1$ and $g \circ f \simeq 1$. Then, $X$ and $Y$ are homotopy equivalent. We denote this by $X \simeq Y$.*

Given two cycles $\sigma, \tau$ with a shared boundary occurring in opposite directions, the sum $\sigma + \tau$ eliminates the shared boundary thus resulting in a cycle homotopic to $\sigma$. Since this holds for any boundary $\tau$ we can look for equivalence classes $\sigma + B_p$ for a $p$-cycle. However, this is the group $Z_k/B_k$. Homology thus lets us measure the cycles up to the homology equivalence relation, i.e., those cycles which are homologous, because it is precisely the cycles modulo the set of boundaries.

## 2.3 Persistent Homology

### 2.3.1 Filtration

For many reasons having a single simplicial complex may not be sufficient. For one, in our context, we are building a simplicial complex out of point cloud data, and as such we must have some method of forming the Simplicial Complex, e.g. Čech or Rips Complex. The problem with this is that these methods rely on a threshold to determine whether two vertices are connected; however, this begs the question, what is an appropriate threshold? There is not an obvious choice for threshold value, but this value has a major impact on the structure of the complex. For example, choosing a threshold $\epsilon$ small enough will mean that there are only 0-simplexes whereas if $\epsilon$ is chosen sufficiently large, then there will exist an $n+1$ simplex where $n$ is the number of points in the point cloud. Persistent Homology offers a way of looking at the structure over a set of increasing threshold values allowing one to study how the structure changes and learn information that is true about the structure of point cloud rather than information that is true only at a specific threshold value. The following section outlines the necessary theory for Persistent Homology.

**Definition 2.26.** *Let $K$ be a simplicial complex defined from a point cloud data set $X$ with some metric $d$. Then the **coarseness** $\delta$ of $K$ is defined as the maximum distance between any two points contained in the same simplex, i.e.,*

$$\delta = \max\{d(x,y) \mid x, y \in X, x, y \in k \in K\}$$

In many cases it may not be obvious what this threshold value should be or it may be desirable to see how the properties of the complex change over various values for $\delta$. To do this we can use a filtration on the complex.

**Definition 2.27.** ***Filtration:*** *A filtration is a function $f$ on $K$ that results in a nested subsequence of complexes $\emptyset = K^0 \subseteq K^1 \subseteq K^2 \subseteq ... \subseteq K^m = K$.*

In this case we call $K$ a filtered complex. Using a filtered complex we can then talk about the homology groups of each complex $K^i$. We denote $C_p^i$, $Z_p^i$, and $B_p^i$ as the $p$-chains, $p$-cycles and $p$-boundaries, respectively, of $K^i$.

**Definition 2.28.** *The $j$-persistent $p$-th homology group in the $k$-th dimension is:*

$$H_p^{i,j} = Z_p^i / (B_p^{i+j} \cap Z_p^i)$$

So then for each $i, j \in \mathbb{N}, 0 \leq i \leq m$ and $0 \leq j \leq m-i$, there exist $j$-persistent $p$-th homology groups $H_p^{i,j}$. These exist for each dimension $p > 0$ as well and if we generalize and let $K^l = K$ for $l > m$ then we can remove the restriction $j \leq m - i$.

To better understand filtrations, consider once again the Čech Complex. By increasing the value for $\epsilon$, a filtration can be constructed out of the Čech Complexes built for each epsilon value (see Figure 2.9). This is by no means the only way to create a filtered complex, but a thorough discussion of methods is beyond the scope of this paper.

Returning to our discussion of filtered complexes - using a filtration allows us to see how the topological properties derived from the homology groups change at different scales. This is particularly useful when dealing with noisy data. Since all sampled data inherently contains noise, we want to be able to

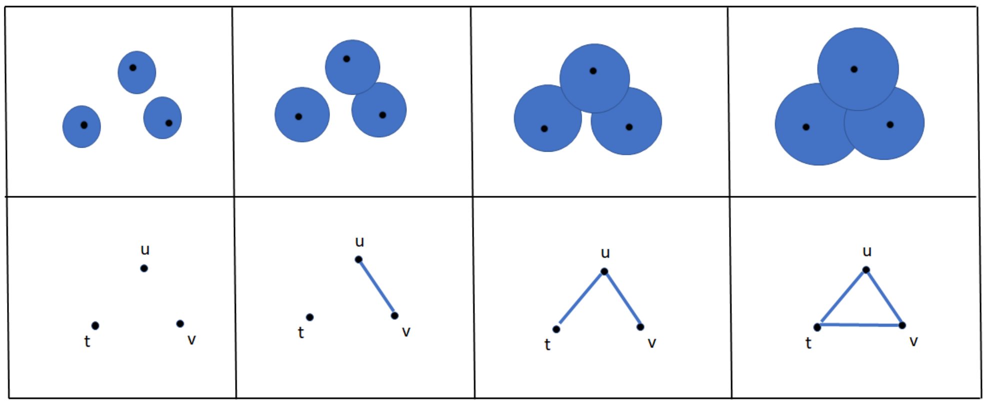


Figure 2.9: *On the top row we show the filtration on a data set containing three points. On the bottom row we show the simplicial complex created at each step of the filtration*

filter out the noise and determine what the ambient non-noisy space really is. By using a filtration, we can view the properties at various scales and ignore "features" which appear only at specific scales and thus focus our attention on those properties that persist across many or all scales.

Using a filtration on a complex $K$, we create groups $H_p^{i,j}$ which we call the persistent homology groups and with them introduce the subfield of *Persistent Homology*. Let us recall what these groups represent. As defined above $H_p^{i,j} = Z_p^i/(B_p^{i+j} \cap Z_p^i)$ so $H_p^{i,j}$ is the set of homologous non-bounding $p$-cycles which are not boundaries in $K^i$ and "survive" or do not become boundaries until $K^{i+j}$. The persistent homology groups then represent topological properties which persist through levels of filtration. The idea is that important topological features are precisely those with a long lifetime, i.e., those that persist through many levels of filtration [65]. Intuitively then, by focusing on those properties that survive through many levels of filtration, we can ignore the properties which arise as features of the noise on our data and as such only persist through a small number of filtration levels.

Just as with standard homology groups we can derive topological information from the ranks of the persistent homology groups, i.e., their Betti Numbers.

**Definition 2.29.** *the $j$-persistent $p$-th Betti number of $K^i$ for each $i, j, p$, denoted $\beta_p^{i,j}$ is defined as the rank of the free subgroup of $H_p^{i,j}$.*

By looking at the $j$-persistent Betti Numbers where $j$ is at least some value $M > 1$ we can observe the topological properties which persist for at least $M$ levels of the filtration. Thus by tuning the parameter $M$ we can remove the spurious features created by noise. We will show later that $\beta_p^{i,j}$ is the number of equivalence classes of non-bounding $p$-cycles in $H^i$ with homologous boundaries in $H^{i+j}$.

Since $K^i \subseteq K^{i+j}$, we have inclusion for $Z_p^i \subseteq Z_p^{i+j}, C_p^i \subseteq C_p^{i+j}, B_p^i \subseteq B_p^{i+j}$, and $Z_p^i \subseteq B_p^i \subseteq C_p^i$, thus we can define a map $\eta_p^{i,j} : H_p^i \to H_p^{i+j}$. Using this then the image of $\eta_p^{i,j}$ is equivalent to $H_p^{i,j}$.

### 2.3.2 Persistence

We have so far discussed the idea of *persistence* but let us define it formally.

**Definition 2.30.** *If $z \in K^i$ is a non-bounding $k$-cycle such that there is no homologous $k$-cycle $z' \in K^l$ for $l < i$ then we say that $z$ is* ***created*** *at time $i$.*

**Definition 2.31.** *If $z \in K^j$ is a $k$-cycle that is also a boundary we say that $z$* ***dies*** *at time $j$.*

**Definition 2.32.** *If $z$ is a non-bounding $k$-cycle that is created at time $i$ and $z'$ is a $k$-cycle that is homologous to $z$ (i.e., $z \sim z'$) that dies at time $j$ then the* ***persistence*** *of $z$ is pers$\{z\} = j - i - 1$. If there exists $z' \in K$ such that $z \sim z'$ (i.e., $z$ does not die) then pers$\{z\} = \infty$. That is, the persistence of a Persistent Homology Group $H_p^{i,j}$ is the number of filtration levels through which the non-bounding cycles within the group persist (i.e., exist).*

**Definition 2.33.** *A $\mathcal{P}$-interval is an ordered pair $(i, j)$ with $0 \leq i < j, i, j \in \mathbb{Z}^{\infty} = \mathbb{Z} \cup \{+\infty\}$*

As we will discuss in the next section, these $\mathcal{P}$-intervals turn out to be quite important and useful for describing the structure of the persistent homology groups.

### 2.3.3 Persistence Complex

We have so far seen how the boundary operator takes us between the chain groups and we have seen how to move between the levels in the filtration. We attempt now to merge these two concepts into one structure and thus obtain a single unifying structure which describes the movement across the chain groups of different dimensions and across different levels of the filtration. By doing so we will construct a more formal and algebraic description of persistent homology.

**Definition 2.34.** *A* ***persistence complex*** *$\mathcal{C}$ is a family of chain complexes $\{C_*^i\}_{i \geq 0}$ over a ring $R$, together with the maps between the chains $f^i : C_*^i \to C_*^{i+1}$.*

This produces the following structure:

$$C_*^0 \xrightarrow{f^0} C_*^1 \xrightarrow{f^1} \cdots \xrightarrow{f^{p-1}} C_*^p$$

Recall that a filtered complex $K$ is made up of a set of subcomplexes $K^i$ which can be traversed using the filtration function. Each subcomplex $K^i$ is itself a chain complex consisting of the chain groups on each set of $p$-simplicies which can be traversed using the boundary operator. So each chain complex produces the following:

$$\cdots \xrightarrow{\partial_{p+2}} C_{p+1} \xrightarrow{\partial_{p+1}} C_p \xrightarrow{\partial_p} C_{p-1} \xrightarrow{\partial_{p-1}} \cdots$$

Thus, expanding this out, we get the following two-dimensional structure:

$$\begin{tikzcd}
{} \arrow[d, "\partial_2"'] & {} \arrow[d, "\partial_2"'] & {} \arrow[d, "\partial_2"'] & \\
C_2^0 \arrow[r, "f^0"] \arrow[d, "\partial_1"'] & C_2^1 \arrow[r, "f^1"] \arrow[d, "\partial_1"'] & C_2^2 \arrow[r, "f^2"] \arrow[d, "\partial_1"'] & \cdots \\
C_1^0 \arrow[r, "f^0"] \arrow[d, "\partial_0"'] & C_1^1 \arrow[r, "f^1"] \arrow[d, "\partial_0"'] & C_1^2 \arrow[r, "f^2"] \arrow[d, "\partial_0"'] & \cdots \\
C_0^0 \arrow[r, "f^0"] & C_0^1 \arrow[r, "f^1"] & C_0^2 \arrow[r, "f^2"] & \cdots
\end{tikzcd}$$

**Definition 2.35.** *A* ***Persistence Module*** $\mathcal{M}$ *is a family of $R$-modules $M^i$ together with homomorphisms $\phi^i : M^i \to M^{i+1}$.*

**Theorem 2.6.** *The homology of a persistence complex is a persistence module.*

*Proof.* We have already discussed that the homology of a filtered complex produces the groups $H_p^i$. So, let $\sigma + B_p^i, \tau + B_p^i \in H_p^i$ and $r \in R$ for some ring $R$, then for any $b \in B_p^i$

1. Recall that $\sigma, \tau, b$ are chains and thus are of the form $\sum_j n_j(\lambda_j)$ for $\lambda_j \in K^i$ and thus $r\cdot[(\sigma+b)+(\tau+b)] = [r\cdot(\sigma+b)]+[r\cdot(\tau+b)]$. By the same reasoning
2. $(r+s)\cdot(\sigma+b) = [r\cdot(\sigma+b)] + [s\cdot(\sigma+b)]$
3. $(r\cdot s)\cdot(\sigma+b) = [r\cdot(\sigma\cdot b)] + [s\cdot(\sigma\cdot b)]$
4. $1\cdot\sigma = \sigma$

Therefore $H_p^i$ is an $R$-Module.
We have already defined the map $\eta_p^{i,j} : H_p^i \to H_p^{i+j}$ which is clearly a homomorphism.
Thus the homology of a persistence complex $\{K^i\}$ with maps $\eta_p^{i,i+1} : H_p^i \to H_p^{i+1}$ is a persistence complex. □

**Definition 2.36.** *A persistence complex $\{C_*^i, f^i\}$ or persistence module $\{M^i, \phi^i\}$ is of* ***finite type*** *if each component complex - or module in the case of the persistence module - is a finitely generated $R$-module and the maps $f^i$ or $\phi^i$ respectively are isomorphisms for some $i \geq m,\ i, m \in \mathbb{N}$.*

**Lemma 2.7.** *The persistence module $\{H_p^i, \eta_p^{i,i+1}\}$ is of finite type.*

This result is trivial from the previous results.

We have now defined a structure for our persistent homology, but at this point what that structure actually looks like, if there is any pattern to it, still alludes us. As such, we now seek to show what this structure looks like and produce a unifying definition for its format.

**Correspondence**
Taking our persistent module $\mathcal{M} = \{M^i, \phi^i\} = \{H_p^i, \eta_p^{i,i+1}\}_{i\geq 0}$ over some ring $R$ we can equip $R[t]$ with the standard grading (defined in Section 2.1), and thus define a graded module:

$$\alpha(\mathcal{M}) = \bigoplus_{i=0}^{\infty} M^i$$

The $R$-module structure is then given by the sum of the structures on each of the individual components where the action of $t$ shifts the elements up one level in grading. More concretely:

$$t \cdot (m^0, m^1, m^2, ...) = (0, \phi^0(m^0), \phi^1(m^1), \phi^2(m^2), ...)$$

**Theorem 2.8.** *The correspondence $\alpha$ defines an equivalence of categories between the category of persistence modules of finite type over $R$ and the category of finitely generated non-negatively graded modules over $R[t]$.*

The proof is the Artin-Rees Lemma which can be found in [12]

It turns out that this is now enough to produce a concrete structure for the persistence homology groups which we will show in the next section.

**Decomposition**
Looking at the correspondence defined above, we arrive at two possibilities. If the ground ring $R$ is a field, then as we have already noted, the graded ring $R[t]$ (more appropriately denoted $F[t]$) is a Principal Ideal Domain and so by its construction the only graded ideals are homogeneous of the form $(t^n) = t^n \cdot R[t]$ for $n \geq 0$. Therefore, its structure is exactly that which is defined in Corollary 2.1.1. Specifically in our case then the structure is:

$$(\bigoplus_{i=1}^{n} \Sigma^{\alpha_i} F[t]) \oplus (\bigoplus_{j=1}^{m} \Sigma^{\gamma_j} F[t]/t^{n_j})$$

The other possibility occurs when $R$ is not a field and thus $R[t]$ is not a PID. In this case no simple classification for the graded $R[t]$ module is known to exist. An example of this is the modules over $\mathbb{Z}[t]$ which are known to be difficult to classify. While it is conjectured that no simple classification exists for such modules over non-PIDs this has yet to be proven.

Turning back to our correspondence over fields, it turns out that an even simpler correspondence exists for these $F[t]$ modules which can be obtained by using the $\mathcal{P}$-intervals described in Definition 2.33. Using these we can define a correspondence $Q$ from $\mathcal{I}$ the set of $\mathcal{P}$-intervals to an $F[t]$ module where $Q$ is defined as follows:

For $i, j \in \mathbb{Z},\ Q(i,j) = \Sigma^i F[t]/t^{j-i}$ and for $(i, +\infty),\ Q((i, +\infty) = \Sigma^i F[t]$.
Thus for any set $\mathcal{I} = \{(i_1, j_1), (i_2, j_2), ..., (i_k, j_k)\}$ of $\mathcal{P}$-intervals, $Q(\mathcal{I}) = \bigoplus_k^n Q(i_k, j_k)$

Clearly if $F[t]$ is of the form $(\bigoplus_{k=1}^{n} \Sigma^{\alpha_k} F[t]) \oplus (\bigoplus_{l=1}^{m} \Sigma^{\gamma_l} F[t]/t^{n_l})$ then for any $\Sigma^{\alpha_k} F[t]$ there exists an $i \in \mathbb{N}, i = \alpha_k$ and thus there exists $(i, +\infty)$ in the set of $\mathcal{P}$-intervals.
Similarly for any $\Sigma^{\gamma_l} F[t]/t^{n_l}$, there exists $i, j \in \mathbb{N}$ such that $i = \gamma_l$ and $j = n_l + i$. By our definition $n_l \geq 0$, so $j > i$ and thus there exists $(i, j)$ in the set of $\mathcal{P}$-intervals.
Since $F[t]$ is finitely generated there is a finite set of intervals $(i, j)$ and $(i, +\infty)$. Therefore $Q$ is a bijection between this set and $Q(S)$.

Now using this correspondence we can simplify our previous correspondence and restate Theorem 2.8 as follows:

**Corollary 2.8.1.** *The correspondence $\mathcal{I} \to Q(\mathcal{I})$ defines a bijection between the finite sets of $\mathcal{P}$-intervals and the finitely generated graded modules over a graded ring $F[t]$ when $F$ is a field (then $F[t]$ is a PID). Consequently, the isomorphism classes of persistence modules of finite type over $F$ are in bijective correspondence with the finite sets of $\mathcal{P}$-intervals.*

This means that computing the set of $\mathcal{P}$-intervals is equivalent to computing the persistent homology over a field, which we arrive at by the following lemma:

**Lemma 2.9.** *Let $\mathcal{T}$ be the set of triangles defined by $\mathcal{P}$-intervals for the $k$-dimensional persistence module (described in Section 2.3.6). The rank $\beta_k^{l,p}$ of $H_k^{l,p}$ is the number of triangles in $\mathcal{T}$ containing the point $(l, p)$.*

This lemma is just a restatement of the K-Triangle Lemma a definition and proof of which can be found in [9].

**Recap** Let us review what we have done so far. We started with a set of point cloud data $X = \{x_i\}$. Using this, we create a simplicial complex $K$ from $X$ and define a filtration $f$ on $K$ resulting in a nested sequence of complexes $\emptyset = K^0 \subseteq K^1 \subseteq K^2 \subseteq \cdots \subseteq K^m = K$. We then calculate the homology group for each dimension on $K^i$, resulting in the groups $H_p^i$ described by their rank $\beta_p^i$. If we take these groups over a field, they become vector spaces and thus our Structure Theorem (Theorem 2.1) can be used to describe their structure. Expanding our view, we need to produce a description for the persistent homology on the full filtration. From Theorem 2.6 we know that this persistent homology is in fact a persistence module $\mathcal{M}$. Thus, taken over a field $F$, we can define a graded module taken over $F[t]$. By Corollary 2.1.1 we know the structure of this graded module and thus we know that there exist compatible bases for the vector spaces. In the next section we will look at how to compute such bases.

### 2.3.4 Computation

Now that we have discussed persistent homologies and have built a structure with which we can describe them, we will take a look at how to effectively compute these persistent homologies. We begin by illustrating the standard reduction algorithm and then show how it can be used to compute homology groups. From there we show how it can be extended to graded modules which of course will be our step from the homology groups to the persistent homology. We then show how the algorithm can be extended to compute persistent homology over a field and then finally over a PID.

#### Standard Reduction Algorithm

**Definition 2.37.** *Let $G$ and $G'$ be free abelian groups with bases $a_1, ..., a_n$ and $a'_1, ...a'_n$ respectively. If $f : G \to G'$ is a homomorphism then $f(a_j) = \sum_{i=1}^{m} \lambda_{ij} a'_i$ for unique integers $\lambda_{ij}$ . The matrix $(\lambda_{ij})$ is called the **matrix of** $f$ relative to the given basis for $G$ and $G'$.*

**Theorem 2.10.** *Let $G$ and $G'$ be free abelian groups of ranks $n$ and $m$ respectively, and let $f : G \to G'$ be a homomorphism. Then there exist bases for $G$ and $G'$ such that, relative to these bases, the matrix of $f$ has the form:*

$$B = \left[\begin{array}{cccc|c} b_1 & & & 0 & \\ & b_2 & & & 0 \\ & & \ddots & & \\ 0 & & & b_l & \\ \hline & & & & \\ & 0 & & & 0 \\ & & & & \end{array}\right]$$

*where $b_i \geq 1$ and $b_1 \mid b_2 \mid \cdots \mid b_l$.*
*The matrix $B$ is said to be in* ***Smith-Normal Form***

The standard reduction algorithm that we will outline below is actually a constructive proof of Theorem 2.10 (see [48] for more information). We first examine the Standard Reduction Algorithm which will later be used to compute the homology. Without loss of generality, we can consider this method using integer coefficients though in our case any PID will work.

**Reduction Algorithm**
The reduction algorithm transforms an $m \times n$ matrix $A = (a_{ij}) = \lambda_{ij}$ into its normal form, the matrix $B$ described in Theorem 2.10, by a series of elementary row and column operations defined below. These preserve a mapping from the bases which make up $A$ to the bases which make up $B$.

Elementary Row Operations:
(r1) Exchanging row $i$ with row $k$
(r2) Multiplying row $i$ by -1
(r3) Exchanging row $i$ by row $i + q * \text{row } k$

*Note: (r2) is analogous to the multiplication of row i by any nonzero real number in Linear Algebra. In our case since we are working in an arbitrary PID and thus only $\pm 1$ are guaranteed to have multiplicative inverses as opposed to every nonzero real number in $\mathbb{R}$.*

Each of these operations corresponds to a change of bases in $G'$ as defined below:

(r1) Swap $a'_i$ and $a'_k$
(r2) Replace $a'_i$ with $-a'_i$
(r3) Replace $a'_k$ with $a'_k - q * a'_i$

Similarly there are three elementary column operations:

(c1) Exchanging column $i$ with row $k$
(c2) Multiplying column $i$ by -1
(c3) Exchanging column $i$ by column $i + q *$ column $k$

Each of these operations corresponds to a change of bases in $G$ as defined below:
(c1) Swap $a_i$ and $a_k$
(c2) Replace $a_i$ with $-a_i$
(c3) Replace $a_i$ with $a_i + q * a_k$

Using these operations the reduction algorithm reduces the matrix $A$ to the form of matrix $B$

**Definition 2.38.** *Let $\alpha(A)$ denote the smallest non-zero magnitude in A, i.e., $\alpha(A) = \min\limits_{i,j}\{|a_{ij}| > 0\}$. $a_{ij}$ is called the minimal entry of $A$ iff $\alpha(A) = |a_{ij}|$*

**Step 1.** Minimize the value of $\alpha(A)$

**Lemma 2.11.** *If $\alpha(A)$ fails to divide some entry of $A$, then it is possible to decrease the value of $\alpha(A)$ by applying elementary operations to $A$; and conversely.*

(See [48] for proof)

The minimization algorithm proceeds as follows:

```
1. Find $a_{ij} = \alpha(A)$
2. Check each entry of column $j$ of $A$ to see if $\alpha(A) \mid a_{kj}$
3. If 2 fails goto 4 else goto 7
4. Find $q \in \mathbb{Z}$ such that $|a_{kj} - q * a_{ij}| < a_{ij}$
5. Replace row $k$ with row $k$ - q*row $i$
6. Goto 1
7. Check each entry of row $i$ of $A$ to see if $\alpha(A) \mid a_{ik}$
8. If 7 fails goto 9 else goto 12
9. Find $q \in \mathbb{Z}$ such that $|a_{ik} + q * a_{ij}| < a_{ij}$
10. Replace column $k$ with column $k$ + q*column $i$
11. Goto 1
12. Check each entry $a_{kl},\ k \neq i, l \neq j$ of $A$ to see if $\alpha(A) \mid a_{kl}$
13. If 12 fails goto 14 else goto 16
14. Replace row $i$ by row $i$ + row $k$
15. Goto 5
16. End
```

**Step 2.** Bring the minimal entry to the upper left corner of the matrix.
1. Using elementary operations bring $a_{ij} = \alpha(A)$ to $a_{11}$.
2. Make $a_{11}$ positive.
3. Using elementary operations make all other entries in row 1 and column 1 zero. Since $a_{11}$ divides all entries in the matrix this is clearly possible.

*It is important to note here that $a_{11}$ still divides all entries in the matrix $A$*

**Step 3.** Reduce the remaining rows and columns
1. Repeat steps 1 and 2 on the submatrix of $A$ ignoring the first row and column.
2. Terminate when the submatrix is empty or when it is the zero matrix

Since each element $a_{kk}$ divides all elements $a_{ij}, i, j > k$ and since all elements $a_{*k}$ and $a_{k*}$ are zero (except for $a_{kk}$) then the matrix is in Smith-Normal Form.

**Application of the Reduction Algorithm**
We may now consider the boundary operator $\partial_k : \mathcal{C}_k \to \mathcal{C}_{k-1}$. Since the chain groups $\mathcal{C}_k$ are free and the boundary operator is a homomorphism, we can represent $\partial_k$ as an integer matrix $M_k$ relative to the standard bases of the chain groups. By Theorem 2.10 (utilizing the algorithm listed above) $M_k$ can be transformed into Smith-Normal Form, we will call this matrix $\tilde{M_k}$ .

Let $l_k = rank\{\tilde{M_k}\} = rank\{M_k\} =$ number of non-zero diagonal entries in $\tilde{M_k}$. Of course, the algorithm also computes bases $\{e_j\}$ and $\{\hat{e}_i\}$ for $C_k$ and $C_{k-1}$ respectively. By computing this for all dimensions $k$ we get the following:

1. The torsion coefficients of $H_{k-1}$ are the elements $b_i > 1$.
2. $\{e_i | l_k + 1 \leq i \leq m_k\}$ is a basis for $Z_k$ and thus $rank\{Z_k\} = m_k - l_k$ where $m_k$ is the number of columns in $M_k$
3. $\{b_i \hat{e}_i | 1 \leq i \leq l_k\}$ is a basis for $B_{k-1}$ and so $rank\{B_k\} = rank\{M_{k+1}\} = l_{k+1}$

Finally we are able to calculate the Betti numbers by combining 2 and 3 and thus $\beta_k = rank\{Z_k\} - rank\{B_k\} = m_k - l_k - l_{k+1}$

Similarly, we can compute the homology over graded PIDs using the standard reduction algorithm and the structure in Corollary 2.1.1. We begin by representing $\partial_k$ using the standard basis for $C_k$ (which is in fact homogeneous) and a homogeneous basis for $Z_{k-1}$. The algorithm then proceeds as before resulting in a matrix $\tilde{M}_k$. From the reduced matrix we can build a structure of the form presented in Corollary 2.1.1 where the following two rules apply:

i. zero row $i$ contributes a free term with shift $\alpha_i = deg\{\tilde{e}_j\}$

ii. row with diagonal term $b_i$ contributes a torsional term with homogeneous $d_j = b_j$ and shift $\gamma_j = deg\{\tilde{e}_j\}$

### 2.3.5 Computing Persistent Homology

In this section we present the modified standard reduction algorithm developed in [65] which can be used to compute persistent homology over a field. This work builds off the pairing algorithm developed by Edelsbrunner Letscher, and Zomorodian [9].

**Computing Persistent Homology Over a Field** Just as with the previous section we consider the boundary operator $\partial_k$ except in the previous case $\partial_k : C_k \to C_{k-1}$ as opposed to the case when considering persistent homologies $\partial_k : C_k^* \to C_{k-1}^*$. For simplicity, we assume that we are computing the persistent homology over $\mathbb{Z}_2$ although the following algorithm holds for any field. As referenced above we begin by building the matrix $M_k$ using the standard basis for $C_k$ and a homogeneous basis for $Z_{k-1}$. Since we are now using the graded module $\mathbb{Z}_2[t]$ produced by the correspondence $\alpha$ defined in Section 2.3.3, the elements $\hat{e}_i$ which make up the homogeneous basis for $Z_{k-1}$ may not have the same degree as the elements $e_j$. We know $\partial(e_j) = \sum\limits_i a_i \cdot \hat{e}_i$ where $a_i \in \mathbb{Z}_2$, but if $\hat{e}_i$ has degree $n$ and $e_j$ as degree $m$ (by necessity $n \leq m$) then we must shift $\hat{e}_i$ by $m - n$ levels in the filtration and so we use $t^{m-n} \cdot \hat{e}_i$. Therefore, in our matrix we multiply the element in each position $M_k(i, j)$ by the difference in degree between the homogeneous elements $e_j$ and $\hat{e}_i$. We then proceed as in the previous section by reducing the matrix and reading off the description for the homology group.

One may notice at this point that in order to determine the matrix $M_k$ a homogeneous basis is needed for $Z_{k-1}$. This of course is not as easy to find as it is for the group $C_{k-1}$ where we can just use the standard basis. However, if one is trying to classify the entire homology then this can be computed using induction. We start with $k = 1$ (the base case $k = 0$ is trivial), where $Z_0 = C_0$. Using this, the matrix representation of $\partial_1$ may be found and used as above. For the inductive step we assume we have some matrix $M_k$ representative of $\partial_k$ relative to the standard basis $\{e_j\}$ of $C_k$ and a homogeneous basis $\{\hat{e}_i\}$ for $Z_{k-1}$.We must now find a homogeneous basis $Z_k$.

**Definition 2.39.** ***Column Echelon Form*** - *A matrix is said to be in column echelon form iff its transpose is in row echelon form.*

Recall that the composition of consecutive boundary operators is 0, i.e., $\partial_k \partial_{k+1} = 0$. As such, since $M_k$ is the representation of $\partial_k$ we know that $M_k M_{k+1} = 0$. Of course, this is also unchanged by elementary operations on the matrices. Additionally, since the domain of $\partial_k$ is the codomain of $\partial_{k+1}$ column operations on $\tilde{M}_k$ are row operations on $M_{k+1}$. Thus, the column operations used to transform $M_k$ into echelon form zero out rows in $M_{k+1}$ corresponding to nonzero pivot columns in $\tilde{M}_k$ and represent $\partial_{k+1}$ relative to the homogeneous basis for $Z_k$. And so, we arrive at the following lemma [65]:

**Lemma 2.12.** *To represent $\partial_{k+1}$ relative to the standard basis for $C_{k+1}$ and the homogeneous basis computed for $Z_k$ it is only necessary to delete the rows in $M_{k+1}$ that correspond to nonzero pivot columns in $\tilde{M}_k$.*

This gives us the basis (no pun intended) for the algorithm developed in [9] which proceeds as follows:

1. Sort the rows of $M_k$ corresponding to the $\hat{e}_i$ basis elements for $Z_{k-1}$ in reverse degree order.
2. Transform $M_k$ into column echelon form
3. The non-pivot columns of the transformed matrix then form the basis for $Z_k$
4. Write the matrix for $M_{k+1}$ using the basis elements for $C_k$ and $C_{k-1}$ as the column and row bases.
5. Remove the rows corresponding to non-zero pivot columns in $\tilde{M}_k$
6. This matrix is now the representation of $\partial_{k+1}$ that we are after.

It also turns out that having the column echelon form of the matrix representation $M_k$ of $\partial_k$ is enough to determine the $\mathcal{P}$-intervals for $H_{k-1}$

**Theorem 2.13.** *Let $\tilde{M}_k$ be the column-echelon form for $\partial_k$ relative to bases $\{e_j\}$ and $\{\hat{e}_i\}$ for $C_k$ and $Z_{k-1}$, respectively. If row $i$ has pivot $M_k(i,j) = t^n$, it contributes $\sum^{deg\{\hat{e}_i\}} F[t]/t^n$ to the description of $H_{k-1}$. Otherwise, it contributes $\sum^{deg\{\hat{e}_i\}} F[t]$. Equivalently, we get $(deg\{\hat{e}_i\}, deg\{\hat{e}_i\}+n)$ and $(deg\{\hat{e}_i\}, \infty)$, respectively, as $\mathcal{P}$-intervals for $H_{k-1}$.*

Proof can be found in [65].

**Computing Persistent Homology Over a PID** Recall that the Structure Theorem gives a classification for the structure of graded modules over graded PIDs. If our persistent groups are taken over $R$ where $R$ is a field, we can apply the results of the Structure Theorem since $R[t]$ is a PID when $R$ is a field. However, if $R$ is not a field, then we cannot necessarily use the Structure Theorem to provide a simple classification of the graded module. As such we have no simple classification for persistent homologies over PIDs like we do over fields. Despite this, there is an algorithm which produces a description of the groups $H_k^{i,p}$ for fixed $i$ and $p$ that was developed in [9].

As previously discussed, $H_k^{i,p} = Z_k^i/(B_k^{i+p} \cap Z_k^i)$. Without the need to move through the grading (since $i$ and $p$ are fixed) we can still utilize the previous method of building the matrix $M_k^i$ as the representation of $\partial_k^i$ and using the standard reduction method to produce a characterization of $Z_k^i$.

For the denominator then we have $B_k^{i,p} = B_k^{i+p} \cap Z_k^i$ and so instead of considering the matrix $M_{k+1}^i$ we need to build a new matrix $M_k^{i,p}$ which represents the boundary group of $p$-persistent $k$-chains beginning in $K^i$. This matrix is then constructed as follows:

1. Begin by reducing matrix $M_k^i$ to its normal form and thus construct the basis $\{z^j\}$ for $Z_k^i$.
2. Reduce the matrix $M_{k+1}^{i+p}$ to its normal form and construct a basis $\{b^l\}$ for $B_k^{i+p}$.
3. Now construct a matrix $N = [\{b^l\}\{z^j\}] = [B\ Z]$ where the columns of the matrix $N$ are the basis elements of the bases computed in steps 1 and 2 and the matrices $B$ and $Z$ are the sub-matrices defined by the bases respectively. Reduce the matrix $N$ to Smith Normal Form which reveals a basis $\{u^m\}$ using the method described above Of course $\{u^m\} = [\alpha^m \beta^m]$ where $\alpha^m$ and $\beta^m$ are vectors of coefficients from the bases $\{b^l\}$ and $\{z^j\}$ respectively. Both $\{B\alpha^m\}$ and $\{Z\beta^m\}$ are bases for $B_k^{i,p}$ so either can now be used to build the matrix $M_{k+1}^{i,p}$

Then by reducing the matrix $M_{k+1}^{i,p}$ we constructed we now have a description for $H_k^{i,p}$. Using Theorem 2.13 and Lemma 2.9 we can produce a description for the persistence homology via its $\mathcal{P}$-intervals and its Betti numbers.

Some examples of computing persistent homologies can be found in Appendix A.

### 2.3.6 Visualization

**Persistence Diagram** We can immediately see that the $\mathcal{P}$-intervals can be used to describe the equivalence class of cycles created in $K^i$ and dying in $K^j$. The use of these $\mathcal{P}$-intervals allows us to visualize the persistence pairs across the levels of filtration within the filtered complex. First by examining the homologous cycles in a two-dimensional space consisting of the filtration index by the persistence, we see that a homologous cycle exists non-bounding in $K^i$ and remains until $K^j$. So, in our two-dimensional space, a homologous $p$-cycle exists at the point $(i, 0)$ and exists at $(l, 0)$ for $l < j$ and at $(i, l)$ for $l < j - 1$. Thus, it exists in the triangular convex hull of these points with open boundary the line $l + p = j$.

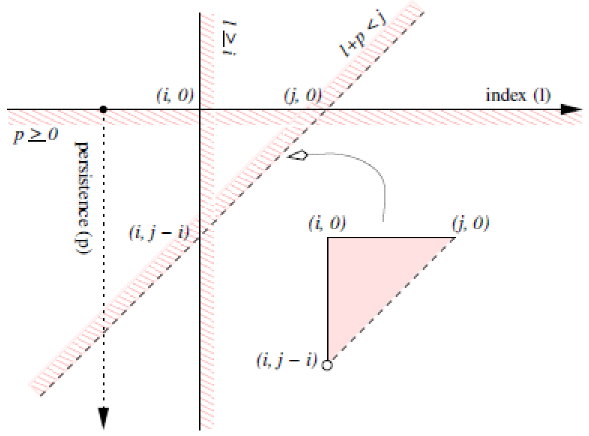


Figure 2.10: *Triangular Region containing homologous cycles defined by the inequalities $p \geq 0, l \geq i, l + p < j$ [65]*

We have seen that the Persistent Homology over a field can be decomposed into simplex pairs $(\sigma_i, \sigma_j)$ - where $\sigma_i$ is created in $K^i$ and $\sigma_j \sim \sigma_i$ dies in $K^j$ - and those non-bounding cycles $\tau \in K$. In this case each pair $(\sigma_i, \sigma_j)$ which gives us the pair $(i, j)$ produces a triangular region like the one depicted in Figure 2.10. A Persistence Diagram takes all of the triangles and simplex pairs and combines them together into a single diagram like the one depicted in Figure 2.11. On the bottom half of the diagram, we have the triangles associated with each pair and along the top we have line segments equal to the persistence of $\sigma_i$. For those non-bounding cycles in $K$ the line continues until the edge of the diagram.

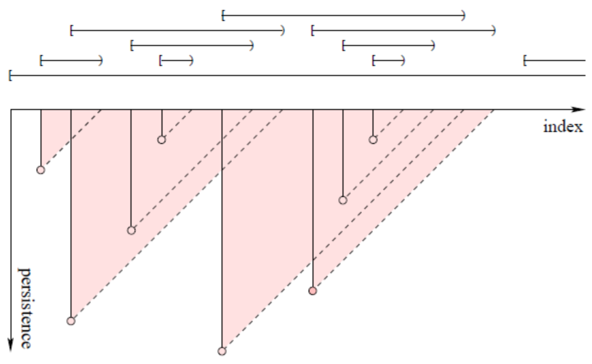


Figure 2.11: Persistence Diagram [65]

**Bar Code** We can also use a visualization scheme known as the Bar Code [10]. The Bar Code functions similarly to the line segments along the top of the persistence diagram, except in the case of the Bar Code diagram we also display the dimension $p$ of the $p$-cycles. Along the vertical axis we display the dimension and along the x-axis, the levels of the filtration. We denote non-bounding cycles in $K$ using an arrow rather than a line segment.

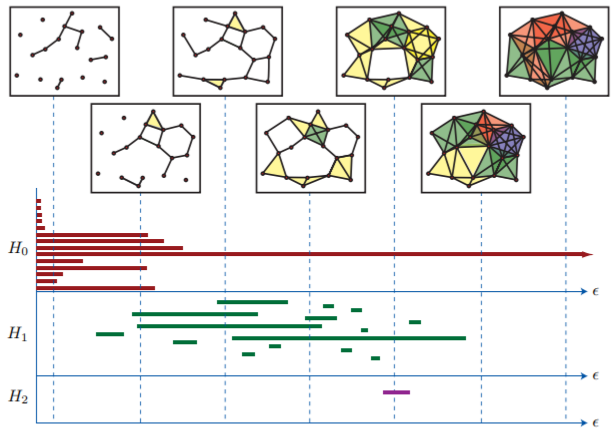


Figure 2.12: *Example of a Bar Code diagram [19]*

## 3 Convolutional Neural Networks

The last piece of background material needed is a discussion of Convolutional Neural Networks, these will be the application of the TDA theory being defined. One of the most recognized and used machine learning techniques is the Artificial Neural Network. Artificial Neural Networks, typically shortened to Neural Networks or ANNs, are a machine learning method that seeks to replicate the organization of neurons in the human brain to achieve pattern recognition. A Neural Network consists of a set of nodes known as the input layer, one or more hidden layers, and an output layer. Each node is connected to one or more nodes in the following layer and has an associated weight and activation function.

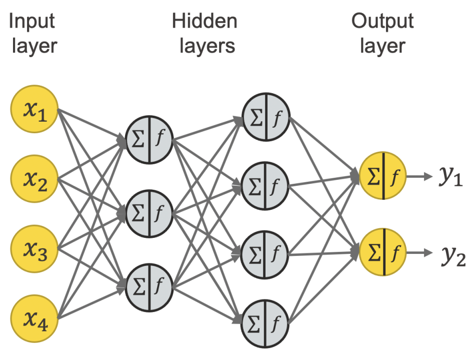


Figure 3.1: *Diagram of a Neural Network with 2 hidden layers (source: [45])*

Each node, after the input layer, sums the output of an activation function applied to the dot product of the output values of the nodes in the prior layer with the weights associated with each connection. Typically, each node in the output layer is associated with a class and outputs the probability of the input belonging to its associated class (in the case of classification) or it outputs the estimated value (in the case of regression).

Usually, a Neural Network is created by defining a network architecture - number of layers and nodes in each layer as well as the activation functions - and then a training data set is used to tune the weights of the network. The data is fed through the network and then the output is compared to a ”truth” value which is used to tweak the weights through a method known as backpropagation. This process is repeated a number of times either with all of the data or some subset of the training data with each round of training with a particular set being referred to as an epoch. After a predetermined number of epochs, or a desired threshold of accuracy achieved, the network is ready to be used. The data in question can be input into the first layer of the network and the network will output the most likely class associated with that input (or value in the case of regression).

The concept of Neural Networks as an algorithm or mathematical model is typically attributed to Warren McCullough and Walter Pitts [44]. After their advent, they were studied extensively until 1969 when Marvin Minsky and Seymour Papert [46] showed that computing power was not sufficient to handle any Neural Network sophisticated enough to produce good results. Neural Networks have waxed and waned since. In the most recent decade, they have had a resurgence under their use in deep learning which uses neural networks with many hidden layers.

Convolutional Neural Networks, as referenced in Section 1, are a subclass of Artificial Neural Networks. While an overview was given in Section 1.1 it is necessary here to delve a bit further into the theory behind CNNs.

One of the issues with Neural Networks, especially when it comes to image processing, is that all of the elements of an input vector are considered together, assuming they are connected to the same node in the next layer. In a fully connected neural network, meaning that every node is connected to every node in the subsequent layer, all input elements are related. In ANNs which are not fully connected the architect of the network has to carefully choose which values should be considered together. While this is desired in a network that is handling basic vectors of parameterized data, such as a network considering health data to predict cancer, in image processing and many other areas this is a problem. For example, when performing image processing it is desirable to only consider pixels which are near to each other, this is known as the receptive field and is how the neurons in the human visual cortex function. Convolutional Neural Networks offer a way of doing this by making each node correspond to an $m \times n$ filter which is "scanned" across each position in the image. In this way, each node corresponds to a field around a pixel instead of an individual pixel. CNNs have three main types of layers.

## 3.1 Convolutional Layers

First and most importantly, is the convolutional layer. The convolutional layer is where the majority of the computation is done in the CNN architecture. The idea of the convolutional layer is that a filter is applied to local patches (i.e., receptive field) in the layer to attempt to detect features in local areas of the data. This is performed by convolving an $m \times n$ matrix of filter weights with the values in the layer. So, given a matrix of data $A$ the $m \times n$ filter $F$ is multiplied element-wise by values in the upper left corner, $A_{1..m,1..n}$ and the values are then summed, so $(F * A)_{1,1} = \sum_{i=1}^{m} \sum_{j=1}^{n} F_{i,j} A_{i,j}$. The filter is then moved $s$ number of positions to the right and the process is repeated. Once the filter has made it all the way across the matrix it returns to the left and is stepped down $s$ number of positions ($s$ is known as the stride and is discussed in more detail below). See figures 3.2, 3.3 for examples of the convolutional operation.

For those familiar with signal processing the term convolution should be familiar. However, while the convolution operation in CNNs is similar, it is not quite the same. The CNN convolution operation is more akin to the correlation operation in signal processing whereas the convolution operation in signal processing represents a flip of the filter in both vertical axis and the horizontal axis (see Figure 3.4).

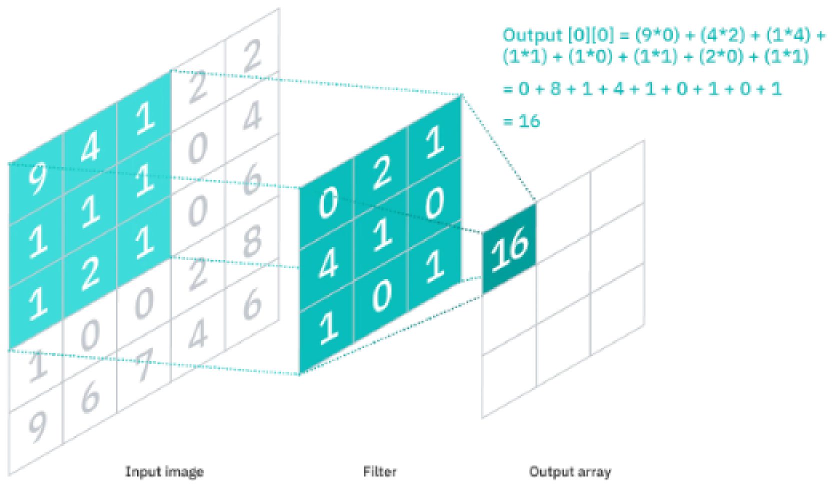

Output [0][0] = (9*0) + (4*2) + (1*4) + (1*1) + (1*0) + (1*1) + (2*0) + (1*1)
= 0 + 8 + 1 + 4 + 1 + 0 + 1 + 0 + 1
= 16
Input image
Filter
Output array

Figure 3.2: *Graphical representation of a 3 × 3 filter applied to a set of values in a convolutional layer (source: [30])*

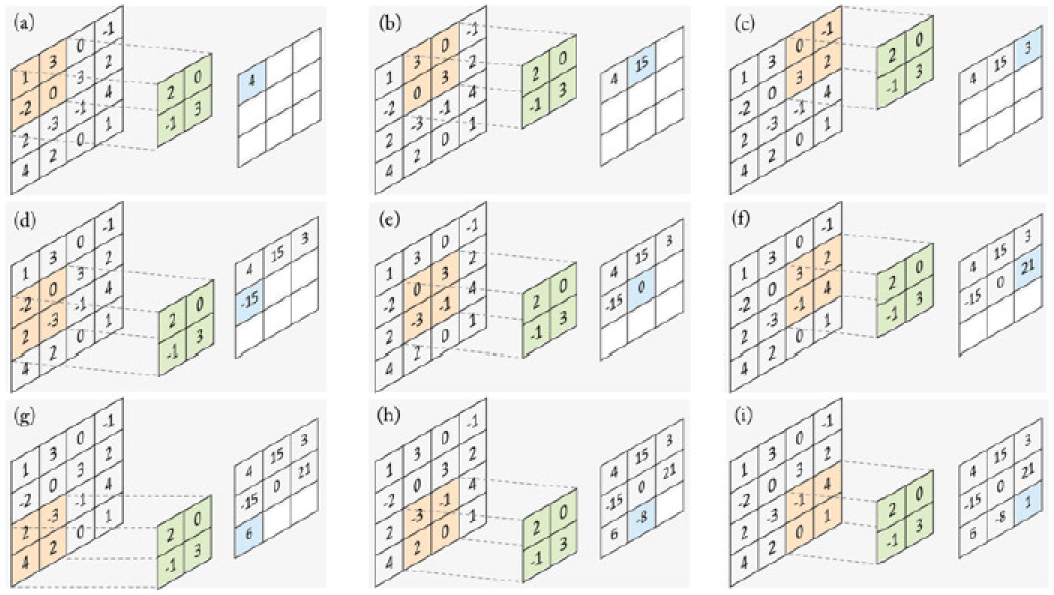


Figure 3.3: *Graphical representation of a 2 × 2 filter (shown in green) applied to a set of 4 × 4 values in a convolutional layer. The receptive field (i.e., the values being multiplied by the filter) are shown in orange while the output is shown in blue. (source: [36])*

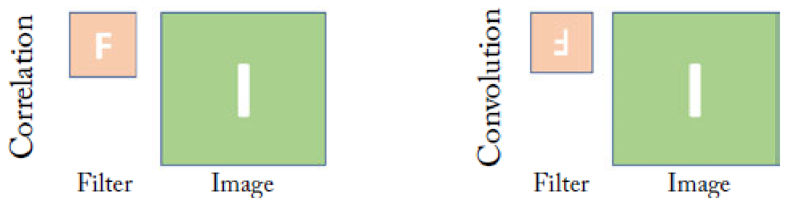


Figure 3.4: *Graphical comparison between the correlation operation and convolution operation in signal processing. The CNN convolution is the same as the correlation operation (source: [36])*

From a neural network perspective, in the convolutional layer, each node is connected to an $m \times n$ set of local points, meaning that the points exist in an $m \times n$ rectangle in the image, data matrix, or CNN layer. So, if the receptive field is 3x3 then the upper right node in the convolutional layer is connected to the upper rightmost 3x3 grid. The next node to the right of this is then connected to a 3x3 patch to the right of the upper rightmost 3x3 grid, the distance between which is referred to as the stride. In this way the filter is "slid", across the input layer.

To compute the output in the subsequent layer node, the matrix of filter weights is multiplied by the matrix of nodes/values it is connected to in the previous layer. These values are then summed and passed through an activation function. The output of this function is then pass to the next node.

The dimension of the following layer depends on the size of the filter, the stride, and the padding applied to the data. As referenced above, the stride is the number of positions the filter is "stepped" between each set of multiplications. So for a given stride $s$ we get $B_{1,1} = (F * A)_{1,1} = \sum_{i=1}^{m} \sum_{j=1}^{n} F_{i,j} A_{i,j}$ and $(F * A)_{1,2} = \sum_{i=1}^{m} \sum_{j=s}^{n+s} F_{i,j} A_{i,j}$. These values along with the height $h$ and width $w$ of the input matrix $A$ determine the size $h' \times w'$ of the output matrix $B$. Where:

$$h' = \lfloor \tfrac{h-m+s}{s} \rfloor \text{ and } w' = \lfloor \tfrac{w-n+s}{s} \rfloor$$

Given this, the dimension of the output of each convolutional layer must always be less than the input. However in some applications this is not a desirable quality. As such, we can introduce the concept of zero padding. Zero padding inserts zeros around the edges of the input data matrix increasing the size of $A$ from $h \times w$ to $h + 2p \times w + 2p$. The padding can be one of three types:

- **Valid Padding:** In this case no zeros are padded around the image/layer so the dimension will decrease
- **Same Padding:** This pads the edges with zeros so that the output dimension is the same as the input dimension
- **Full Padding:** This can be any padding that increases the dimension of the output.

This then causes the size $h' \times w'$ of the output matrix $B$, to be:

$$h' = \lfloor \frac{h-m+s+p}{s} \rfloor \text{ and } w' = \lfloor \frac{w-n+s+p}{s} \rfloor$$

In general:

$$(F * A)_{l,k} = \sum_{i=s(l-1)+1}^{m+s(l-1)} \sum_{j=s(k-1)+1}^{n+s(k-1)} F_{i,j} A_{i,j}$$

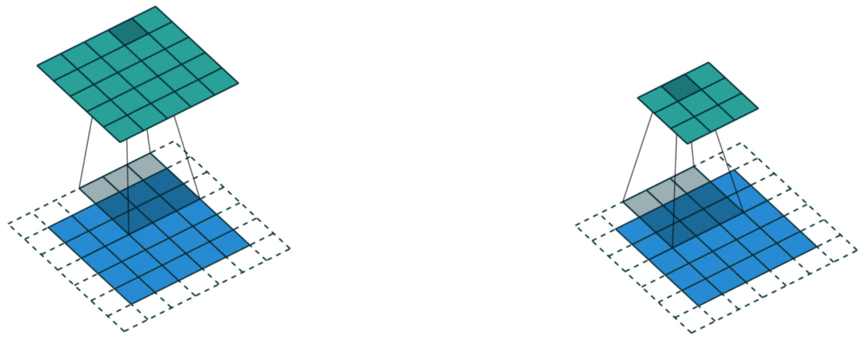


Figure 3.5: *Left: Convolutional Layer with a 3x3 filter, padding, and a stride of 1, Right: Convolutional Layer with a 3x3 filter, padding, and a stride of 2 (source: [55])*

Unlike traditional ANNs, a convolutional layer has two dimensions (height and width) which are determined by the size of the input layer, the filter size, the stride, and the zero padding. Additionally, most convolutional layers also have a 3rd dimension known as the number of channels. If the input image is a color image, then each pixel actually contains three values known as RGB. (Digital images use an encoding called RGB which gives values for the hue of red, green, and blue, respectively contained in the pixel, using red, green, and blue, any color can be represented). A convolutional layer then should include at least one channel for each color value, typically 3. This being said, there is no theoretical reason why a convolutional layer can only have one channel for each color value. Convolutional layers can actually have multiple channels for each color value. This allows the layer to tune the weights in different ways to represent different lenses with which to look at an image.

Lastly, the convolutional layer must also contain an activation function. Once the matrix of filter weights is applied to the matrix of input values, the output is fed through an activation function which determines the value to be fed through to the next node. These are typically the same sort of functions used in ANNs and include ReLU, threshold, softmax, logistic, and others. A detailed discussion of these can be found below in Section 3.4.

With all of this then the architecture of a convolutional layer contains the following:

- Filter size
- Stride
- Zero Padding
- Number of channels
- Activation Function

## 3.2 Pooling Layer

Then there is the pooling layer. Pooling layers perform the function of dimensionality reduction, sometimes referred to as downsampling, which is very similar to the concept of the same name in Digital Signal Processing. The pooling layer functions very similar to the convolutional layer, by using a filter that is slid across the previous layer, but instead of using weights to try to detect features it uses an aggregation function across the values to which the filter is applied. Typically, Pooling Layers use one of two functions, Max Pooling - in which case the output is the largest value of the nodes to which the filter is applied, or average pooling - in which an average is taken across the values to which the filter is applied. The idea in either case is that a receptive field can be reduced down to a single node, or smaller set of nodes, representing the feature or features of that specific field.

While both max pooling and average pooling induce information loss, they preserve the most important information. In this way, both - but especially max pooling - act as a sort of noise suppressant or high pass filter which preserves higher outputs from the activation functions of the previous nodes and suppresses or throws out noisy activation values. In general, because of its ability to suppress noisy features, max pooling is much more effective and therefore more popular in CNN architectures.

A Pooling layer is defined by its function (max or average pooling), its filter size, the stride, and the amount of padding. These uniquely determine the shape of the layer as the size of its output can be computed from the size of the previous layer and the filter and padding.

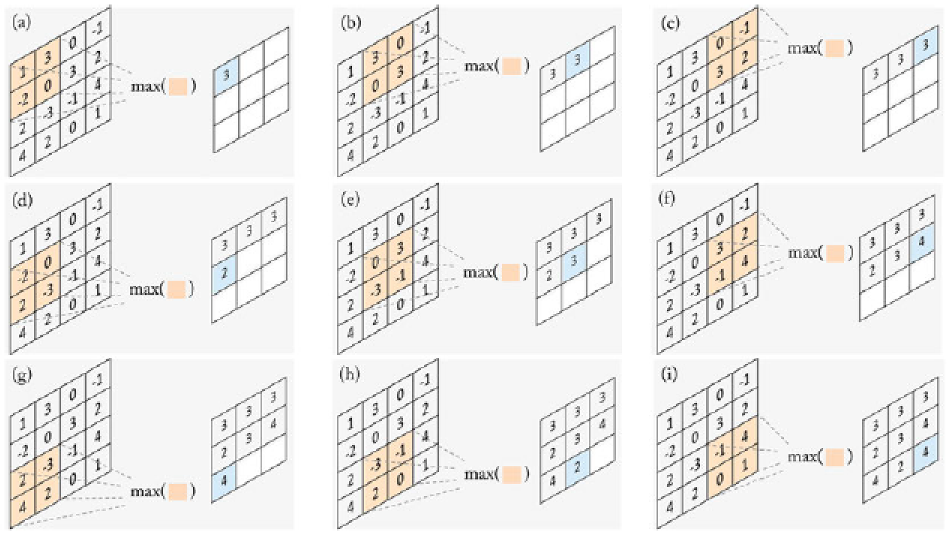


Figure 3.6: *Example of a* 2×2 *max pooling filter on a* 4×4 *input data with stride 1. The receptive field is highlighted in orange while the output is highlighted in blue (source: [36])*

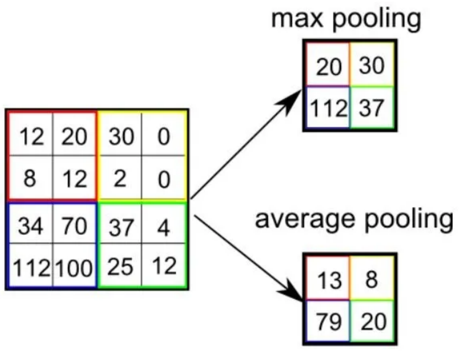


Figure 3.7: *Example of the results of* $2 \times 2$ *max and average pooling filters with stride 2 on a* $4 \times 4$ *input (source: [55])*

Similar to the convolutional layer the output size is determined by the filter size, the stride, and the zero-padding. As a result, the size $h' \times w'$ of the output matrix $B$, are:

$$h' = \lfloor \frac{h-m+s+p}{s} \rfloor \text{ and } w' = \lfloor \frac{w-n+s+p}{s} \rfloor$$

## 3.3 Fully-Connected Layer

The Fully-Connected layer is basically the same as the layers in a typical ANN. It is used to "flatten" the CNN as every node in the previous layer is mapped to every node in the fully connected layer (hence the name) and thus instead of having a 3-D $h \times w \times d$ representation there are just $h \times w \times d$ nodes in the fully connected layer giving it a flat representation.

Thus, if $N_i$ is the i-th node with $W_i$ its associated weight values, and $A_{i,j,k}$ is the $i$-th row, $j$-th column and $k$-th channel of the previous layer, then $N_i = g(\sum_{i=1}^{h} \sum_{j=1}^{w} \sum_{k=1}^{d} A_{i,j,k} W_{i+(j-1)*(h)+(k-1)*(h*w)})$ where $g$ is the activation function.

Each node then represents a feature (or set of features) detected in the original input layer. These features are then mapped, using a standard ANN weight and activation function, typically softmax, to a set of output nodes representing the possible classes. This architecture can be seen in Figure 3.8.

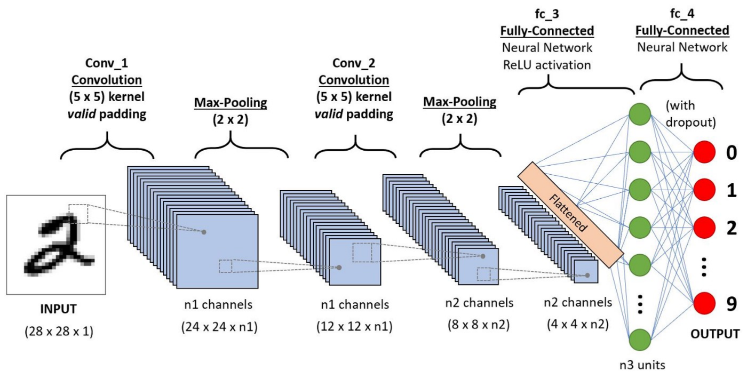


Figure 3.8: *Diagram of a Convolutional Neural Network (source: [55])*

## 3.4 Activation Functions

So far, we have mentioned two specific activation functions which are used in CNNs, ReLU and softmax. While there are many different activation functions ReLU and softmax are the most common in CNNs. An activation function in its most basic form is just a map from $\mathbb{R}^n$ into $X \subseteq \mathbb{R}$. Typically $X = \mathbb{R}^+$ or in some cases $X = \{0, 1\}$. There is nothing precluding $X$ from being negative; however, it is not desirable to have a negative activation function and therefore it is almost always non-negative. The output of the activation function is used to determine whether or not to activate a neuron $f(v) > 0$ or not $f(v) = 0$. In linear classification or other statistical learning models this function is used to determine a class to which an input belongs by using it to describe a probability and then using a threshold to determine membership. So explicitly $p : \mathbb{R}^n \to [0, 1]$ and $f : \mathbb{R}^n \to \{0, 1\}$ where $f(v) = 1$ if $p(v) \geq \rho \in (0, 1)$ and $p(v) = 0$ if $p(v) < \rho$.

While technically any function which maps a vector from $\mathbb{R}^n$ into $\mathbb{R}$ is eligible to be used as an activation function, there are several properties which are assumed in order to give the function the behavior necessary for it to provide value to the learning method to which it is applied.

First, the function should be monotonic, i.e., for $x, y \in \mathbb{R}^n, x > y \Rightarrow f(x) \geq f(y)$ (increasing) or for $x, y \in \mathbb{R}^n, x > y \Rightarrow f(x) \leq f(y)$ (decreasing). The reason for this is that in training if there is point that is not monotonic than the training can get "stuck" oscillating between values above and below this point and never converge to an actual solution. While this is not guaranteed (nor does monotonicity imply that training will always converge), the absence of monotonicity greatly increases the risk of nonconvergence.

Secondly, the function ought to be differentiable. This allows methods like gradient descent and back propagation to be used to find optimal, or converge towards optimal, solutions. This also implies that the function must be continuous which is a useful property as it provides some "smoothness" to the training preventing rapid jumps in output (this is why modern ANNs tend to prefer $\mathbb{R}^+$ to $\{0, 1\}$).

Additionally, it is general practice to assume that the function is increasing. While this is not necessary it lends itself to the theory behind neural nets from neuroscience which uses activation functions to model synapses which only fire after a threshold of energy is surpassed.

Activation functions tend to be the integral of bell-shaped functions. This is because the most important data is centered around the threshold or separating line and so these values must be pulled rapidly away from this line in order for the method to converge to a solution. That being said this does not hold for ReLU which is the most commonly used activation function for CNNs. A discussion of why ReLU is favored can be found below.

The most commonly used activation functions then are:
ReLU
$R(x) = \max\{0, x\}$

Softmax
$\sigma(x)_i = \dfrac{e^{x_i}}{\sum_{j=1}^{k} e^{x_j}}$ for $i \in 1..k, x \in \mathbb{R}^k$

Standard Logistic Function:
$f(x) = \frac{1}{1+e^{-x}}$

Additional activation functions include:
Hyperbolic Tangent Function:
$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

Gudermanian Function:
$gd(x) = \int_0^x \text{sech}(z)dz = \arcsin(\tanh(x))$

Error Function:
$erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-z^2} dz$

**ReLU**
The ReLU or Rectified Linear Unit is the following function:

$$R(x) = \max\{0, x\}$$

The ReLU then activates the node if the value is positive and otherwise is 0 and therefore inactivate. This means that features will never negatively contribute to the output of the network.

Unlike ANNs the ReLU is used almost exclusively in all but the final layer of a CNN. This is for three main reasons. First off, ReLUs themselves are computationally inexpensive as they are only a simple comparison instead of a set of multiplies and divides as is the case with logistic functions such as the softmax function. Secondly, the derivative of the ReLU function is either 0 or 1 which makes computation for the backpropagation algorithm substantially easier and cheaper. These two reasons are of great importance as CNNs are very computationally intensive and so any place where we can achieve good results but reduce computations it is necessary to do so. In the case of the CNN, as the number of nodes increase the weights increase exponentially, but the activation functions increase linearly. Lastly, the ReLU does not contribute negatively to the network, as mentioned above. This is advantageous as in a CNN a neuron being not relevant does not mean that other local neurons are less relevant as neurons represent different lenses applied to local receptive fields and often times important features can be found locally near irrelevant features or noise.

**Softmax**
The other function commonly used in CNNs is the softmax function. The softmax function is defined as:

$$\sigma(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{k} e^{x_j}} \text{ for } i \in i..k, x \in \mathbb{R}^k$$

In the case that there are only two classes (binary classification) the softmax function is the same as the Standard Logistic function:

$$S(x) = \frac{e^x}{1 + e^x} = \frac{1}{1 + e^{-x}}$$

The softmax function is beneficial because it maps each output into [0,1] in such a way that the sum of the $\sigma_i$'s sum to 1, allowing it to be interpreted as a probability. This gives it an advantage over the ReLU function used throughout most of a typical CNN as a ReLU's output gives only a 0 or a positive real, so while this could be used in the final layer, the probability interpretation of the softmax gives smoother training and provides a measure of error or uncertainty both for training and in the final use of the network. Thus, it is often used in the final layer of CNNs and ANNs as it can be used to give a probability of the input belonging to each output class.

Unlike the hidden layers in a CNN, a softmax activation function can be used in the final layer as the dimension of the data has been greatly reduced by the time it reaches the final layer, therefore the concern of computational complexity is removed. For these reasons the softmax (or Standard Logistic in the case of binary classification) is the most common activation function for the output layer of a CNN.

**Hyperbolic Tangent Function**
The hyperbolic tangent function can also be used for Linear Classification or Neural Net Activation.While it maps into [-1,1] and thus does not have a probabilistic interpretation of the output it does still have a logical connection to probability theory. If one assumes that the output is distributed with a Gaussian distribution around their respective means then the hyperbolic tangent function is just the difference between the conditional probabilities.

Let $D(x) = P(0|x) - P(1|x)$ where 0 and 1 are our two classes. Then assuming the distributions for the two classes are Gaussian with means equally spaced from the origin $(\mu, -\mu)$, and standard deviations of $\sigma$, then:
$D(x) = \frac{P(x,0)-P(x,1)}{P(x,0)+P(x,1)}$
$= \frac{e^{\frac{-(x-b)^2}{2h^2}} - e^{\frac{-(x+b)^2}{2h^2}}}{e^{\frac{-(x-b)^2}{2h^2}} + e^{\frac{-(x+b)^2}{2h^2}}}$
$= \frac{e^{\frac{bx}{h^2}} - e^{-\frac{bx}{h^2}}}{e^{\frac{bx}{h^2}} - e^{-\frac{bx}{h^2}}}$
$= \tanh(\frac{bx}{h^2})$

The hyperbolic tangent function can also be shown to be a scaled, stretched, and shifted standard logistic function:
$\frac{1}{2} + \frac{1}{2} * \tanh(x) = \frac{1}{2} + \frac{e^x - e^{-x}}{2(e^x + e^{-x})}$
$= \frac{e^x - e^{-x} + e^x + e^{-x}}{2(e^x + e^{-x})}$
$= \frac{2e^x}{2(e^x + e^{-x})}$
$= \frac{1}{1+e^{-2x}}$
$= f(2x)$ where f(x) is the Standard Logistic function

At first glance this would appear then to be no different than the Standard Logistic function, but hyperbolic tangent actually has some advantages over the Standard Logistic function. First, it has a larger gradient around the threshold value. This allows the training to converge faster as data is pulled away from the threshold more quickly than with the Standard Logistic function. Also, in neural net training, it is advantageous to have the mean of the function centered around 0 so as not to bias the network. So, when using the standard logistic function, which has a mean of 0.5, the network is biased towards activation and so the network will tend towards saturation of the neurons; however, since the hyperbolic tangent function has a mean of 0, it does not have this problem. Though as noted prior, with CNNs this property is not desirable as the output is negative.

The derivative of the hyperbolic tangent function is $\frac{d}{dx}\tanh(x) = \text{sech}^2(x)$. But similar to the standard logistic function this is just a form of the original function.
$1 - \tanh^2(x) = 1 - (\frac{e^x - e^{-x}}{e^x + e^{-x}})^2$
$= \frac{e^{2x}+2+e^{-2x}}{(e^x+e^{-x})^2} - \frac{e^{2x}+2-e^{-2x}}{(e^x+e^{-x})^2}$
$= \frac{4}{(e^x+e^{-x})^2}$
$= \frac{1}{\cosh^2(x)} = \text{sech}^2(x) = \frac{d}{dx}\tanh x$
So, by using memory, it is only necessary to perform a simple multiplication (square) of the original function value and then subtract this value from 1.

**Gudermanian Function**
While less common than the standard logistic function and the hyperbolic tangent the Gudermanian is also a sigmoid function which is used for activation. The Gudermanian Function was designed to be a bridge between the circular trigonometric functions and the hyperbolic trigonometric functions without explicitly using complex numbers. It is most commonly used in the Mercator Projection as a line of constant latitude is displaced from the equator by an amount proportional to the inverse Gudermannian of the latitude.

The Gudermanian has similar properties to the hyperbolic tangent. In fact, around its mean, the two functions are nearly identical and only diverge as they approach their limits. Its range is from $\frac{-\pi}{2}$ to $\frac{\pi}{2}$ with mean at 0 and it has a steeper slope around its mean than the standard logistic function. It however, does not have a simplified (in complexity) version of its derivative.

**Error Function**
The error function, also known as the Gauss Error Function is another sigmoid function which can be used for activation/classification. Given a random variable from a normal distribution with mean 0 and variance 1/2 the error function evaluated at $x$ gives the probability of the random variable falling in the region $[-x, x]$. The error function cannot be evaluated in closed form so it must be approximated by integrating its Maclaurin Series expansion. This

is generally not a problem since most calculations for activation or classification would be performed using a computer package which in general would use a series expansion to evaluate the integral rather than a closed form solution.

The Error Function also behaves similarly to the hyperbolic tangent and Gudermannian functions. Its range is from -1 to 1 just like the hyperbolic tangent. It does have a higher gradient in the neighborhood around 0 than either the Gudermannian or the hyperbolic tangent, but it is in general not enough to make a substantial difference. The derivative of the error function is given by $\frac{d}{dx}erf(x) = \frac{2}{\sqrt{x}}e^{-x^2}$.

## 3.5 Training

The training of an ANN can be interpreted as an optimization problem, where the weights are the unknowns which can be tuned to minimize the error or loss in the output values. By starting with the error in the output layer this can be passed (or propagated) backwards (backpropagation) through each layer calculating an error or loss function at each layer using the loss and optimal weights calculated in the following layer. Using this method to train a neural network, each training sample is input into the network and its output calculated and then using this value the error is calculated, via a loss function. The loss is propagated backwards through the network using an optimization method to tune the weights at each layer to minimize the error. Once this has been done all the way through to the input layer the method of sample input→error calculation→backpropagation is repeated. This process will be repeated either a set number of times, or until a minimum error at the output is reached.

### 3.5.1 Gradient Descent

In almost all neural networks the optimization method used is gradient descent.

**Definition 3.1.** ***Gradient*** *The gradient of a function $f : \mathbb{R}^n \to \mathbb{R}$ is defined as the vector of the partial derivatives of the function with respect to each dimension, i.e.,*

$$\nabla f : \mathbb{R}^n \to \mathbb{R}^n \textit{ where } \nabla f = [\tfrac{\partial f}{\partial x_1} \tfrac{\partial f}{\partial x_2} ... \tfrac{\partial f}{\partial x_n}]$$

Gradient descent is a method for finding a local minimum of a function where a starting point $p$ is selected and then the algorithm moves in steps of size $\gamma$ in the opposite direction of the gradient.

Since the gradient of $f$ at a point $p$ (i.e., $\nabla f(p)$) is the direction of the greatest increase of $f$ at $p$ then moving in the opposite direction will move towards a local minimum. It can be shown that for small enough $\gamma$ if $p_{n+1} = p_n - \gamma * \nabla f(p_n)$, then $f(p_0) \geq f(p_1) \geq f(p_2) \geq \cdots$ is a decreasing function. This means that if we can select $\gamma$ small enough then gradient descent will take us towards a local minimum. Of course, $\gamma$ must be selected large enough that the sequence converges which poses a problem for how to select $\gamma$. Selection and tuning of $\gamma$ (known as the learning rate) is beyond the scope of this paper, but a fairly common value is 0.001.

In order to perform training for a neural network the training data is broken up into two sets, one which is used to train, and one which will be used to validate the training. It is necessary to split the data and use a set which is not trained on to perform validation because after long training times neural networks can overfit the training data, meaning they learn exactly the data being trained with and not the ambient data space from which the samples were taken. This means that when presented with new data that does not exactly fit the training data, the network will perform poorly. When an ANN overfits and learns only the training data and is not able to perform well on data outside the training class, it is said of the ANN that it does not generalize well. For more explanation see Section 3.7.

Using the training data set, the samples are input into the network and an error is calculated at the output layer using the loss function. The backpropagation algorithm then propagates these errors back through the network calculating new weights at each connection. This process is referred to as an epoch.

**Definition 3.2.** ***Epoch*** *An epoch is one iteration of a set of samples through the network, a calculation of errors at the output layer, and then an iteration of the backpropagation algorithm using these errors to tune the weights.*

Often times it is advantageous to use different randomly sampled subsets of the training data for each epoch. This further helps prevent overfitting and also reduces the time it takes to perform an epoch. The size of the random sample is known as the batch size.

One problem with backpropagation is that in large networks it becomes computationally expensive. Since the gradient requires computing a partial derivative for each dimension, or in our case connection. As the connections in the network grow the number of computations needed to compute all the gradients grows with it. As such two alternatives are often used.

**Definition 3.3.** ***Mini-Batch Gradient Descent*** *functions the same as gradient descent except instead of using the full gradient* $[\frac{\partial f}{\partial x_1} \frac{\partial f}{\partial x_2} \cdots \frac{\partial f}{\partial x_n}]$ *it uses some subset of the features so we get* $[\frac{\partial f}{\partial x_{i_1}} \frac{\partial f}{\partial x_{i_2}} \cdots \frac{\partial f}{\partial x_{i_m}}]$ *where* $m \ll n$.

**Definition 3.4.** ***Stochastic Gradient Descent*** *functions the same as gradient descent except instead of using the full gradient* $[\frac{\partial f}{\partial x_1}\frac{\partial f}{\partial x_2}\cdots\frac{\partial f}{\partial x_n}]$ *it uses a randomly selected feature* $[\frac{\partial f}{\partial x_i}]$ *for some* $i \in \{1..n\}$

Both of these methods drastically reduce the number of computations at each iteration; however, they are noisy estimators so they result in the need for more epochs to reach a desired error tolerance.

## 3.6 Loss Functions

As mentioned above the loss function gives a measure of ”goodness” for the output or output(s) of a neuron and is used to optimally tune the set of weights feeding a neuron. A Loss Function is part of a more general class of functions called objective functions which are used in mathematical optimization. In Optimization the idea is to find the global maximum or minimum of the objective function. In general, if a function is not convex it is not guaranteed to have a single (global) minimum or maximum and thus the problem simplifies to finding local maximum or minimum. For training of neural networks an objective function is used which measures the error or loss in a neuron’s output, so for training of neural networks we seek to find a local minimum as the goal will be to minimize the amount of error in the system.

Almost all ANNs use either a cross-entropy loss function (of which there are several depending on the type of network) or a mean squared error loss function.

### Cross Entropy

**Definition 3.5.** ***Entropy*** *Entropy is the average level of information or uncertainty inherent in a random variable’s possible outcomes. It is formally defined as:*

$$H(x) = -\sum_{x\in X} P(x)\log(P(x)) = E[-\log(P(x)]$$

*where* $x$ *is a random variable in the alphabet* $X$ *distributed according to* $p(x) : X \to [0,1]$ *i.e., the probability distribution of* $x$
*This can also be considered the number of bits required to transmit a randomly selected event from a probability distribution.*

Cross-entropy is the measure of the difference between two probability distributions. Cross-Entropy originates from the field of information theory and is based on the idea of entropy.

**Definition 3.6.** ***Cross Entropy*** *Is the number of bits required to represent an event from one estimated distribution $q$ compared to the true distribution $p$ and is defined as:*

$$H(p,q) = -E_p[\log(q)] = -\sum_{x \in X} p(x)\log(q(x))$$

*where $E_p$ is the expected value operator with respect to $p$.*

From this we see that if a probability distribution $q$ is the same as $p$, then the cross-entropy is equal to the entropy.

For training a neural network classifier with $m$ classes, we first use a method called one-hot-encoding. One-hot-encoding is a method for converting categorical data into numerical data which is required for processing data in a neural network. Since most categorical data does not have an implicit ordering to the categories, we instead assign each category a different random integer $1..m$ and then give each example as output an $m \times 1$ vector $I_n = [0, 0, ..., 1, ..., 0, 0]$ where a one is placed in the $n$th position given that the example's class is assigned the integer $n$. This then gives a discrete probability distribution of $I_n$ for each example.

Calculating the entropy for each example results in an entropy of 0. Thus, if we are able to perfectly represent the true distribution of each example in the output of the ANN, the cross-entropy will be equal to the entropy which is 0. As the distribution estimated by the output neurons of the ANN differs from the true distribution the cross-entropy will grow. Therefore, by minimizing the cross-entropy of the output neurons we get the optimal weights for the network.

As an example, consider a binary classification with a target distribution of [0 1]. Calculating the cross-entropy of an estimated probability distribution results in the image show in Figure 3.9.

If the reader is familiar with maximum likelihood estimation, it turns out that while they are different functions/measurements, as a loss function the cross-entropy loss is equivalent to a maximum likelihood function under a Bernouli or Multinouli probability distribution [2].

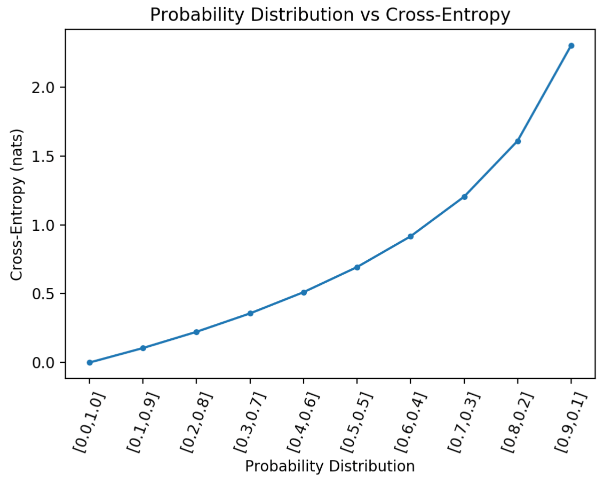


Figure 3.9: *Line Plot of Probability Distribution vs Cross-Entropy for a Binary Classification Task With Extreme Case Removed [2]*

**Mean Squared Error**

As discussed above, Cross-Entropy is suited well for classification problems where the network needs to output the probability of an input belonging to each class. However, if the problem is a regression problem rather than classification, meaning that we are trying to predict a real valued output, then cross-entropy will not work as well. The range of Cross-Entropy is $[0, 1]$ so many regression problems that require a range outside of this will either fail completely or require a scaling factor which does not perform well. Additionally, from a logical perspective, cross-entropy is attempting to compute the loss for an entirely different type of output then regression. As such, we need a different loss function for regression problems. This is where the mean squared error function is useful.

**Definition 3.7.** ***Mean Squared Error (MSE)*** *The mean squared error is the mean of the squared differences between the predicted values and actual values, and is defined as follows:*

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

*interpreted with respect to an estimator* $\hat{\theta}$,

$$MSE(\hat{\theta}) = E_\theta[(\hat{\theta} - \theta)^2]$$

It is fairly easy to see the motivation for MSE. If we have an estimate $\hat{Y}_i$ for $Y_i$ then the error for $\hat{Y}_i$ is $(Y_i - \hat{Y}_i)^2$ and the closer this value gets to 0 the closer $\hat{Y}_i$ approximates $Y_i$. So, by summing these errors we get an estimate of the loss in the estimators. This works better for regression as the loss is precisely the error in the estimates and the range of MSE is $[0, \infty)$. MSE is the most common loss function used in ANNs applied to regression problems.

## 3.7 Overfitting

Overfitting is one of the biggest problems in ANN's and in machine learning in general. The basis of the problem is: how does one build an estimator for a population, using a sample set, that adequately estimates the population rather than only the sample set. When training a Neural Network, a training set taken from some ambient population space is used to optimize the Neural Network estimator. The goal is to get the estimator to approximate the ambient population space rather than the training data set. Often times this is called "generalization"; that is, can the Neural Network "learn", i.e., approximate, patterns in the training data set which apply in general to the larger ambient population space. Overfitting occurs when the estimator estimates the training set space rather than the more general population space.

Typically, overfitting is detected by splitting the training data set into two different sets, one which will be used for training, and one which will be used for evaluating the model at each epoch. If the network performs with much higher accuracy on the training data set than the evaluation data set this generally implies that the network has overfitting. Meaning it has learned specifically the training data set and not the more general population space from which the training data was sampled.

### 3.7.1 Why Does Overfitting Occur

Overfitting occurs because of two main issues. The first is bias in the training data set.

**Definition 3.8.** ***Bias*** *Statistical bias is anything that causes a difference between the sampled data set and the population set from which it is sampled.*

In any data set there is statistical bias. Typically, this occurs when, due to the manner in which the data was sampled, some features are over or under represented versus the population. For example, if a random sample of people was taken on a college campus the age demographic of 18-22 would be over represented versus the actual population. Or if a political survey was conducted in a rural town in Kansas as opposed to downtown Los Angeles the political leanings of the constituents would be drastically different (skewed conservative or liberal respectively) as compared to the overall population.

While these biases seem obvious and should be easy to catch, many biases are not. Take for example the analysis done by Goldfarb in [21] on the VGG16 CNN [59] applied to a subset of ImageNet images [31]. The author found that an image of a black panther was incorrectly categorized as a chimpanzee because many of the training images of chimpanzees were taken at zoos and therefore had bars or cages in them. This led the network to associate bars with chimpanzees and so when presented with an image of a black panther in a cage it categorized it as a chimpanzee.

Besides bias, the other cause of overfitting is the ability of ANNs to learn very abstract and complex patterns. Due to this, if trained for long enough, ANNs are able to learn almost exactly and specifically each element in the training data set instead of learning general patterns that apply across categories. That is, rather than learning that tusks, large ears, and trunks are indicative of an elephant, the model is capable of learning the set of pixels in each specific image of elephants in the training set. This will of course not generalize to new pictures.

### 3.7.2 How To Solve Overfitting

There are many different methods for limiting overfitting in a neural network. The usefulness of each depends on the reason for the overfitting which is typically difficult to discern, although TDA is a method which may lead to a better understanding of overfitting, as is seen in [21]. Given this difficulty, it is standard to start with the easiest methods and work towards the more complex checking to see if any solve the problem. While a comprehensive review of techniques is beyond the scope of this paper, below is a synopsis of the more common techniques including the ones which were used during the analysis outlined in Section 7.2.

### 3.7.3 Partitioning the Dataset

The simplest way to limit the effects of overfitting is to split the dataset into training and validation sets. While this does not prevent overfitting as nothing is done to the data or methods being used, this does reveal the presence of overfitting which can lead to the introduction of other techniques to prevent overfitting as well as the prevention of overtraining.

### 3.7.4 Overtraining

One of the most common reasons for overfitting, and one of the easiest to fix is that of overtraining. Many practitioners are fooled into thinking that an

ANN should be trained until it reaches a specific accuracy, but often times this causes more harm than good. It is best to train an ANN until a ”knee in the curve” is reached in the accuracy. In training, the accuracy of most ANNs will increase steadily and then begin to level out at a certain point, normally this signals that the network has learned all of the general patterns that it can learn using the training set, architecture of the network, and methods supplied and that instead it is starting to specify on the individual training data. Thus, it is best to train to the point that the accuracy begins to level off and then stop as lower accuracy is better than overfitting since overfitting will lead to low accuracy when presented with samples not found in the training set. Figure 3.10 illustrates this knee in the curve where the accuracy levels off and the loss of the evaluation data set begins to increase.

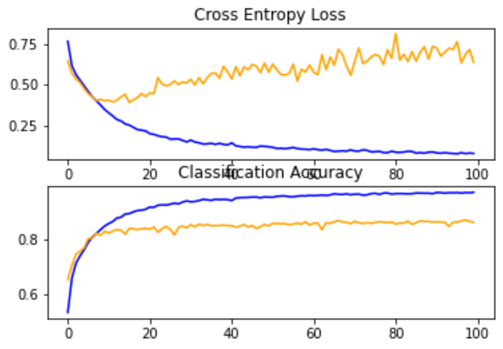


Figure 3.10: *Plot of Cross-Entropy Loss and Classification Accuracy of a CNN applied to a training data set (Blue) and evaluation data set (yellow) across 100 Epochs of training*

### 3.7.5 Optimizers

Optimizers are an add on to the gradient descent method used to train ANNs. With typical gradient descent (batch, stochastic, or mini-batch) the parameters used for the algorithm are constant. For instance, the learning rate is set before training and stays the same throughout training. Optimizers introduce an adaptive capability to this method. Optimizers seek to improve gradient descent by shifting the parameters, or sometimes by using multiple gradient descent algorithms and aggregating their results. As an example, many optimizers start with a specified learning rate and then gradually alter the rate across each iteration of the gradient descent algorithm based on the magnitude of the gradient, this means faster convergence while avoiding step sizes that miss the minimum. This method is known as momentum. Below is a list of the most common optimizers:

- Adagrad
- Adadelta
- RMSprop
- Adam

Adam is the most used optimizer.

### 3.7.6 Dropout

Dropout is a method for preventing overfitting that works by zeroing the output of a random set of neuron outputs in a given layer. This functions very similarly to the concept of boosting and works because if features that are important across a large number of samples (general features) are zeroed then the loss function will go up and other neurons will begin to learn these features, whereas if a neuron representing a feature specific to only one or a very small number of samples (specific feature) is zeroed it will not affect the overall error of the network and thus the loss function will not change by much.

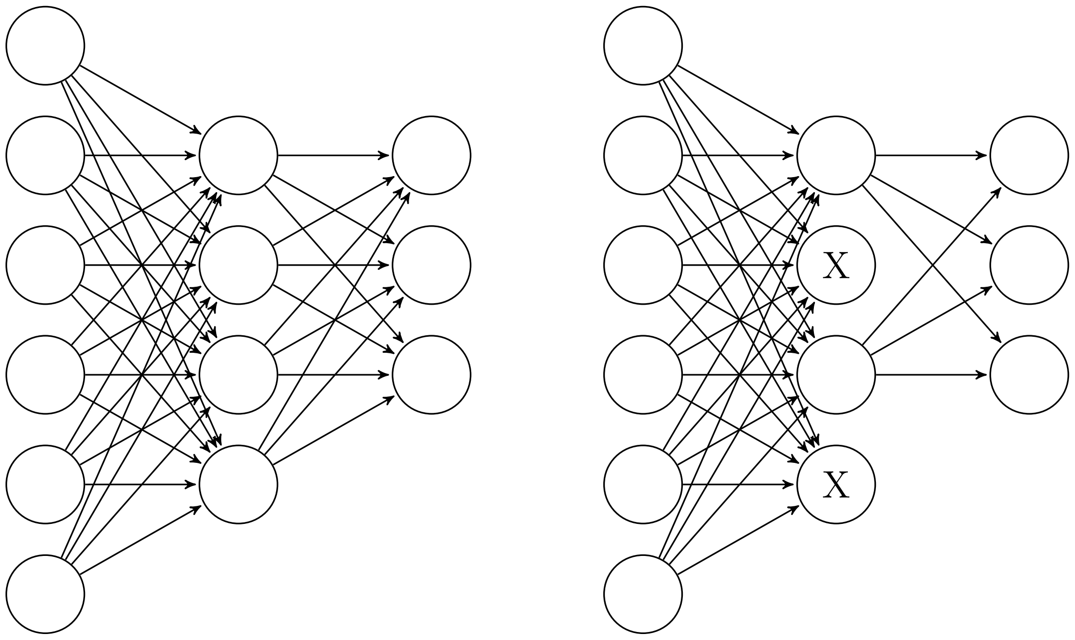


Figure 3.11: *Left: Fully Connected Neural Network Layers without Dropout, Right: Fully Connected Neural Network Layers with Dropout in the middle layer*

### 3.7.7 L1 and L2 Regularization

L1 and L2 Regularization works by adding a cost to the loss function associated with the magnitude of the weight values. By doing this, many of the less important features become or approach zero which simplifies the model and thus prevents overfitting. If $H(X_i, \beta)$ is the loss function for a model acting on a set of weights $\beta$, then the regularized loss function is:

$$H_R(X_i, \beta_) = H(X_i, \beta) + R(\beta)$$

where $R$ is some cost function on the weights.

The two main regularization functions are L1 (also known as Least Absolute Shrinkage and Selection Operator - LASSO Regression) and L2 (also known as Ridge Regression).

**Definition 3.9.** ***L1 Regularization*** *If $H$ is the loss function and $\beta$ the set of weights for a given node, then the L1 Regularized loss function is:*

$$H_R(X_i, \beta) = H(X_i, \beta) + \lambda \sum_{j=1}^{k} |\beta_j|$$

*where $k$ is the number of weights and $\lambda$ is some constant in* $[0, \mathbb{R}]$

**Definition 3.10.** ***L2 Regularization*** *If $H$ is the loss function and $\beta$ the set of weights for a given node, then the L2 Regularized loss function is:*

$$H_R(X_i, \beta) = H(X_i, \beta) + \lambda \sum_{j=1}^{k} \beta_j^2$$

*where $k$ is the number of weights and $\lambda$ is some constant in* $[0, \mathbb{R}]$

With L1 regularization the weights of less important features will be shrunk to 0 while in L2 regularization the weights of less important features will be shrunk but will still have some impact on the output. Because of this L1 regularization will be robust to outliers but will produce models capable of handling only simpler features; whereas L2 Regularization is able to handle more complex features but is not robust to outliers.

### 3.7.8 Data Augmentation

Data Augmentation is a technique which attempts to deal with the issue of having sparse training data. One of the big problems in training CNNs is the limited availability of data. The fewer images available to train with, the less effective a model will be and the more likely it will overfit. Data

Augmentation works by taking each image and creating a set of additional training images by performing one or more operations taken from a pool of operations which typical includes, rotation, flip vertically/horizontally, crop, add noise. By doing this the size of the training set is increased which provides more examples for the model to fit to and makes it harder to overfit since it becomes increasingly more difficult to fit each individual image as the number of images grows. It also helps the model generalize by presenting alternate views of images. If every image is taken at the same scale with the same angle to the object, training a model to perform image classification would become much easier. Unfortunately, in the real world, most images are taken from different ranges with different values of zoom, different rotations of the camera, different aspect angles, and different centerings. As such, performing data augmentation helps the model prepare for these more general cases.

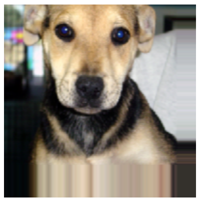


Figure 3.12: *Original Image before Data Augmentation*

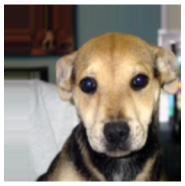
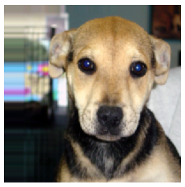
| | | | | |
|---|---|---|---|---|
| | |  | | |
| | | | |  |

Table 3.1: *Examples of Data Augmentation applied to an Image*

The downside to data augmentation is that it drastically increases the size of the training set which increases training time. A balance is needed between the number of additional images added by data augmentation and the need for reasonable training times.

## 3.8 Why CNN's

A big question has so far been left unanswered, why use CNNs over ANNs? While there are many answers for this, the most important boil down to two facts about ANNs:

1. ANN's assume that the input data has low dimension, i.e., if $x_i \in \mathbb{R}^k$ then $k$ is small.
2. ANN's assume that the input features are linearly independent

Unfortunately for images this does not hold. Thus, while ANNs can work on images they produce suboptimal models.

CNNs on the other hand use filters or lenses to look at portions of the input and try to observe features rather than relationships between the input values. It turns out that generally, the features become linearly independent even if the data is not, which in an image much of the data is not completely linearly independent. Also, by learning features rather than each individual pixel the dimension of the data being learned is much lower. Given this and the recent increase in computing power CNNs have become increasingly popular for handling image recognition and other image processing tasks.

## 3.9 Problems with CNNs

While CNNs are better suited to handle images than ANNs and have shown to be impressive image processing models, there are still many issues with CNNS.

**Large Number of Hyper-parameters**
The first is that throughout all the sections above there are a large number of user defined parameters (hyperparameters) that have a substantial impact on the system. Gradient Descent learning rate, regularization lambda, filter size, number of layers, etc. All of these impact the performance, usefulness, and functionality of CNNs. With so many parameters, it is incredibly difficult to tune each one optimally or to control for one to test how it drives the learning of the model. As such, most practitioners work using "rule of thumb" guidelines which may not be the correct choices for these parameters.

**Training Time**
Even with recent increases in processing speed and available memory, training CNNs is still very resource and time intensive. For the Dogs vs. Cats data set discussed later, the author's CNN still took approximately 6 hours to execute each run of 60 epochs even while running on a Google cloud server with GPU acceleration and with a relatively small set of images (only two classes). This training could not run on the author's own machine in any sort of reasonable

amount of time. Until computing resources grow exponentially more or further breakthroughs are made in optimization of training, the development of CNNs and their use will continue to be difficult.

**Use Outpaces Theory**
Like many areas in machine learning, practice far outpaces theory. The proliferation of libraries for creating CNNs has allowed users to create and use CNNs without an understanding of what is actually happening. These libraries make it easier to create a CNN by trial and error - adjusting parameters and the architecture without understanding why these adjustments are having an effect or what is happening in the model. This leads to results that are not understood and can lead to problems when the model behaves in ways that are not expected in the real world, especially when used in safety critical applications such as health care or self-driving vehicles. While these libraries are very beneficial, a substantial amount of work is needed to better understand the theory.

**CNNs Are Black Boxes**
Similarly, since CNNs are very complex data structures, little is often understood about the inner workings of a CNN. What patterns are learned and why is difficult to ascertain which can lead to problems as anytime the reason for decisions is not understood there is room for wrong decisions to be made, but in this case, it often happens without supervision. This is where TDA can lend a hand, as TDA, and the process outlined above and in Section 4, can help practitioners understand why certain decisions are being made by CNNs and how to solve issues with misclassification.

## 3.10 What Is a Convolutional Layer Weight Space?

In a convolutional layer, as described in Section 3.1, each filter position on the previous image contains a set of $m \times n$ weights (see Figure 3.2). These weights are initialized to some set of random values before the network is trained. Then using the backpropagation method described in Section 3.5, the weights are trained to minimize the overall loss function. Given that these weights are each a set of $m \times n$ values they constitute a subset of $\mathbb{R}^{m \times n}$. Initially before training, since they are randomly chosen, the set of weights will approximate a multivariate distribution which should approximate a sphere in $\mathbb{R}^{m \times n}$ with a density inversely proportional to the distance from the center (assuming Gaussian distribution). The question that we will attempt to answer, alongside the work of Carlsson, Gabrielsson, et. al. is after training what structure do these weights approximate? For instance, is the spherical structure maintained or perhaps do the weights begin to approximate only the surface of the sphere ($S^8$), or is there some other structure entirely? While there are weight spaces for each layer in a CNN that has weights the term weight space in this paper will be restricted to only convolutional layers.

## 3.11 Visual Cortex

Let us now pause for a second to consider a few interesting topics related to this study of CNN Convolutional Weights and the Primary Circle learned by their training. As discussed in Section 1, Neural Networks are based on the biological model of the neurons and connected synapses in the mammalian brain. In particular, CNNs are designed to mimic the connection of neurons in the visual cortex particularly C-Cells which the pooling layer is based off of and S-Cells which the convolutional layer is based off of. It turns out though that the neurons in the visual cortex are arranged so that different groups work together to detect lines of different orientations. From [27], "once information reaches the primary visual cortex, these circular receptive fields combine to create receptive fields that are activated by lines. These receptive fields cause neurons in the primary visual cortex to respond best to a line in a specific orientation. The firing rate of the neuron will increase as the line rotates toward the preferred orientation. The firing rate will be highest when the line is in the exact preferred orientation. Different orientations are preferred by different neurons."

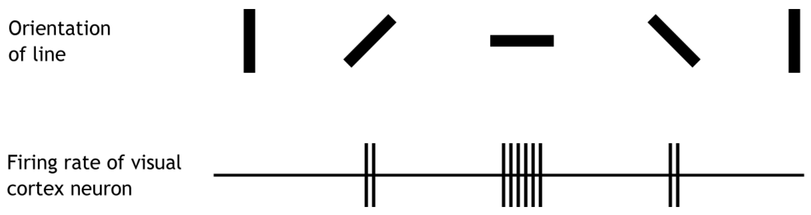


Figure 3.13: *Neurons in the primary visual cortex show increased firing rates in response to a preferred line orientation. Lines rotated away from the preferred orientation will not cause activity. CNS Receptive Field Responses by Casey Henley is licensed under a Creative Commons Attribution Non-Commercial Share-Alike (CC BY-NC-SA) 4.0 International License.*

This becomes particularly interesting in regards to this study of CNNs because what the work of Carlsson and Gabrielsson and what our work in this paper seek to show is that the nodes in the convolutional layers of CNNs trained on natural images turn out to mimic the exact behavior of the neurons in the primary visual cortex. That is, the annulus or $H_1$ cycle discussed in Section 1 and demonstrated in the subsequent work of this paper is made up of nodes which recognize or fire based on a specific gradient, i.e., orientation of a line. So what is seen is that constructing a CNN with the same basic functionality or architecture as the primary visual cortex and training it with natural images causes a CNN to actually learn the same structure that the neurons in the primary visual cortex are designed with.

In general, this process of recognizing gradients or orientations of lines is known as "Edge Detection". Edge Detection is one of the most important concepts in performing image classification as the discovery of edges is the first step in separating objects from each other and the background which must be performed before classifying. As such, Edge Detection is almost always considered the first step necessary when creating an algorithm to perform image processing. It is also quite interesting that the first step a CNN performs when attempting to do image classification is to perform edge detection even though this step is in no way instructed or architected in the design of CNNs. (See [64] for a more thorough discussion of edge detection.)

# 4 Topological Data Analysis

With the mathematical theory for Topological Data Analysis (TDA) in hand it is appropriate now to introduce the general notion of TDA. Often in data analysis applications, a set of samples or data points is known while the ambient space from which these points were sampled remains unknown. The goal then is to determine the ambient space from the samples so that more general conclusions can be drawn. While there exist many ways to try to determine the ambient space, one recent approach, which has shown much promise, involves using abstract simplicial complexes to derive topological data from data sets. This method has sparked the creation of a whole new field known as Topological Data Analysis. TDA seeks to utilize the topological or geometric structures underlying data to try to make larger generalizations about the ambient space and thus the data beyond just the samples being examined. This, at least in theory, prevents the problems that arise from overfitting in Neural Networks and other data analysis techniques.

TDA consists of four main areas referred to by Chazal and Michel [6] as the "TDA Pipeline". While these areas are not mutually exclusive, and many approaches overlap some or all of them, they illustrate the process that TDA takes to perform analysis and categorization of the data.

1. Data Processing - The data that we start with is assumed to be a finite set of point cloud data. This data may come with a given metric or a metric may need to be chosen. In either case, the choice of metric is pivotal to the type of features which will be discovered via the TDA process. How to choose an appropriate metric and how metrics impact the output remain open questions.

2. Structure Application - Once the input data has been described, character-

ized, and an appropriate measure has been applied, a "continuous" structure must be applied to the data. This is done to provide a more continuous shape to the discrete data so that the underlying structure can be analyzed or examined. There are many ways to apply a structure to the data but simplicial complexes and filtrations are the most common.

3. Information Extraction - Once the continuous structure is applied to the data one can begin to extract topological or geometric information from this structure.

4. Feature Description - Once the information from the structures is extracted it can be examined to produce features of the data or - more importantly - of the ambient space from which the data was taken.

A thorough discussion of these fields is beyond the scope of this work but the reader is referred to [6] for a more in-depth study. Nevertheless, this is mentioned to show the broader area of TDA and situate the work outlined in this paper in the larger context of the field of TDA. Specifically, our work falls under the concepts of structure application and information extraction, namely the construction of abstract simplicial complexes, filtrations, and persistent homology. This includes a novel method for building simplicial complexes from high dimensional data sets known as Mapper.

In the remainder of this section, we will discuss the concept of data pre-processing for TDA and then the primary methods for performing TDA on point cloud sets. Then in Section 5 we describe the process used for applying TDA to the convolutional weight vectors from the CNNs analyzed in sections 6 and 7.

## 4.1 Pre-Processing Data for TDA

The TDA Process begins with a point cloud data set which must be pre-processed before analysis can occur. The pre-processing consists of normalizing the data and then filtering to limit the results to the dense clusters of data (a way of removing noise) and finally PCA is performed to reduce the dimension and provide a lens on the data. After this, the data is ready for analysis. Specifically, we will use a filtration of Rips Complexes to compute persistence diagrams and the "Mapper" algorithm, which will be described in this section, to provide a 2-D visualization.

### 4.1.1 Normalization

In the first step, the data is normalized. By doing this, the data, which may or may not have units, is transformed so that the relative values can be compared rather than the absolute values. In many applications, such as CNN weight values, the values have no bounds and as such the absolute values have little to no meaning. Rather, the values relative to other elements of the data set are what is important. So, by normalizing, we transform the vectors in the point cloud set to relative values preserving the ratios between the values. This becomes especially important when comparing weights over multiple runs as the absolute values may differ wildly but the relative ratios should remain the same. In a CNN, higher weights are associated with features that are important for classification (i.e., the weight is a measure of importance), but only as a comparison to other weight values. Their absolute magnitudes are essentially meaningless as the weights of the network are unitless. The values have much more to do with the training process and the magnitude of the input values (in CNNs for image processing this is typically related to contrast and brightness). Because of this, the information and relationship of the weights are stored in their relative value to other weights. Thus, it is necessary to normalize so that ratios between weights in different parts of the layers can be compared.

In normalization, each weight vector is normalized so that it has unit variance and centered so that its mean is 0. More formally if $v = \{v_i\}_{i=1..n}$ is a weight vector, with mean $\mu = \frac{1}{n}\sum_{i=1}^{n} v_i$ and variance $\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(\mu - v_i)^2$ then the normalized weight vector $\hat{v} = \{(v_i - \mu)/\sigma^2\}_{i=1..n}$.

As discussed in Section 1, this is the same process that Mumford et. al. [47] used on their analysis of high-variance local image patches. The result of normalizing the weight vectors is a projection from the weight space $\mathbb{R}^{m\times n}$ to $S^{(m\times n)-1}$ i.e., the surface of the ball with radius 1 in $\mathbb{R}^{(m\times n)-1}$.

### 4.1.2 K-Nearest Neighbors Density Filtration

Now that the data is normalized, it is necessary to filter the data. While it is possible to perform the TDA steps on the full data set, this requires a very large amount of computing power and is not really necessary. The areas where the data is "clumped together", i.e., dense, are the areas of interest and so by filtering out the vectors which are not located near others, some of the noise within the data set is filtered out. This is akin to throwing out outliers. Recall that the information stored in the vectors is their relative value to each other not their absolute magnitudes, so in this case rather than outliers lying at the extremes of the dataset, outliers are weights which are not located near other weights in the weight space.

To filter the data, a K-Nearest Neighbors Density Filtration is used. The K-Nearest Neighbors Density Filtration comes from the concept of the K-Nearest Neighbors Density Estimator which was first introduced by Loftsgaarden and

Quesenberry [42] as an estimator of the probability density function of a random variable $X \in \mathbb{R}^d$ with a continuous distribution function. The K-Nearest Neighbors Density Estimator or KDE is defined as follows:
Given a point $x \in \mathbb{R}^n$, with $D_k(x)$ the distance from $x$ to its $k$-th nearest point, then

$$\hat{p}_{kde} = \frac{k}{n} \cdot \frac{1}{V_d \cdot (D_k(x))} = \frac{k}{n} \cdot \frac{1}{Volume\ of\ B_d(x, D_k(x))}$$

where $V_d = \frac{\pi^{d/2}}{\Gamma(d/2+1)}$ is the volume of a unit dimensional ball in $\mathbb{R}^d$ and $\Gamma(x)$ is the Gamma function.

Loftsgaarden and Quesenberry showed in [42] that $\hat{p}_{kde}$ is a consistent density estimator. As such, it is a valid choice for use in performing a density filtration. This is the same density filtration used by Carlsson and Gabrielsson in [18] which is why it was chosen for our work in order to show consistency.

In K-Nearest Neighbors Density Filtration the pairwise distance is calculated for each set of points $v_i, v_j \in V$ the set of weight vectors where $i \neq j$. Any distance metric can be used, but the standard Euclidean distance is most common and was used for this research. Once all of the distances are computed the $k$-th nearest distance to each point is used and then the top $\rho\%$ of points are kept. So, if $k$ is 100 and $\rho$ is 0.3 (30%) and the data set has 3000 vectors, then the top 1000 vectors with the shortest distance to the 100-th closest point from each are kept. This results in a reduction of the data where outliers are filtered out and areas of interest are highlighted.

To be more precise let $v_i, v_j \in V$ be points in the point cloud $V$ with distance function $d(v_i, v_j)$. Then for each point $v_i \in V$ we can define the set of distances of each point from $v_i$, $D^i = \{d(v_i, v_j) \mid \forall v_j \in V\}$ and the sorted set of distances $\tilde{D}^i = \{d_k \in D^i \mid d_k \leq d_{k+1}\}$. Then we can define $\tilde{D}^i_k$ as the $k$-th element of $\tilde{D}^i$ (i.e., the $k$-th closest point to $v_i$). Using this, let $A^k = \{D^i_k \mid i = 1..|V|\}$ and $\tilde{A}^k = \{a_i \in A^k \mid a_k \leq a_{k+1}\}$. Then the K-Nearest Neighbors Density Filtration $F(k, \rho) = \{\tilde{A}^k_i\}_{i=1..\lfloor \rho * |A^k| \rfloor}$

### 4.1.3 PCA

Another step often performed in data pre-processing for TDA is dimensionality reduction. Dimensionality reduction projects the data onto a space which is easier to visualize and understand, and is often used whenever a lens is needed to summarize the data in some way. Specifically, as will be seen in the next section, this is used in the TDA algorithm known as Mapper. While there are many ways to perform dimensionality reduction on a point cloud set, the most popular is Principal Component Analysis. In Principal Component Analysis (PCA) the data is projected onto the first $n$ principal components of the data. The principal components are unit vectors where the $i$-th vector is the vector which best fits the data (i.e., minimizes the least squares distance) while being orthogonal to the first $i-1$ components. These principal components are the eigenvectors of the covariance matrix of the data set. These components then form an orthonormal basis and are used to perform a change of basis which reduces the dimension of the data to the number of components used. For more information on PCA see [1]. Gabrielsson and Carlsson state that other lenses can be used to reduce the dimension of the data set, but that PCA gave the best results [5]. PCA is used for the analysis in this paper.

## 4.2 TDA Methods

The primary TDA methods we used in our analysis are the Mapper algorithm and Persistence Diagrams. These are two of the most commonly used TDA methods and were the methods which would allow us to best evaluate 1-st persistent homology groups and the existence or lack of the Primary Circle corresponding to a high persistence cycle in $H_1$. The Mapper Algorithm is discussed in detail in the next section. The Persistence Diagram was discussed in Section 2 and so it will be glossed over here.

In short, a Filtered Rips Complex is constructed using the point cloud set of convolutional weights. This filtered complex is then used to compute the $p$-persistent Betti numbers of $H_0$ and $H_1$, which are then plotted based on the filtration level in which each cycle was created and the level that it died (i.e., the $\mathcal{P}$-intervals). These are then plotted in a diagram with the axes of birth and death (see Figure 4.1). In Figure 4.1, the value on the birth and death axes is the threshold value $\epsilon$ used at the corresponding level of the filtration. Recall that the Rips Complex $R_\epsilon$ is defined by a threshold value $\epsilon$. The Filtered Rips Complex then is formed by gradually increasing this threshold at each level of the filtration. So, rather than listing out the numerical level of the filtration, we can instead use the value $\epsilon$ to indicate where in the filtration a cycle was created and a homologous cycle died.

### 4.2.1 Mapper

Mapper is a computational algorithm created by Singh et. al. [60] which is intended to create simple descriptions of high dimensional data sets through the use of simplicial complexes, while still maintaining the topological and geometric properties of the data set at a given resolution. Mapper has been widely adopted by the TDA community as a method for determining the structure of high dimensional data at a specified resolution and, together with the use of persistence diagrams, can provide insight into the underlying topological structure of a data set. In particular, Mapper was the method used to visualize the structure of the CNN weights in the analysis of the Dogs vs Cats CNNs.

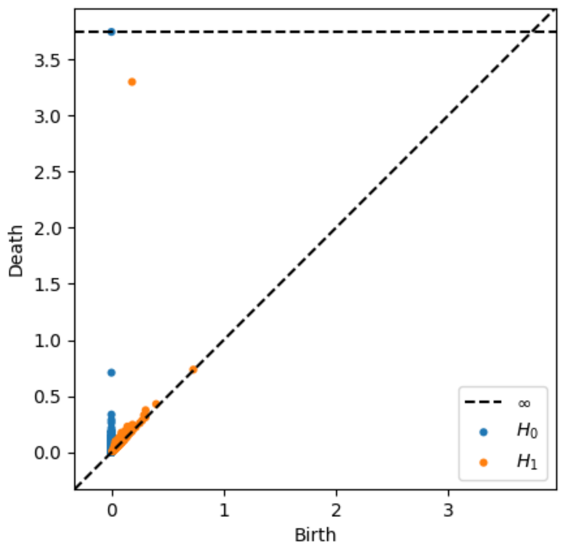


Figure 4.1: *Example of a Persistence Diagram created in Python using the Ripser package from Sckikit TDA [53]*

While the main use of the Mapper algorithm is for use in determining the structure of point cloud data, in general it can be applied to any topological space. As such we begin with a generic topological space and then discuss in more detail its application to point cloud sets.

**Definition 4.1.** *Given points $x$ and $y$ of a space $X$, a path in $X$ from $x$ to $y$ is a continuous map $f : [a, b] \to X$ such that $f(a) = x$ and $f(b) = y$. A space $X$ is said to be path connected if $\forall x, y \in X$ there exists a path in $X$ from $x$ to $y$.*

**Definition 4.2.** *The Path Connected Components of $X$ are the equivalence classes defined by the equivalence relation $x \sim y$ if there exists a path in $X$ from $x$ to $y$.*

**Theorem 4.1.** *The path connected components of $X$ are path-connected disjoint subspaces of $X$ whose union is $X$, such that each nonempty path-connected subspace of $X$ intersects only one of them. [48]*

Let $T$ be a topological space, then the Mapper algorithm is

1. Chose $f : T \to Z$ a continuous function mapping $T$ to a metric space $Z$ (typically $Z$ is $\mathbb{R}^n$)
2. Construct an open covering of $Z$, $\mathcal{U} = \{U_\alpha\}_{\alpha \in A}$ for some finite indexing set $A$.
3. Define $X_\alpha = \{x | f(x) \in U_\alpha\}$ Given that $f$ is a continuous map, the sets $f^{-1}(U_\alpha)$ also form an open covering of $T$.

4. Decompose $f^{-1}(U_\alpha)$ into its path-connected components, so $f^{-1}(U_\alpha) = \bigcup_{i=1}^{\beta_\alpha} V(\alpha, i)$ where $\beta_\alpha$ is the number of path-connected components in $f^{-1}(U_\alpha)$

5. Define $\mathcal{M} = \{\{V(\beta,\gamma_i)\}_{\beta\in B, i\in\{1..|B|\}} \subseteq \{X_\alpha\}_{\alpha\in A} :$
   $B \subseteq A, \bigcap\{V(\beta,\gamma_i)\}_{\beta\in B, i\in\{1..|B|\}} \neq \emptyset\}$
   i.e., $\mathcal{M}$ is the simplicial complex $\mathcal{N}(\mathcal{U})$ whose vertex set is $V(\alpha,i)$ and where a $k$-simplex exists in $\mathcal{M}$ for all non-empty intersections of $k+1$ $V(\alpha,i)$'s

Consider now $X$, a point cloud set inside of a metric space $S$. The goal of the Mapper algorithm on point cloud sets is to construct a simplicial complex $\mathcal{M}$ from $X$ which captures the topological properties of $S$ at a specified resolution. The problem with $X$ is that it is itself not necessarily a subspace and therefore the sets $X_\alpha = f^{-1}(U_\alpha)$ cannot be decomposed into path connected components. Instead a clustering algorithm is used to separate $X_\alpha$ into disjoint sets where the points in each cluster have some notion of "closeness". By applying a clustering algorithm $\mathcal{C}$ to each of the $X_\alpha$ we get $X_\alpha = f^{-1}(U_\alpha) = \bigcup_{i=1}^{\beta_\alpha} C_\alpha^{\beta_\alpha} = \bigcup \mathcal{C}(X_\alpha)$. The Point Cloud Mapper algorithm is:

1. Define a continuous map $f : S \to Z$, where $Z$ is the reference metric space and $X \subseteq S$

2. Construct a finite open covering $\mathcal{U} = \{U_\alpha\}_{\alpha\in A}$. of $Z$

3. Define $X_\alpha = \{x | f(x) \in U_\alpha \, x \in X\} = f^{-1}(U_\alpha) \cap X$

4. Apply a clustering algorithm $\mathcal{C}$ to each $\{X_\alpha\}$, this results in $\mathcal{C}(X_\alpha) = \{C_\alpha^1, C_\alpha^2, ...C_\alpha^{\beta_\alpha}\}$ such that $X_\alpha = \bigcup_{i=1}^{\beta_\alpha} C_\alpha^{\beta_\alpha} = \bigcup \mathcal{C}(X_\alpha)$ and $i \neq j \to C_\alpha^i \cap C_\alpha^j = \emptyset$

5. Define $\mathcal{M} = \mathcal{N}(\{\mathcal{C}(X_\alpha)_{\alpha\in A}\})$, that is
   - 0-simplexes (nodes) are the clusters $C_\alpha^\beta$
   - Non-empty intersections form edges/larger k-simplexes
   i.e., a family $\{(C_{\alpha_0}^{\beta_0}), ..., (C_{\alpha_k}^{\beta_k})\}$ spans a k-simplex if and only if their intersection is nonempty

Mapper has four different choices/hyper-parameters which must be made and which impact the outcome of the algorithm. These are the filter map $f$, the reference metric space of the filter $Z$, the open covering $\mathcal{U}$, and the choice of clustering algorithm.

The function $f$ is referred to as the filter or lens on the data. The job of the filter is to embed the structure of the more complex (or unknown) topological space in a simpler space. Examples of filters include,

- Distance function $f(x) = \parallel x - p \parallel_d$

- A dimension reduction function such as Principal Component Analysis (see Section 4.1.3)
- Density estimates
- The centrality function $f(x) = \sum_{y \in X} d(x, y)$ or the eccentricity function $f(x) = max_{y \in X} d(x, y)$

In our analysis we used 2-Component PCA as the filter.

The reference metric space is typically chosen in conjunction with the filter as the filter often drives the choice of reference space. The reference space chosen is almost always $\mathbb{R}$ or $\mathbb{R}^2$ as these are easily visualized and using a higher dimensional space does not typically provide any additional structural information.

After choosing a filter and reference space the open cover $\mathcal{U}$ must be chosen. The typical choice (when the reference space is $\mathbb{R}^n$) for $\mathcal{U}$ is a set of evenly spaced overlapping intervals. When this is the choice for the open cover, there are two corresponding parameters that must be chosen, resolution and gain. The resolution is typically the length of each interval although in some work the resolution refers to the number of intervals as in [18]. For the sake of clarity, in this paper we refer to this as number of cubes instead of resolution. The other parameter, gain, is typically the percentage of overlap. Although, sometimes the gain is defined as gain = 1 / (1 - percentage of overlap). For the sake of clarity, we will use percentage of overlap.

Lastly the clustering algorithm must be chosen. Any clustering algorithm can be chosen as long as it produces disjoint clusters. Examples include K-Means [41], Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [13], and Agglomerative Hierarchical Clustering [56]. In their original paper Singh et. al. [60] chose a single linkage clusterer [32, 33]. This is also the method chosen by Carlsson and Gabrielsson in their work on MNIST [18] as it is implemented in their proprietary software AYASDI. Single-linkage clustering is a form of agglomerative clustering.

Figure 4.2 gives two examples of the Mapper algorithm. In the top example, a 3-dimensional double torus is shown with one hole much larger than the other. Since the smaller hole is completely covered by a single $U_\alpha$ while the larger hole is not the resulting simplicial complex has one 1-cycle (element of $H_1$) instead of two, even though the topology of the double torus has Betti number $\beta_1 = 2$. The bottom example shows the Mapper algorithm on a point cloud set, specifically a set arranged in a circular pattern. Since no single $\mathcal{U}_\alpha$ covers the whole set, a 1-cycle appears in the simplicial complex.

These examples illustrate the need for care in choosing the filter and open cover for the Mapper algorithm. Typically, multiple different runs are needed with varying parameters to see what structures appear in the data. This is of course the importance of the persistent homology structure and associated

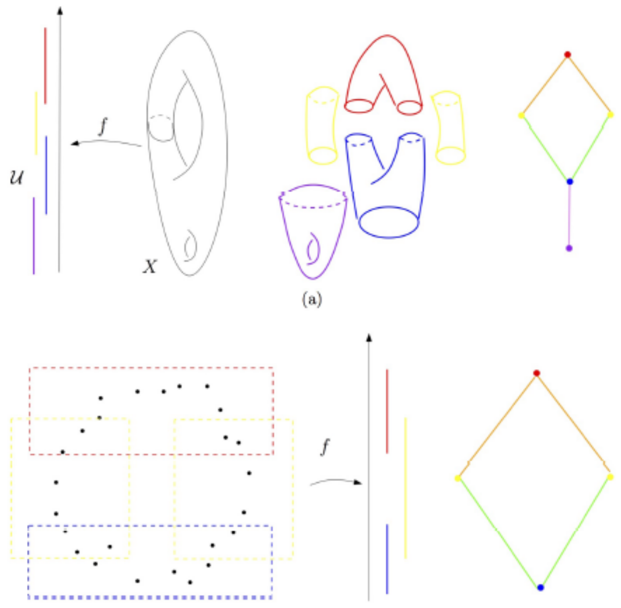


Figure 4.2: *Visual example of the Mapper algorithm [6]*

visualizations such as the persistence diagrams in that they show the structure over many different resolutions rather than a single one. However, the persistence visualizations only give persistent Betti numbers and so indicate the homology type of the ambient space for the point cloud data, but they fail to give any indication of what that structure truly looks like. This is where the Mapper diagram can be of great value. These two facts together are why we utilized both persistence diagrams and the Mapper algorithm in our analysis.

## 4.3 TDA Process

In order to perform the TDA processes outlined above we used the Python programming language. It is possible to perform the calculations by hand using the TDA methods outlined above; however, that would have taken a painful amount of time so instead it was decided to use computational packages to perform the work once the mathematical theory had been understood and verified. In order to do this, the Scikit Learn Preprocessing package [57] was used to perform the normalization of the data. After this, we used a K-Nearest Neighbors Density filtration function which we wrote in Python, and finally the PCA package from the Scikit Learn Decomposition library [57] was used to apply the PCA lens to the filtered and normalized data.

Once the data had been pre-processed the Ripser package [53] which is a part of the Scikit TDA package was used to generate the persistence diagrams.

Then the Keppler Mapper package [37] from the Scikit TDA package was used to build out the Mapper diagrams. This package implements the Mapper algorithm which is outlined in the TDA section. All of our analysis code can be found in a GitHub repository [61].

# 5 Methods

Now that we have considered the historical context for our work in Section 1 and the theoretical framework in sections 2-4 it is time to move into the actual work. Recall that the objective is to study the structure of weight spaces in CNNs built to classify natural images and in particular a previously unstudied complex image set known as Dogs vs. Cats. In so doing we would like to understand if the structure of the weights matches the known space of high-variance local image patches of natural images which was found by Mumford et. al. [47], i.e., a manifold with non-trivial geometry and in particular a non-empty first homology group. The presence of a structure approximating this space in the weights of a CNN built on natural images gives a pathway for a better theoretical understanding of CNNs and also gives increased confidence in the CNN model. This structure was found by Carlsson and Gabrielsson in CNNs built on MNIST, CIFAR-10, and SVHN [18] and used to posit the Carlsson-Gabrielsson Hypothesis stated in Section 1.2. The datasets used in [18] are fairly simple so our work is to extend the study to a more complicated dataset and show that the structure of the weights for CNNs built on more complicated datasets still match that of high-variance local image patches of natural images.

Our methodology is as follows. We first start by designing a CNN architecture that achieves reasonable accuracy and does not have overfitting. While an overfit or low accuracy CNN may have the same structure, these models are generally not considered valid and therefore there is little point in studying them. Once a valid architecture has been defined, we train $M$ number of CNNs for $N$ epochs each recording the weights from the convolutional layers at each epoch.

After completing the training, the weights must be pre-processed. As discussed in Section 4, this is done by normalizing the weights and then using a filtration to find the densest sets of points. In our case, we use the K-Nearest Neighbor Density Filtration. Once the pre-processing is complete the weights are ready for TDA analysis. In our work, we used two different TDA methods for analysis. In the first, we used a filtration of Rips Complexes to build the Persistence Homology and Persistence Diagrams in order to determine the Persistent Betti numbers. The other method used is the Mapper algorithm. This was used to create a visualization of the weight structure. Lastly, the Persistence Diagrams and Mapper Diagram are inspected to determine if there is any visible evidence of a clear structure to the weight data.

In particular, we are looking for two main results. First, we expect to see a single high-persistence element in $H_1$ (cycle). This will tell us that the structure of the weights approximates a manifold with non-trivial geometry and is consistent with what was found by Carlsson and Gabrielsson. It should be noted here that in some cases Carlsson and Gabrielsson found three elements of $H_1$ rather than one; however due to the use of grayscale images for MNIST and separate color channels for Dogs vs Cats we do not expect to see one for each color; therefore, only 1 $H_1$ cycle should appear. The other expected result is revealed by the Mapper diagram. In the Mapper diagram, we expect to see the Primary Circle, i.e., a circle of rotating gradients. This corresponds to the cycle found in the persistence diagram. Supporting all of this we also expect that as training increases the persistence of the $H_1$ cycle and the clarity of the Primary Circle should also increase at least until the model hits a "knee in the curve" in its training accuracy. (Typically, Neural Networks increase in accuracy up to a point where the line becomes asymptotic or may actually decrease, this is known as the "knee in the curve".)

In our work, we applied the methods described above to CNNs trained for two different datasets, MNIST and Dogs vs Cats. For MNIST, we followed the work of Carlsson and Gabrielsson in [18]. As discussed in Section 1, for CNNs trained on the MNIST dataset, Carlsson and Gabrielsson found a clear high persistence cycle in $H_1$ in the persistence diagrams and the Primary Circle present in the Mapper diagram. My first goal was to replicate their results since they used a proprietary software package (Ayasdi) and thus their work has so far not been replicated. Additionally, this will allow us to verify our methods and show that they are valid.

To study MNIST we designed a CNN with the same architecture as used in [18] using the Python programming language. 100 different CNNs (with this architecture) were trained and the weights from their convolutional layers were extracted resulting in 64*100 = 6400, 9-dimensional weight vectors from Layer 1 and 32*64*100 = 204800 9-dimensional weight vectors from the second layer. These weights were then analyzed using the methods described above. The full process and results are outlined in Section 6.

## 5.1 Main Work: TDA of CNNs Trained on Dogs vs Cats

After analyzing the MNIST dataset we turned to the original portion of the work which is a study of CNNs trained using the Dogs vs Cats dataset. We started by training a number of different architectures, using the Python programming language, in order to find one that achieved reasonably high accuracy ($> 90\%$) without overfitting. Once a good architecture was found 50 different CNNs (with this architecture) were trained to 60 epochs using Python on the Google Colab cloud server [24] and "The Foundry" [28], the computing cluster at the University of Missouri Science and Technology. Another CNN was built by adding an additional convolutional layer (4 instead of 3) to the good architecture. Together these CNNs were then analyzed to determine the structure of the convolutional weights.

In order to study the structure of the CNN weight spaces from CNNs trained using Dogs vs Cats, we used two different paths. The first was a statistical analysis using Voronoi Cells and the Kullback Leibler Distance (see Section 7.3.1). This method is a more traditional method which was used to determine whether the weights appeared with some sort of Normal or Uniform distribution. If the weights follow the same structure as expected, the distribution of weights in the weight space should appear highly non-normal.

After performing a statistical analysis, we moved to the application of TDA methods. Specifically, we used the Persistence Homology and Persistence Diagrams to determine the Persistent Betti numbers this should show the presence of a high persistence cycle in $H_1$ (the 1st-homology group). Then the Mapper algorithm was used to create a visualization of the weight structure. This could then show whether the data was arranged in the expected Primary Circle model. This process was performed using the methods outlined in Section 4.3 executed on the University of Missouri Science and Technology cluster "The Foundry" [28] using the Python Scikit Learn and Giotto TDA packages (the code can be found in the GitHub repository [61]). The full process and results are outlined in Section 7.

# 6 MNIST Analysis

As briefly discussed in the prior section, the first step in our research was to replicate the work of Carlsson and Gabrielsson in [18]. Using the MNIST data set a CNN of the same architecture was created and trained on the data set. The results were then compared to the work in [18]. Since Carlsson and Gabrielsson used a proprietary software suite to achieve their results and did not publish any of their code their results have been left unverified. An attempt was made by Larsen [39] to verify the results of Carlsson and Gabrielsson with only limited success. By replicating the work of Carlsson and Gabrielsson we were able to verify their results and also demonstrated the validity of our methods and subsequent findings. This section outlines our replication of the Carlsson and Gabrielsson work on MNIST as well as further research on CNNs built using MNIST. We were able to verify the results found by Carlsson and Gabrielsson. We also were able to show why their work was unable to find a clear $H_1$ cycle in their 2nd Layer and how to improve on their method in order to show that a clear high persistent cycle does exist in the 2nd layer. Further, we explored different CNN architectures and found an interesting and not fully explained result on CNNs for MNIST with more than two layers. These results are outlined in the rest of this section.

## 6.1 MNIST Dataset

The MNIST data set [62] is a set of 28x28 pixel images of handwritten digits (0 to 9) taken from two databases created by NIST which were sampled from a set of high school students and employees of the US Census Bureau. The data set is commonly used in machine learning and image processing systems. It consists of 60,000 training images and 10,000 test images. Below is an example of some of the images from the MNIST data set.

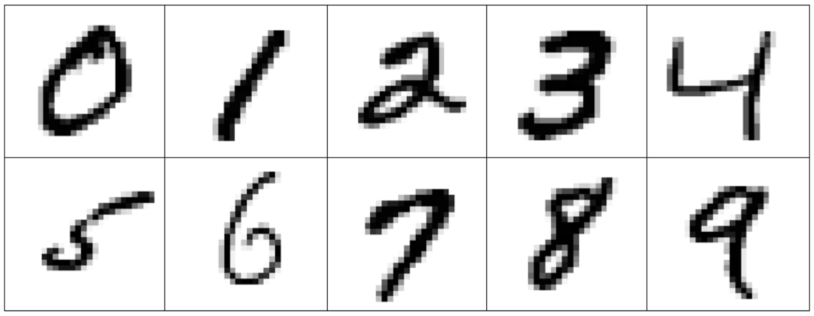


Figure 6.1: *Example images from the MNIST dataset*

MNIST is commonly used because it consists of relatively simple images which are already tagged with classes (which digit was being drawn). Because the images are small (28x28 pixels) the amount of memory needed to hold and process the data is substantially less than that needed to process standard full-size images. This makes training and testing much quicker, and thus allows for easier experimentation of networks and their architectures.

Using the MNIST data set we started by following the work of Gabrielson and Carlsson and attempted to replicate the results found here [18] before moving on to a more complicated data set. The goal in replicating the results of Gabrielson and Carlsson was to further validate their results since they do not offer publicly available source code to repeat their experiments and also to ensure that our code was functioning properly.

## 6.2 Gabrielson and Carlsson MNIST CNN

According to [18], Gabrielson and Carlsson used the MNIST data set to train 100 CNNs with the architecture listed in Table 6.1 below for 40,000 batch iterations with a batch size of 128, which turns out to be around 85 epochs. This resulted in a test accuracy of about 99.0%.

| Layer | Type | Input Dim. | Channels | Activation |
|---|---|---|---|---|
| Layer 1 | Convolutional | 28x28 | 64 | ReLU |
| Layer 2 | Max Pooling | 14x14 | 64 | n/a |
| Layer 3 | Convolutional | 14x14 | 32 | ReLU |
| Layer 4 | Max Pooling | 7x7 | 32 | n/a |
| Layer 5 | Fully Connected | 7x7 | 64 | ReLU |
| Layer 6 | Flatten | | 3136 | n/a |
| Layer 7 | Dropout (50%) | 3136 | 3136 | n/a |
| Layer 8 | Fully Connected | 3136 | 10 | softmax |

Table 6.1: *Architecture of the MNIST CNN created in [5]*

After performing the training, the weights of the first layer were extracted as 9-dimensional vectors which were then normalized. After this a k-nearest-neighbor density filtration with k = 200 and $\rho = 0.3$ was used to get 1920 points. To this point cloud Carlsson and Gabrielsson applied TDA, specifically they used Ayasdi to create a Persistence Diagram and apply Mapper (resolution = 30, gain = 3) with Variance Normalized Euclidean Norm and two PCA lenses. The resulting graph can be seen in the Figure 6.2.

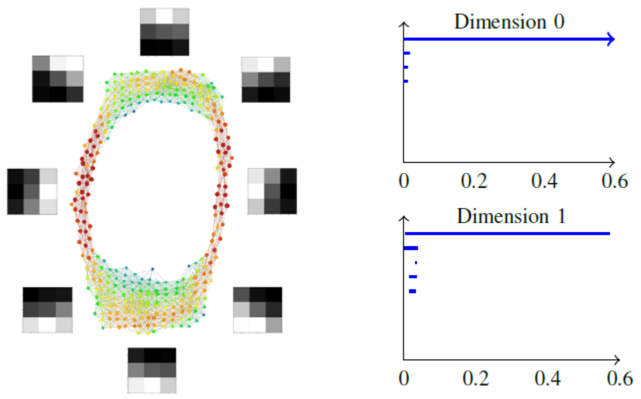


Figure 6.2: *Mapper output of the first layer of the MNIST CNN created and published in [18]*

As can be seen from Figure 6.2 there is a very clear cycle in $H_1$, with a long lifespan (persistence), shown in the bar code plot. This cycle, when the clusters making up the cycle are examined, matches the Primary Circle (rotating gradients). This can be seen on the left of Figure 6.2.

## 6.3 Author's Replication

In order to replicate the results of Gabrielsson and Carlson we utilized the Python language and available package repository, primarily Keras [35], Scikit Learn [57], and Giotto-TDA [20]. Analysis and training of CNNs was done in Spyder on an Intel(R) Core(TM) i5-4210U CPU  1.70GHz with 8GB of RAM hosting Windows 10, Google Colab using GPU acceleration [24], and the University of Missouri Science and Technology computing cluster "The Foundry" [28].

The MNIST data set was obtained from the Keras Python package and a CNN was built using the Python Keras package with the same architecture as is shown in Table 6.1 above.

This CNN was then trained using the optimizer ADAM, 100 times, recording the weights at 1,2,3,4,5,10,15,20,40,60,80,100, and 200 epochs, for each run. The weights were normalized and filtered using the process outlined in Section 4 with Filtration(200, 30%) and then PCA (2 component) was applied to the normalized and filtered weights. Using this data, and the Scikit Learn and Giotto TDA packages, the persistence diagrams and Mapper diagrams were constructed for each of the epoch sets.

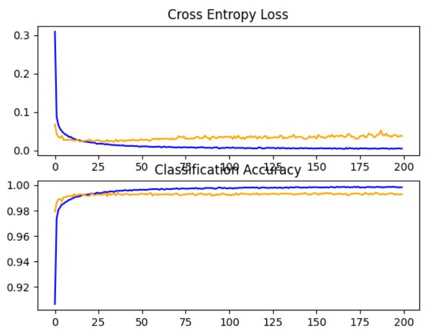


Figure 6.3: *Plot of the Classification Accuracy and Cross-Entropy Loss for the 2-Layer CNN using the MNIST test (yellow) and training (blue) data sets*

In their paper Gabrielsson and Carlson use the same CNN for 100 runs of 40,000 batch iterations with a batch size of 128 on a 60,000 element data set (which turns out to be around 85 epochs). With the weights from the first layers of these runs they found a cycle with a persistence of around 0.6 which is precisely what we found after 60 epochs and just under what we found after 100 epochs (0.65). Given that the results match there is now additional confidence in the results of Gabrielsson and Carlson and more importantly assurance in the legitimacy of the code that we developed and will use for the remainder of the analysis outlined in this paper.

Below are the results from the 2-layer CNN. The first row contains the persistence diagrams at each of the labeled epochs and the second row contains the Mapper diagram. Note that in the Mapper diagram there is information about each point and the connections it has but there is no geometric data describing the distance between points, thus the plotting software for the Mapper diagram takes a best guess and therefore circles may be squeezed and appear more like a line when zoomed out or twisted into a pretzel. The persistence diagram, on the other hand, will indicate the presence of a cycle regardless of how the diagram is plotted.

**Layer 1**

From the analysis of the first layer a cycle (which turns out to be the Primary Circle) appears almost immediately (by the first epoch) and continues to grow in its persistence until between the 5th and 10th epoch where it reaches a peak persistence of approximately 3.5. After this, it starts to decay, although only gradually. Even with this decay there is still a strong cycle all the way through the 200th epoch. The persistence severely wains starting between the 40th and 60th epochs.

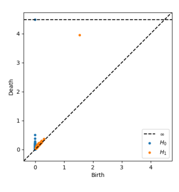
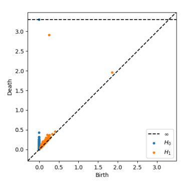
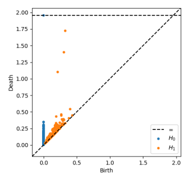
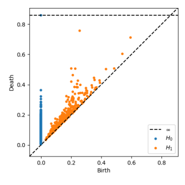
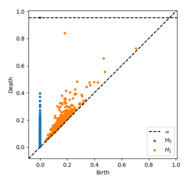
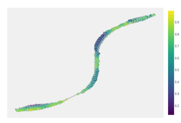
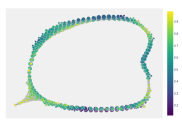
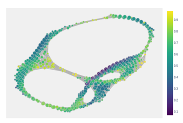
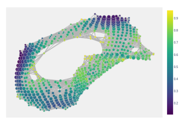
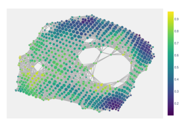
| 1 Epochs | 10 Epochs | 20 Epochs | 60 Epochs | 100 Epochs |
|---|---|---|---|---|
|  |  |  |  |  |
|  |  |  |  |  |

Table 6.2: *Top: Persistence Diagrams for the weights from the 1st layer of the 2-Layer MNIST Bottom: Mapper Diagrams for the weights from the 1st layer of the 2-Layer MNIST*

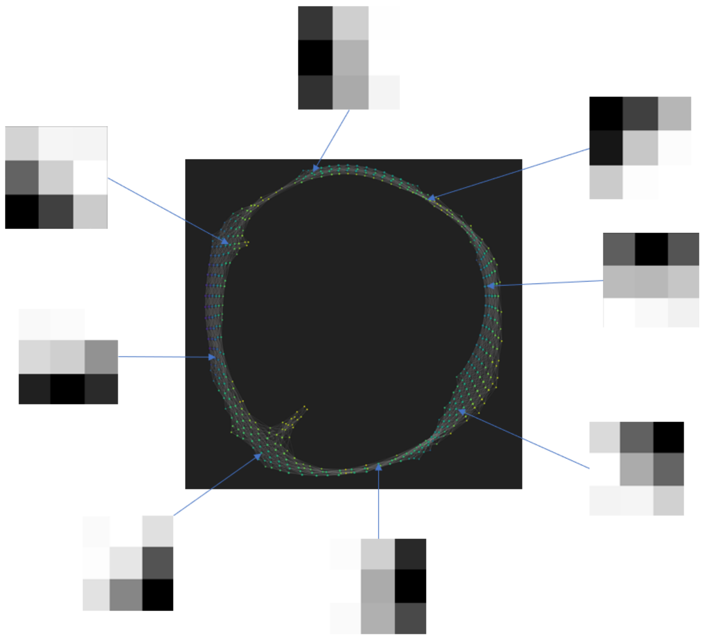


Figure 6.4: *Illustration of the Primary Circle in the Mapper diagram from the Layer 1 Analysis after 10 Epochs*

**Layer 2**
The weights of the second layer were also considered using Filtration(200, 10%) and even better results were obtained than in the first layer. A cycle once again appears after the first epoch with persistence of around 2, and continues to grow until 20 epochs where it peaks near 3 and then decays. However, it still remains very strong (persistence greater than 1.5) at 200 epochs.

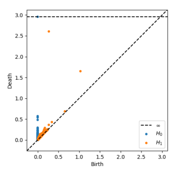
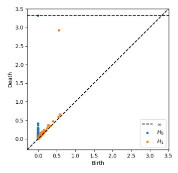
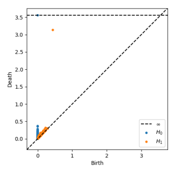
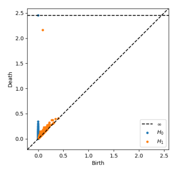
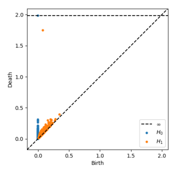
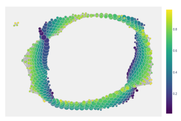
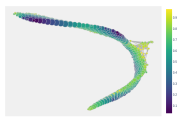
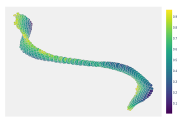
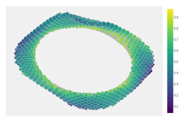
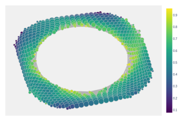
| 1 Epochs | 10 Epochs | 20 Epochs | 60 Epochs | 100 Epochs |
|---|---|---|---|---|
|  |  |  |  |  |
|  |  |  |  |  |

Table 6.3: *Top: Persistence Diagrams for the weights from the 2nd layer of the 2-Layer MNIST with filtration(200, 10%) Bottom: Mapper Diagrams for the weights from the 2nd layer of the 2-Layer MNIST with filtration(200, 10%)*

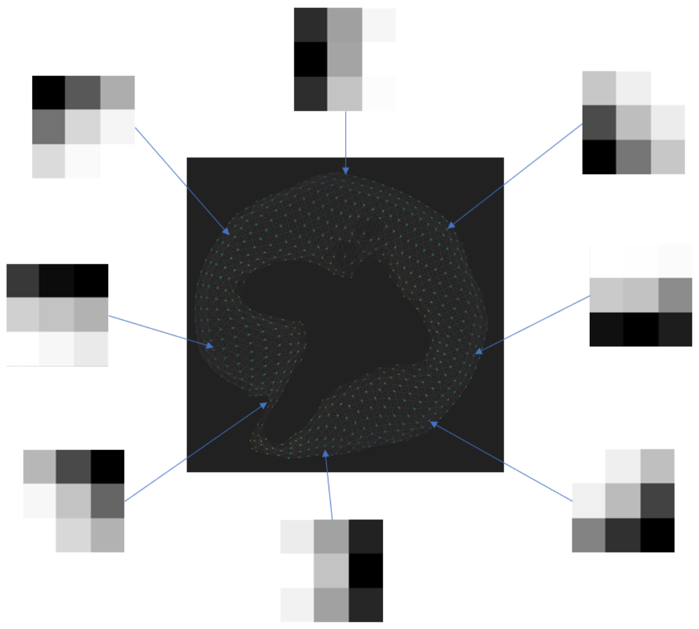


Figure 6.5: *Illustration of the Primary Circle in the Mapper diagram from the Layer 2 Analysis after 20 Epochs*

**Layer 2 - Filtration(10,10%)**
In their paper, Gabrielsson and Carlson state that they analyzed the second layer and found only a very weak Primary Circle (”significantly weaker than that found in the first layer” [18]). However, Gabrielsson and Carlson use a very strong density filtration, Filtration(10, 10%). We ran this same density filtration on the second layer and found similar results, a very weak Primary Circle does appear and appears much weaker than in the first layer or the second layer with the same weaker density filtration that was used on the first layer. (See below for the comparison.) While it is not clear why Gabrielsson and Carlson used a different filtration it is likely that this was done in order to handle the much larger number of points and subsequent computational load required when using the second layer (layer 1 has 64*100 = 6400, 9-dimensional weight vectors while the second layer has 32*64*100 = 204800 9-dimensional weight vectors).

In using a stronger filtration, the data becomes over filtered and no longer represents the larger data structure underneath the data but rather represents only certain local patches of the data. However, if too weak a filtration is used, then the structure also becomes hidden as there are enough noisy points to fill the space causing the cycle to appear only weakly. Finding the correct filtration requires a number of different runs controlling for each parameter of the filtration, ($k$ and $\rho$).

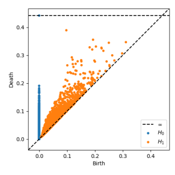
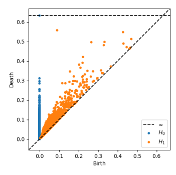
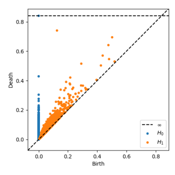
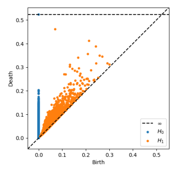
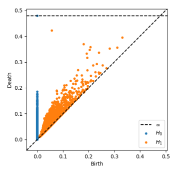
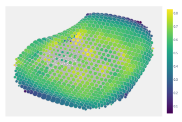
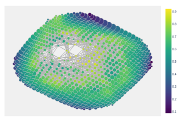
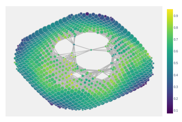
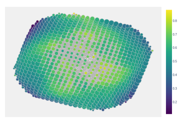
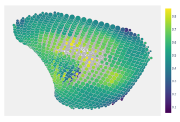
| 1 Epochs | 10 Epochs | 20 Epochs | 60 Epochs | 100 Epochs |
|---|---|---|---|---|
|  |  |  |  |  |
|  |  |  |  |  |

Table 6.4: *Top: Persistence Diagrams for the weights from the 2nd layer of the 2-Layer MNIST with filtration(10, 10%) Bottom: Mapper Diagrams for the weights from the 2nd layer of the 2-Layer MNIST with filtration(10, 10%)*

## 6.4 Additional MNIST Results

In order to further test the C-G Hypothesis on the MNIST data, a single convolutional layer model and a 3 convolutional layer model were trained and analyzed in the same way as the 2-layer network. The goal of this was to verify that the structure found in the weights was not a feature of 2-layer networks, but rather is a property of the ambient space which will be learned by any reasonably well performing CNN (acc $> 90\%$), in this case the architecture should only matter insofar as it leads to a reasonably well performing CNN.

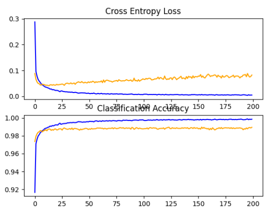
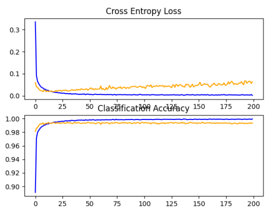
| 1-Layer CNN | 3-Layer CNN |
|---|---|
|  |  |

Table 6.5: *Classification Accuracy and Cross-Entropy Loss for the 1-Layer and 3-Layer CNNs*

### 6.4.1 One Layer CNN

For the single Convolutional Layer network (for architecture see Table 6.6) the weights were analyzed using the same method as the 2-layer CNN with Filtration(200, 0.3%). Using this, a very strong cycle (persistence around 3) was found in the first epoch and increased until peaking (around 4) in the 5th epoch and then the persistence decreased slowly through each epoch. This is not surprising as the first layer had significant overfitting (Table 6.5) which started around the 5th epoch. The fact that overfitting reduces the persistence (i.e., overall strength) of the $H_1$ cycle in the data lends further credence to the C-G Hypothesis, as one would expect that if the $H_1$ cycle is a true generic property of the ambient space, then as the model overfits and learns the specific examples rather than the space, the properties of the ambient space would weaken.

| Layer | Type | Input Dim. | Channels | Activation |
|---|---|---|---|---|
| Layer 1 | Convolutional | 28x28 | 64 | ReLU |
| Layer 2 | Max Pooling | 14x14 | 64 | n/a |
| Layer 3 | Fully Connected | 7x7 | 64 | ReLU |
| Layer 4 | Flatten | | 3136 | n/a |
| Layer 5 | Dropout (50%) | 3136 | 3136 | n/a |
| Layer 6 | Fully Connected | 3136 | 10 | softmax |

Table 6.6: *Architecture of the 1-Layer MNIST CNN*

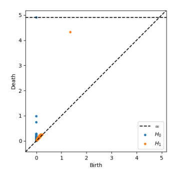
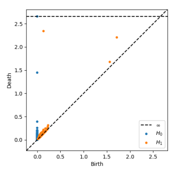
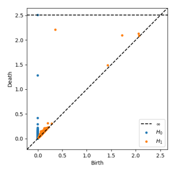
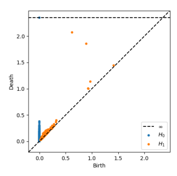
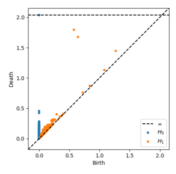
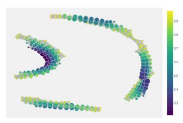
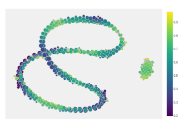
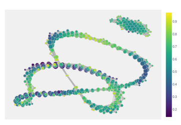
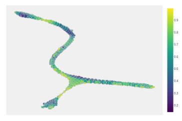
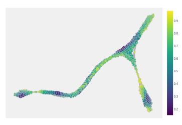
| 1 Epochs | 10 Epochs | 20 Epochs | 60 Epochs | 100 Epochs |
|---|---|---|---|---|
|  |  |  |  |  |
|  |  |  |  |  |

Table 6.7: *Top: Persistence Diagrams for the weights from the 1st layer of the 1-Layer MNIST CNN Bottom: Mapper Diagrams for the weights from the 1st layer of the 1-Layer MNIST CNN*

### 6.4.2 Three Layer CNN

For the three layer CNN (for architecture see Table 6.8), in layer 1 a clear cycle appears in epoch 1 (persistence 3) and then decreases through each epoch becoming very weak to non-existent around epoch 40. In layer 2, a clear cycle appears in epoch 1 (persistence 2.3) and then slowly decreases reaching persistence of 0.7 by epoch 200. In layer 3 no cycle was found. A number of different filtrations were used in order to determine whether this was due to filtration, but in all cases no cycle appeared. For further discussion on this see Section 8.1.2.

| Layer | Type | Input Dim. | Channels | Activation |
|---|---|---|---|---|
| Layer 1 | Convolutional | 28x28 | 64 | ReLU |
| Layer 2 | Max Pooling | 14x14 | 64 | n/a |
| Layer 3 | Convolutional | 14x14 | 32 | ReLU |
| Layer 4 | Max Pooling | 7x7 | 32 | n/a |
| Layer 5 | Convolutional | 7x7 | 64 | ReLU |
| Layer 6 | Max Pooling | 4x4 | 64 | n/a |
| Layer 7 | Fully Connected | 4x4 | 64 | ReLU |
| Layer 8 | Flatten | | 3136 | n/a |
| Layer 9 | Dropout (50%) | 3136 | 3136 | n/a |
| Layer 10 | Fully Connected | 3136 | 10 | softmax |

Table 6.8: *Architecture of the 3-Layer MNIST CNN*

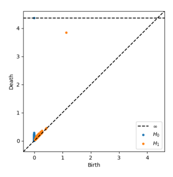
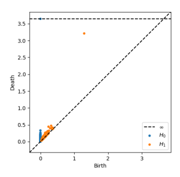
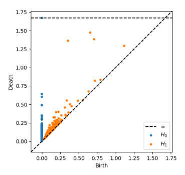
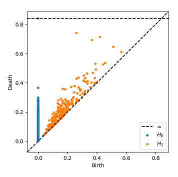
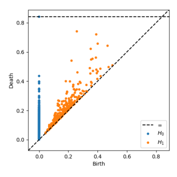
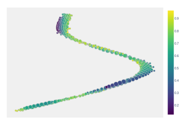
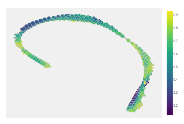
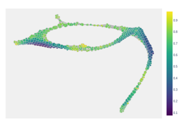
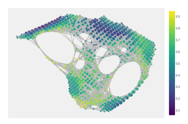
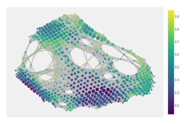
| 1 Epochs | 10 Epochs | 20 Epochs | 60 Epochs | 100 Epochs |
|---|---|---|---|---|
|  |  |  |  |  |
|  |  |  |  |  |

Table 6.9: *Top: Persistence Diagrams for the weights from the 1st layer of the 3-Layer MNIST CNN Bottom: Mapper Diagrams for the weights from the 1st layer of the 3-Layer MNIST CNN*

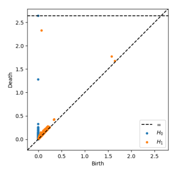
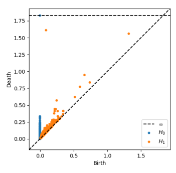
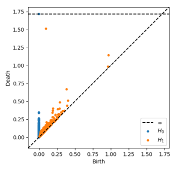
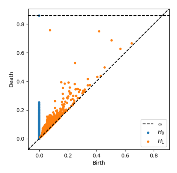
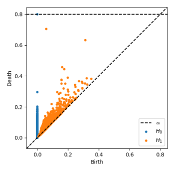
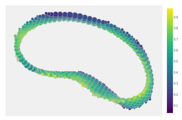
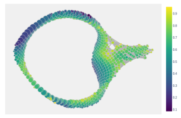
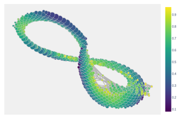
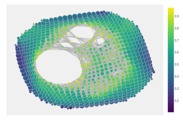
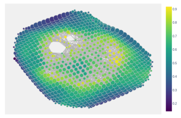
| 1 Epochs | 10 Epochs | 20 Epochs | 60 Epochs | 100 Epochs |
|---|---|---|---|---|
|  |  |  |  |  |
|  |  |  |  |  |

Table 6.10: *Top: Persistence Diagrams for the weights from the 2nd layer of the 3-Layer MNIST CNN Bottom: Mapper Diagrams for the weights from the 2nd layer of the 3-Layer MNIST CNN*

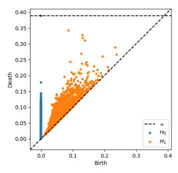
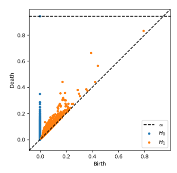
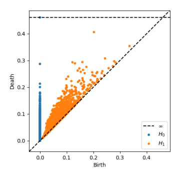
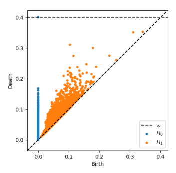
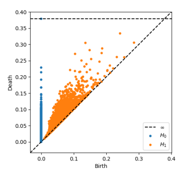
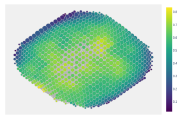
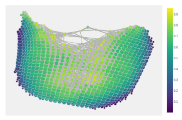
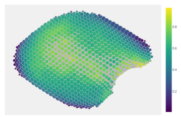
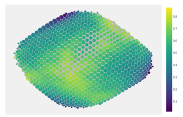
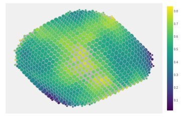
| 1 Epochs | 10 Epochs | 20 Epochs | 60 Epochs | 100 Epochs |
|---|---|---|---|---|
|  |  |  |  |  |
|  |  |  |  |  |

Table 6.11: *Top: Persistence Diagrams for the weights from the 3rd layer of the 3-Layer MNIST CNN Bottom: Mapper Diagrams for the weights from the 3rd layer of the 3-Layer MNIST CNN*

## 6.5 MNIST Conclusion

The results from the work presented in this section corroborate the results found by Carlsson and Gabrielsson. In the 2-Layer CNN a clear high persistence $H_1$ cycle can be found in the persistence diagrams for the 1st Layer and a clear circle can be seen in the Mapper diagram for the 1st Layer. Additionally, we were able to show why the cycle in the 2nd layer weights is not very clear in the results of Carlsson and Gabrielsson, and by altering the filtration parameters show that there is a clear high persistence $H_1$ cycle in the 2nd layer. Building on this work we also showed that there is a clear high persistence $H_1$ cycle in the weights of a 1-layer CNN and in the 1st and 2nd layer weights of a 3-layer CNN. We also showed that there is no cycle in the 3rd layer of a 3-layer CNN. For further discussion on these results see Section 8.

## 7 Dogs vs Cats

After verifying the results found by Carlsson and Gabrielsson and ensuring that our analysis methods function correctly, it is time to move to the bulk of our work (both in content and originality). As previously stated, the goal of our research is to further the application of CNNs by studying the structure of weight spaces in CNNs built to classify a previously unstudied complex image set known as Dogs vs. Cats. In this section, we will discuss the Dogs vs Cats dataset and why I chose it, the CNN architecture and methods used to determine it, and then the results of the statistical and TDA analysis of the convolutional weights.

The Dogs vs. Cats data set comes from a 2013 Kaggle competition [34] used to test the robustness of Asirra (Animal Species Image Recognition for Restricting Access) [11] which is a form of CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart) developed by Microsoft in conjunction with petfinder.com. The Kaggle competition was intended to test whether Asirra was safe from attack and "benchmark the latest computer vision and deep learning approaches" [34].

Figure 7.1: *Some examples of images from the DC data set*

Dogs vs Cats (referred to from here on as DC) was chosen for a couple of reasons. First of all, it is a more complicated data set than MNIST. Even though there are fewer classes in the DC data set, the images presented are much more complicated and the differences between cats and dogs are much more nuanced than handwritten digits. For instance, a "1" digit follows a very clear and defined pattern, in fact most people could describe the pattern of a digit very clearly, and that becomes even easier when asked to describe the difference between two digits say a one and a seven; however, when asked to describe the pattern of a cat (versus a dog) this becomes much more difficult and nearly impossible to do. While we all implicitly can discern the difference between a cat and a dog, actually enumerating this is an incredibly difficult task. MNIST images are only 28x28 pixels whereas DC is 200x200 pixels. While this is still not the size of a full resolution image (typically 1280x720 up to 3840x2160 for 4K) it is more representative and allows features to be spread out into multiple pixels instead of the compression that happens on a 28x28 pixel image. On the flip side, using an image that is 200x200 (rather than 3840x2160) greatly reduces the number of data points as the runtime for training. For research purposes, this is important as it allows more iterations to be conducted in order to perform thorough research and allows the use of standard computing resources without the need for intense processing power. Although, even with 200x200 pixel images, a computing cluster was still needed to perform the TDA and to achieve reasonable run times. DC images are also in color as opposed to MNIST which is gray-scale.

The other reason DC was chosen is that it is an easily available data set with many examples of high performing CNN architectures. While it is not difficult to design a CNN with high accuracy on a simple data set like MNIST, once the data set becomes complex it can become difficult to determine an architecture which will perform with high accuracy on the data. One of the issues with CNNs, and with ANNs in general, is that there is no solid science for how to design the network. While there are rules of thumb, most of architecture design is trial and error until an architecture is found that produces the desired results. Since the design of a CNN architecture is beyond the scope of this work, this data set allowed more time to be spent on analysis of the weights of the network rather than its design. That being said, the author did try several different configurations as outlined below. This was both to find an architecture which achieved good accuracy and also to present some different architectures which could be tested to see if the expected structure appears in different architectures analogous to the results from the MNIST data and corresponding CNNs. Unfortunately, as is described below, only two architectures were found with reasonably high accuracy. This is a potential avenue for future work.

The goal was to analyze this more complicated data set to determine what the structures of the weight spaces are, and from that, gain a better understanding of CNN weight spaces. In so doing, the work will lead to a better understanding of CNNS, offer an avenue for improvements in speed and accuracy of CNN training, and give further confidence in the robustness of the CNN model. Additionally, together with the work of Carlsson and Gabrielsson, this work will go a long way in both validating the C-G hypothesis and also in helping define and demonstrate the underlying geometry of CNNs.

Specifically, the goal was to see if the CNN weights approximate a manifold with non-trivial geometry especially a non-empty first homology group. This will - along with the MNIST results - add credence to the hypothesis in [18] that this structure appears in a more general sense for CNNs built on all image data sets, i.e., the C-G Hypothesis. If successful, this will be the first time someone has shown the presence of this non-trivial geometry with a cycle in $H_1$, as well as the presence of the Primary Circle, on the DC data set or the larger superset of images from Asirra.

In order to determine the underlying topological structure of the weight space, persistence diagrams were used to discover the high persistence features (i.e., high persistence Betti numbers of $H_0$ and $H_1$) that arise as the accuracy of the CNN increases and then Mapper was utilized to determine the structure of the weights. My work was restricted to the first two homology groups as larger groups become increasingly resource intensive to compute and according to the work done by Mumford, Carlsson, et. al the structure of local 3x3 image patches and similarly CNN weights using a 3x3 filter can be primarily described using a 2-dimensional manifold, thus much can still be understood by looking solely at $H_0$ and $H_1$.

Additionally, if the C-G hypothesis holds, a high persistence $H_1$ cycle should appear in the persistence diagrams as the accuracy increases, and when examined the Mapper diagram should contain a clear circle. The nodes of this circle, when traced back, should contain filter values that appear as a rotating transition from dark to light. What this shows is that a CNN trained on the DC data set learns the Primary Circle as the underlying structure of its weights, that is, the weights are an approximation of the Primary Circle which is (at least in part) a feature of the ambient space from which the optimal classifier is a part. This ambient space then is an analog for the space from which the design for the neurons in the human visual cortex comes.

## 7.1 Dogs vs Cats CNN Architecture

To analyze the Dogs vs Cats data several different CNN models were run. Since this data set is more complicated it was necessary to experiment with different CNN configurations to determine which performed the best and to make sure that the CNN achieved reasonable accuracy without overfitting. In order to analyze the CNNs with this data set each CNN was run for 50 epochs to determine their accuracies. Each of these CNNs were created using the Python Keras package from TensorFlow [35] and trained on the Google Colab cloud server [24]. Below is shown the configurations of the CNNs which were tested.

### 1-Layer

| Layer | Type | Input Dim. | Channels | Activation |
|---|---|---|---|---|
| Layer 1 | Convolutional | 200x200 | 32 | ReLU |
| Layer 2 | Max Pooling | 100x100 | 32 | n/a |
| Layer 3 | Flatten |  | 320000 | n/a |
| Layer 4 | Fully Connected |  | 128 | ReLU |
| Layer 5 | Fully Connected |  | 1 | Standard Logistic |

Table 7.1: *Architecture of the 1 Convolutional Layer CNN*

### 2-Layer

| Layer | Type | Input Dim. | Channels | Activation |
|---|---|---|---|---|
| Layer 1 | Convolutional | 200x200 | 32 | ReLU |
| Layer 2 | Max Pooling | 100x100 | 32 | n/a |
| Layer 3 | Convolutional | 100x100 | 64 | ReLU |
| Layer 4 | Max Pooling | 50x50 | 64 | n/a |
| Layer 5 | Flatten |  | 160000 | n/a |
| Layer 6 | Fully Connected |  | 128 | ReLU |
| Layer 7 | Fully Connected |  | 1 | Standard Logistic |

Table 7.2: *Architecture of the 2 Convolutional Layer CNN*

### 3-Layer

| Layer | Type | Input Dim. | Channels | Activation |
|---|---|---|---|---|
| Layer 1 | Convolutional | 200x200 | 32 | ReLU |
| Layer 2 | Max Pooling | 100x100 | 32 | n/a |
| Layer 3 | Convolutional | 100x100 | 64 | ReLU |
| Layer 4 | Max Pooling | 50x50 | 64 | n/a |
| Layer 5 | Convolutional | 50x50 | 128 | ReLU |
| Layer 6 | Max Pooling | 25x25 | 128 | n/a |
| Layer 7 | Flatten |  | 80000 | n/a |
| Layer 8 | Fully Connected |  | 128 | ReLU |
| Layer 9 | Fully Connected |  | 1 | Standard Logistic |

Table 7.3: *Architecture of the 3 Convolutional Layer CNN*

**3-Layer with Dropout**

| Layer | Type | Input Dim. | Channels | Activation |
|---|---|---|---|---|
| Layer 1 | Convolutional | 200x200 | 32 | ReLU |
| Layer 2 | Max Pooling | 100x100 | 32 | n/a |
| Layer 3 | Dropout (20%) | 100x100 | 32 | ReLU |
| Layer 4 | Convolutional | 100x100 | 64 | ReLU |
| Layer 5 | Max Pooling | 50x50 | 64 | n/a |
| Layer 6 | Dropout (20%) | 50x50 | 64 | ReLU |
| Layer 7 | Convolutional | 50x50 | 128 | ReLU |
| Layer 8 | Max Pooling | 25x25 | 128 | n/a |
| Layer 9 | Dropout (20%) | 25x25 | 128 | ReLU |
| Layer 10 | Flatten | | 80000 | n/a |
| Layer 11 | Fully Connected | | 128 | ReLU |
| Layer 12 | Dropout | | 128 | ReLU |
| Layer 13 | Fully Connected | | 1 | Standard Logistic |

Table 7.4: *Architecture of the 3 Convolutional Layer with Dropout CNN*

The following accuracies were achieved after 50 Epochs:
1 Block Model Acc = 73%
2 Block Model Acc = 74%
3 Block Model Acc = 80%
3 Block Model with Dropout = 81%

Given these results, the 3 block model with dropout was chosen because it performed the best. This was then used to perform 100 training runs saving the weights after 1,2,3,4,5,10,15,20,40, and 60 epochs. However, after investigating the results further it was revealed that not only did the model achieve less than desired accuracy it also suffered from overfitting. Thus, it was necessary to remove the problem of overfitting before analysis of the weights could be completed.

## 7.2 Solving the Overfitting Problem

The first attempt to solve the problem of overfitting was to use an optimizer. In the previous MNIST example the Adam Optimizer was used and is generally known to produce good results so it was chosen. For more information on overfitting see Section 3.7

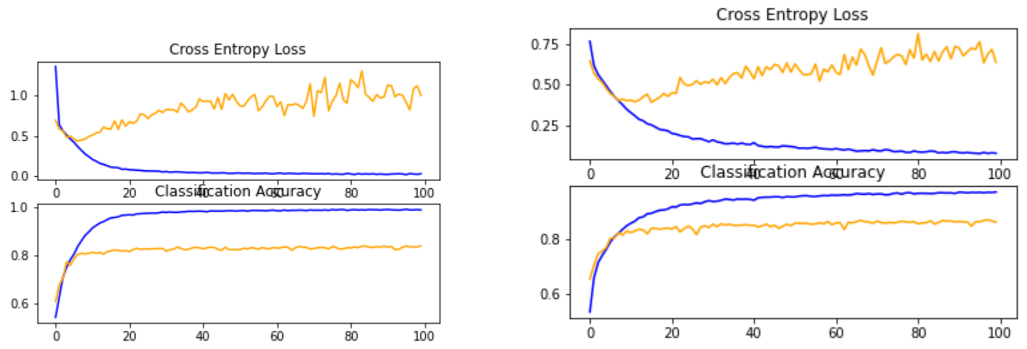


Figure 7.2: *Left: 3-Layer Network with 128 Node 3rd Layer and Adam Optimizer, Right: 3-Layer Network with 32 Node 3rd Layer and Adam Optimizer*

As can be seen from the results in Figure 7.2, this did not fix the problem of overfitting so a different method to counteract this was sought. Additional methods tried were, Dropout after each layer, L1 Regularization, and L2 Regularization. Ultimately, none of these methods were successful in removing the problem of overfitting. As such an alternate method known as Data Augmentation was used.

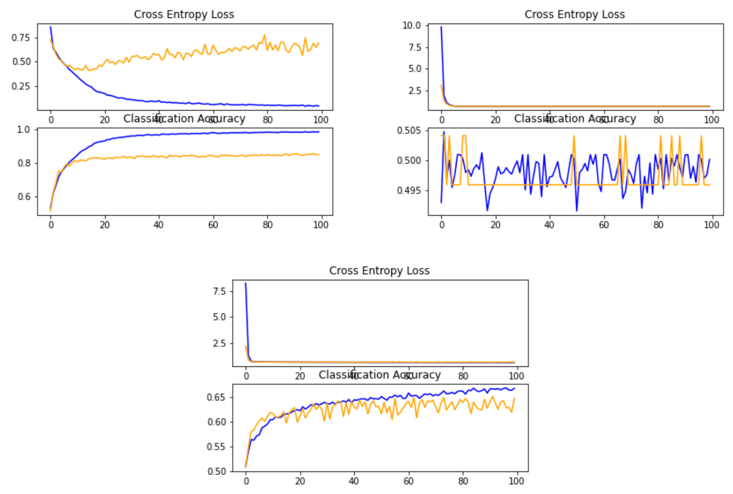


Figure 7.3: *Left: Dropout 0.4, Center: L1 Regularization, Right: L2 Regularization*

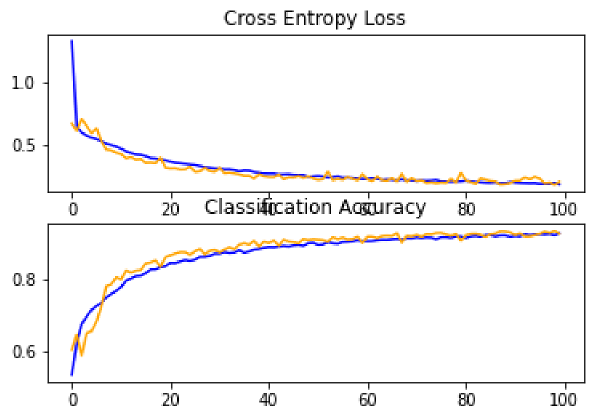


Figure 7.4: *3-Layer CNN with Dropout and Data Augmentation*

Using Data Augmentation we were able to achieve the nearly identical results on both the training and test data set which demonstrated that the CNN was not overfitting. For the rest of the paper, the results used were achieved using this 3-Layer CNN with data augmentation (unless otherwise stated). Table 7.5 lists the architecture for each of the CNNs considered when trying to solve the overfitting problem.

| | 1st Layer | 2nd Layer | 3rd Layer | Layer Dropout | FC Layer Dropout | Regular-ization | Data Aug |
|---|---|---|---|---|---|---|---|
| Adam | 32 | 64 | 128 | 0.2 | 0.5 | None | None |
| Adam 32 | 32 | 64 | 32 | 0.2 | 0.5 | None | None |
| Dropout 0.4 | 32 | 64 | 128 | 0.4 | 0.5 | None | None |
| L1 Regularization | 32 | 64 | 128 | 0.2 | 0.5 | L1 | None |
| L2 Regularization | 32 | 64 | 128 | 0.2 | 0.5 | L2 | None |
| Data Augmentation | 32 | 64 | 128 | 0.2 | 0.5 | None | Yes |

Table 7.5: *Architectures for the models used to test solutions to the problem of overfitting*

### 7.2.1 Training

Once the problem of overfitting had been dealt with the 3-Layer CNN with dropout and data augmentation was trained 50 times for 60 epochs. From the prior overfitting analysis, it was shown that the CNN is capable of achieving around 94% accuracy by epoch 100. While training out to 100 or 200 epochs would have been preferred for completeness and for the accuracy, training for 60 epochs even with GPU acceleration took six to eight hours per run, so training for 200 epochs would have taken nearly twenty hours per run which would have made the training last for nearly 3 months. Instead, it was decided that since the CNN achieved 90% accuracy by 60 epochs (90% is a typical benchmark for ”good” performance in a Neural Network) that 60 epochs would be enough to provide reasonable results and allow for the analysis of the structure of the weight data for the Dogs vs. Cats data set.

For each of the 50 runs the weights for each of the three convolutional layers and the model accuracy were recorded at epochs 1, 2, 3, 4, 5, 10, 15, 20, 40, and 60 epochs. This results in 3*32*50 = 4800 layer 1 weights (3 color channels * 32 filter channels * 50 runs), 32*64*50 = 102,400 layer 2 weights (32 channels from the prior layer * 64 filter channels * 50 runs), and 128*64*50 = 409,600 layer 3 weights (128 channels from the prior layer * 64 filter channels * 50 runs). After training for 60 epochs the mean training accuracy achieved was around 90%.

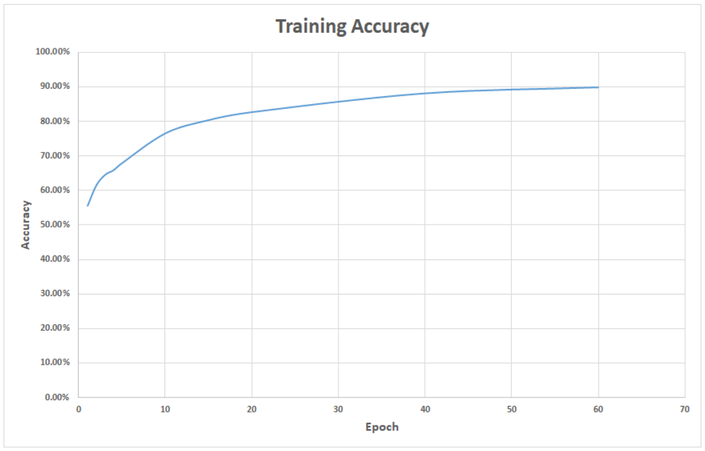


Figure 7.5: *Mean training accuracy across the 50 training runs for the 3-Layer with dropout CNN on the Dogs vs Cats dataset*

## 7.3 Statistical Analysis

The analysis we performed on the Dogs vs Cats CNNs is split into two different methods. First, we performed a more classical statistical analysis of the data. This is in keeping with the work performed by Mumford et. al. [47] on the set of high variance high density image patches. The goal was to use traditional statistical methods to show that the weights from the convolutional layers of the 3-Layer CNN model are highly non-normal and thus demonstrate that there must be some interesting shape to the data before moving on to examine the data using TDA. In doing both of these methods, we were able to demonstrate that the results from TDA analysis are in line with the results of traditional statistical analysis and further show how TDA is able to provide more information and a better description of the structure of the data. The statistical analysis methods included the computation of Voronoi Cells and their density and the calculation of the KL Distance for the CNN convolutional weights.

### 7.3.1 Kullback-Leibler Distance - Theory

The Kullback-Leibler Distance or Kullback-Leibler Divergence, is a commonly used statistical measure for determining how two probability distributions $P$ and $Q$ differ over the same variable $x$ and is denoted $D_{KL}(P(x)|Q(x))$. The Kullback-Leibler (KL) Distance was originally introduced by Solomon Kullback and Richard Leibler [38]. Given two probability distributions $P(x)$ and $Q(x)$ of a discrete random variable $x$ It is defined as:

$$D_{KL}(P(x)|Q(x)) = \textstyle\sum_{x \in X} Q(x) \log_2 \frac{P(x)}{Q(x)}$$

It is often referred to as the KL Divergence instead of KL Distance because it is not actually a metric due to its lack of symmetry, i.e., $D_{KL}(P(x)|Q(x)) = m$ does not imply $D_{KL}(Q(x)|P(x)) = m$. It should also be noted that when computing the KL Distance, it is necessary to only use samples for which $Q$ is defined.

The KL Distance is closely related to the idea of entropy (see Section 3.6) from the field of information theory.

From an information theory perspective, the KL Distance then measures the information lost when using $Q(x)$ to model $P(x)$ or more explicitly, the expected number of bits required to code samples from $P(x)$ when using $Q(x)$.

From a statistical perspective, the Neyman-Pearson Lemma [49] states that the best way to distinguish between two distributions is through their log likelihood ratio, i.e., $\log(P(x)) - \log(Q(x))$. The KL Distance then measures the expected value of the log likelihood ratio, i.e., $D_{KL}(P(x)|Q(x)) = E[\log(P(x)) - \log(Q(x))]$.

While it has many applications, most notably using samples to determine if a distribution matches a theoretical model, it is often used to determine whether a distribution is Gaussian or Uniform. One only need define a Gaussian or Uniform distribution over the same set of samples and then calculate the KL Distance. In our case, we can use it to show from a statistical perspective that the distribution of convolutional weights from a trained CNN are not Gaussian.

### 7.3.2 Voronoi Analysis

Using Voronoi Cell analysis, the density of the points on the 7-sphere ($S^7$) can be determined and used to show that the weights are localized to some sub set of $S^7$. (Recall that the normalization process of the point cloud set $X \subseteq \mathbb{R}^9$ projects the data on $S^7$.) The statistical distribution of the points illustrates their relative density on $S^7$. This shows whether the points are distributed in a Gaussian distribution or arranged in some more sparse manner. Unlike TDA however, it does not give any indication of the shape of the points.

To perform this analysis, we followed the methods used in [47] to analyze high-variance high-density natural image patches. The first step is to define the set of Voronoi Cells. Since the data lies on $S^7$ after it is normalized, then to define the set of Voronoi Cells, a set of evenly distributed points are needed on $S^7$ with a small enough distance between the points to accurately capture the density of the data. Mumford et. al. pointed out that this is analogous to the "kissing number" problem in $\mathbb{R}^8$, i.e., how to arrange a maximum number of non-intersecting spheres of radius 1 so that they all touch the unit sphere, $S^7$. This problem is itself non-trivial [7], but fortunately a solution does exist for $\mathbb{R}^8$ given by the $E_8$ lattice.

From [7], the first spherical shell is the unique solution to this problem. In [47] the authors use the 4th spherical shell without explanation. It is assumed that this is to have a higher fidelity analysis of the points since the first spherical shell only has 240 points while the 4th has 17520, but it is not explained in the paper. For the sake of comparison to the results from Mumford et. al. on natural images, we also chose the 4th spherical shell as the set of Voronoi points. This results in the following set of points:

1. The 112 permutations and sign changes of $< 2, 2, 0, 0, 0, 0, 0, 0 > /\sqrt(8)$
2. The 8960 permutations and sign changes of $< 2, 1, 1, 1, 1, 0, 0, 0 > /\sqrt(8)$
3. The 256 permutations and sign changes of $< 1, 1, 1, 1, 1, 1, 1, 1 > /\sqrt(8)$
4. The 7168 permutations and sign changes of $< \frac{3}{2}, \frac{3}{2}, \frac{3}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2} > /\sqrt(8)$
5. The 1024 permutations and sign changes of $< \frac{5}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2} > /\sqrt(8)$

Using these points, the Voronoi Cell of each weight vector is determined by: $\Omega_i = \{x \in X | dist(x, P_i) \leq dist(x, P_j) \text{ for } P_j \in P\}$ where $P$ is the set of Voronoi Points defined above and $X$ is the set of weight vectors.

One will note that although lying on $S^7$, the weight vectors are in $\mathbb{R}^9$ while the Voronoi Points are in $\mathbb{R}^8$. In order to compute the distance, we must either transform the Voronoi Points into $\mathbb{R}^9$ or transform the weight vectors into $\mathbb{R}^8$. The latter option was chosen in which case it was necessary to determine an orthonormal basis for $\mathbb{R}^8$ which could be used to project the weight vectors into $S^7$. The Gram-Schmidt Orthogonalization Process was used to find the following orthogonal basis.

Since the weight vectors lie on $S^7$, an 8-dimensional subspace of $\mathbb{R}^9$, we can define an 8-dimensional basis for the weight vectors $X$. However, in order to project the points in $X$ to $\mathbb{R}^8$ we need an orthogonal basis. To do this we start with our basis $V = \{v_1, v_2, ...v_8\}$ and then use the Gram-Schmidt Orthogonalization Process to find an 8-dimensional orthogonal basis.

The Gram-Schmidt Orthogonalization Process is as follows:
Given a basis $V = \{v_1, v_2, ...v_p\}$ for a subspace $W$ of $\mathbb{R}^n$, one can construct an orthogonal basis $\{b_1, b_2, ...b_p\}$ of $W$ such that for any $i = 1, ..., p$, $span\{v_1, v_2, ...v_i\} = span\{b_1, b_2, ...b_i\}$ by the following steps:

$b_1 = v_1$
$b_2 = v_2 - Proj_{b_1}(v_2) = v_2 - \frac{v_2 b_1^t}{b_1 b_1^t} b_1$
$b_3 = v_3 - Proj_{span\{b_1,b_2\}}(v_3) = v_3 - \frac{v_3 b_1^t}{b_1 b_1^t} b_1 - \frac{v_3 b_2^t}{b_2 b_2^t} b_2$
$b_p = v_p - Proj_{span\{b_1,...b_{p-1}\}}(v_p) = v_p - \frac{v_p b_1^t}{b_1 b_1^t} b_1 - \frac{v_p b_2^t}{b_2 b_2^t} b_2 - ... - \frac{v_p b_{p-1}^t}{b_{p-1} b_{p-1}^t} b_{p-1}$

Starting with a basis for $\mathbb{R}^8 \subseteq \mathbb{R}^9$ we have the following:

$b_1 =< 1, 0, 0, 0, 0, 0, 0, 0, -1 >$
$b_2 =< 0, 1, 0, 0, 0, 0, 0, 0, -1 >$
$b_3 =< 0, 0, 1, 0, 0, 0, 0, 0, -1 >$
$b_4 =< 0, 0, 0, 1, 0, 0, 0, 0, -1 >$
$b_5 =< 0, 0, 0, 0, 1, 0, 0, 0, -1 >$
$b_6 =< 0, 0, 0, 0, 0, 1, 0, 0, -1 >$
$b_7 =< 0, 0, 0, 0, 0, 0, 1, 0, -1 >$
$b_8 =< 0, 0, 0, 0, 0, 0, 0, 1, -1 >$

Performing the Gram-Schmidt Orthogonalization Process results in the following orthogonal basis:

$b_1 =< 1, 0, 0, 0, 0, 0, 0, 0, -1 >$
$b_2 =< \frac{1}{2}, 1, 0, 0, 0, 0, 0, 0, -\frac{1}{2} >$
$b_3 =< \frac{1}{3}, \frac{1}{3}, 1, 0, 0, 0, 0, 0, -\frac{1}{3} >$
$b_4 =< \frac{1}{4}, \frac{1}{4}, \frac{1}{4}, 1, 0, 0, 0, 0, -\frac{1}{4} >$
$b_5 =< \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, 1, 0, 0, 0, -\frac{1}{5} >$
$b_6 =< \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, 1, 0, 0, -\frac{1}{6} >$
$b_7 =< \frac{1}{7}, \frac{1}{7}, \frac{1}{7}, \frac{1}{7}, \frac{1}{7}, \frac{1}{7}, 1, 0, -\frac{1}{7} >$
$b_8 =< \frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}, 1, -\frac{1}{8} >$

Once normalized this set provides an orthonormal basis for $\mathbb{R}^9$. By normalizing we are able to use $B = \begin{bmatrix} \tilde{b}_1 \\ \tilde{b}_2 \\ \tilde{b}_3 \\ \tilde{b}_4 \\ \tilde{b}_5 \\ \tilde{b}_6 \\ \tilde{b}_7 \\ \tilde{b}_8 \end{bmatrix}$ where $\tilde{b}_i = b_i / \|b_i\|$

to project our weight vectors $x \in X$ where $Bx^t$ projects to 8-dimensions while preserving the position on $S^7$, i.e., norm of 1.

We now use this to define our distance function $dist(x, P_i) = \arccos(Bx^t \cdot P_i)$ for $x \in X$ and $P_i \in P$ the set of Voronoi Points defined above.

With this distance function and the membership function $\Omega_i(x)$ defined above, the points $X$ can be binned into their respective Voronoi Cells.

To perform the analysis, the weights from each of the 3-layers were taken after 60 epochs (Layer 1 = 4800 points, Layer 2 = 102400 points, Layer 3 = 409600 points). The points were then normalized using the same method as applied during the TDA analysis. Similarly, a K-Nearest Neighbors Density Filtration was applied using $\rho = 30\%$ and $k = 100$. Lastly, the points were then divided by 3 (since unit variance meant that the points all had a norm of 3 not 1). Using the filtered points (Layer 1 = 1440 points, Layer 2 = 30720 points, Layer 3 = 122880 points) the points were then binned and the percentage of bins containing at least one point was calculated. The results are show below:

Percentage of Voronoi Cells Containing at least one point in Layer 1 = 6.67%
Percentage of Voronoi Cells Containing at least one point in Layer 2 = 20.10%
Percentage of Voronoi Cells Containing at least one point in Layer 3 = 40.56%

In order to compare to a normal distribution, a set of random Gaussian 9-dimensional vectors were generated using a mean of 0 and a standard deviation of 1. Using 4800, 102400, and 409600 random points, to compare to layers 1, 2, and 3 respectively the same analysis (normalization and filtration) was performed resulting in the following:

Percentage of Voronoi Cells Containing at least one point with 4800 (filtered to 1440) Random Normal Points = 7.51%
Percentage of Voronoi Cells Containing at least one point with 102400 (filtered to 30720) Random Normal Points = 67.60%
Percentage of Voronoi Cells Containing at least one point with 409600 (filtered to 122880) Random Normal Points = 96.30%

Additionally, the Kullback-Leibler Distance (Divergence) was used to measure the deviation of our data from a uniform distribution. We can compare to a uniform distribution here because a Gaussian distribution is uniform after whitening which has already been done via the scaling and normalization process.[47].

We define the probability distribution function for the data points as:

$p(\Omega_i) = \frac{N(\Omega_i)}{\sum_i N(\Omega_i)}$ where $N(\Omega_i)$ is the number of points in the $i$-th Voronoi Cell.

Then the uniform distribution is defined as:

$q_u = \frac{vol(\Omega_i)}{vol(S^7)}$ where $vol(\Omega_i)$ is the volume of the $i$-th Voronoi Cell.

From [47] the volume of the Voronoi Cells are $6.3*10^{-3}, 1.8*10^{-3}, 4.1*10^{-3}, 1.8*10^{-3}, 1.8*10^{-3}$ for the 5 types of Voronoi Cells defined above respectively. The volume of $S^7$ is $\pi^4/3$.

The KL Divergence is then:

$$D_{KL}(p|q_u)) = \sum_i p(\Omega_i) \log_2 \frac{p(\Omega_i)}{q_u(\Omega_i)}$$

This results in the following:

KL Distance of Layer 1 = 3.72
KL Distance of 4800 (filtered to 1440) Random Points = 3.40

KL Distance of Layer 2 = 2.89
KL Distance of 102400 (filtered to 30720) Random Points = 0.49

KL Distance of Layer 3 = 1.93
KL Distance of 409600 (filtered to 122880) Random Points = -0.03

From this analysis it can be seen that the weight vectors are highly non-uniform meaning that the original non-whitened vectors are highly non-Gaussian. The Gaussian points have a high KL Distance in the first layer because there are only a small number of points which are also density filtered making them no longer normally distributed. Even still, it can be seen that the filtered weight vectors are much more densely packed than the Gaussian points. From a statistical perspective this further demonstrates that the weight vectors are contained in some kind of non-normal subset of the state space ($\mathbb{R}^9$).

## 7.4 TDA Analysis

### 7.4.1 Hyper-parameter Analysis

Moving on to the TDA analysis the first step was to determine the optimal hyper-parameters which will be used. For the MNIST analysis the filtration, numbers of cubes, and overlap were determined by the values used in [18], although, as discussed, different filtration values were studied to achieve better results. For the Dogs vs. Cats analysis, it was necessary to perform a set of analysis runs to determine the optimal values for each of the hyper-parameters (i.e., number of cubes, overlap, $\rho$, and k-value).

The number of cubes and overlap are hyper-parameters provided to the Mapper function to define the cover used by the Mapper function. Recall that Mapper first uses a function $f : X \to \mathbb{R}^d$ to map the data into $\mathbb{R}^d$ for some $d$ (typically $\mathbb{R}^2$). Then a cover $\mathcal{U}$ is used to cover the image of $f$. The Python Kepplermapper package used in this analysis defines a cover using a set number of cubes (squares since it is $\mathbb{R}^2$) $N$ and a percentage of overlap $\beta$. The cover is then defined by splitting the image of $f$ into $N$ equally sized hypercubes with $\beta\%$ overlap.

The filtration of the data, as described above, is performed by the K-Nearest Neighbors Density Filtration. The hyper-parameters for this filtration are two values, the percentage of filtration - $\rho$ and $k$, that is, the $k$-th distance to be used by the filtration. $\rho\%$ refers to the number of points to keep. That is, after the k nearest neighbor is found for each point only the $\lceil \rho * .01 * N \rceil$ closest points are kept (where $N$ is the number of points in the data set). The $k$ value used in the KNN Density Filtration refers to the $k$ value in the $k$ nearest neighbors. That is, the value used to compare is the distance between the point and the $k$-th closest point. So, if $k$ is 10 then for each point $x$ the value used to compare to the other values is the distance to the 10-th closest point to $x$.

**Filtration Parameters**
In order to test the filtration parameters, the 40 epochs set of weights was chosen as it was known to have a clear cycle and thus would make it easy to compare the results of the different filtration parameter combinations. Using this, runs were made with $\rho \in \{0.01, 0.02, 0.03, 0.04, 0.05, 0.1, 0.2\}$ and $k \in \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 150, 200, 250, 300, 350, 400, 450, 500\}$. During prior experimentation $\rho = 0.3,\ k = 200$ were used so by testing less than $\rho = 0.3$ we could evaluate if it performed better as $\rho$ grew smaller and if not then a different set of runs would be made using larger values for $\rho$. As can be seen below, smaller $\rho$ values resulted in better results thus it was not necessary to test with larger values. As shown in figures 7.6-7.8 the persistence increases as $k$ grows, but only until around 100 to 200. The results from these runs for each of the 3 layers are shown in figures 7.6-7.8 where the persistence

of the primary cycle is graphed against k value for each of the $\rho$ values tested. There are only results for a few k-values used in conjunction with $\rho = 0.1$ and 0.2 in Layer 3. This is because there are enough points in these filtrations (0.1*409600 = 40960 and 0.2*409601 = 81,920) that the computing cluster used did not have enough memory to handle the processing, so few results for these values could be obtained.

As can be seen below, the persistence increased for smaller values of $\rho$ and also increased for larger values of $k$. However, the increase in persistence for larger values of $k$ tails off near $k = 100$. Thus $k = 100$ and $\rho = 0.05$ were chosen for the analysis of the 3-layer CNN for Dogs vs Cats.

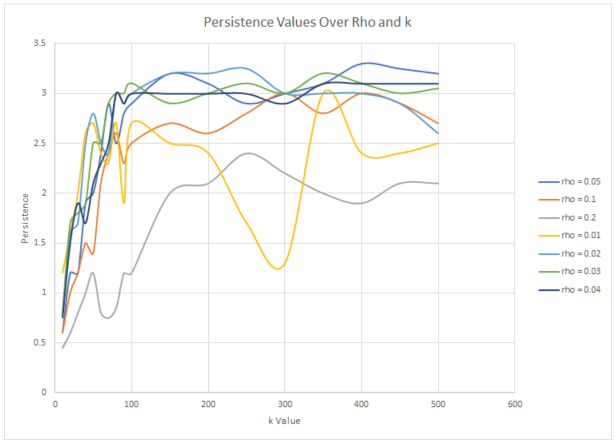


Figure 7.6: *Layer 1 Persistence Over Different $\rho$ and $k$ values*

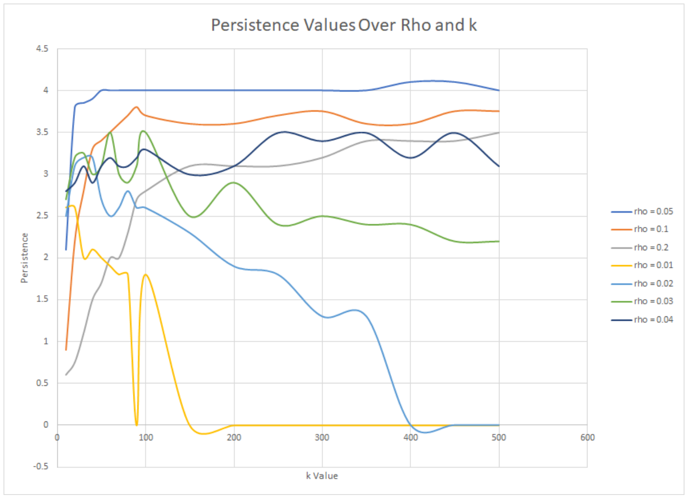


Figure 7.7: *Layer 2 Persistence Over Different $\rho$ and $k$ values*

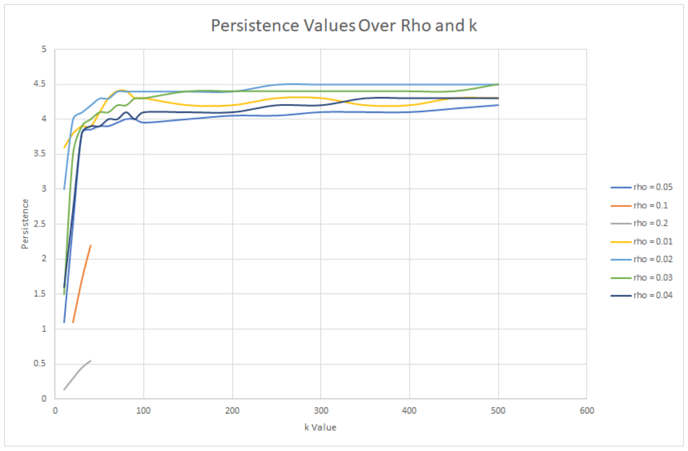


Figure 7.8: *Layer 3 Persistence Over Different $\rho$ and $k$ values*

**Mapper Hyper-parameters**

In order to test the Mapper hyper-parameters (cubes and overlap) the 40 epochs set of weights was chosen as it was known to have a clear cycle and thus would make it easy to compare the results of the different Mapper hyper-parameter combinations. Then runs were made using cubes $\in \{10, 20, 30, 40, 50, 60\}$ and overlap $\in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$.

Using these values, it was discovered that the best results appeared for 20 and 30 cubes in Layer 1 with 40 cubes looking reasonable but not as good as 20 or 30, Layer 2 had good results with 20, 30, and 40 cubes with 20 looking the best, and Layer 3 had good results in 20, 30, and 40 with reasonable results in 50 cubes but the best results were with 40 cubes. Given this, 30 cubes was chosen as the optimal parameter given that it had the best results across all three layers and the desire was to use the same parameters for each layer where possible.

For the overlap, with 30 cubes, the best results were achieved using an overlap of 0.6 and 0.7. As such, 0.66 was chosen as it is between these values and was the value used in the prior analysis for MNIST. For Layer 1, 0.7 % was used as the overlap because the circle in the Mapper diagram was not always closed for 0.66 % and 0.6 % overlap. This is different than the results above because the analysis used to generate those pictures used $k = 200$ and $\rho = 0.1$, whereas the Mapper hyper-parameters test used $k = 100$ and $\rho = 0.05$. The results for the 30 cubes runs with $\rho = 0.6$ and 0.7 are shown in Table 7.6. The full set of cubes vs overlap runs are shown in Appendix B.

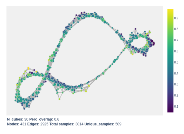
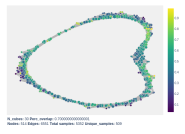
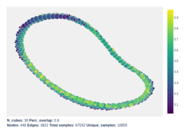
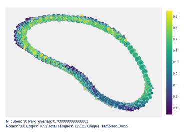
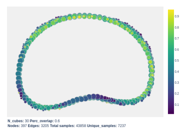
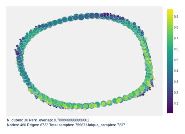
|  | Overlap 0.6 | Overlap 0.7 |
|---|---|---|
| Layer 1 - 30 Cubes |  |  |
| Layer 2 - 30 Cubes |  |  |
| Layer 3 - 30 Cubes |  |  |

Table 7.6: *Mapper diagrams for layers 1,2, and 3 of the 3-Layer CNN with Dropout performed at 40 epochs with varying overlap.*

### 7.4.2 Results

After training for 50 runs, the weights for each of the three convolutional layers were analyzed using the same process as with the MNIST data set. The analysis was done on the University of Missouri Science and Technology cluster "The Foundry" [28] using the Python Scikit Learn and Giotto TDA packages (the code can be found in the GitHub repository [61]). As described above, different hyper-parameters were chosen in order to determine the optimal results. The only variation is that for Layer 3 epochs 40 and 60, 1/2 of the data points were chosen (so only the first 25 runs were used) while the other half were ignored. This was done because the function used to perform the persistence diagram measurements required more computing resources than were available even on "The Foundry" computing cluster. The parameters chosen are summarized in Table 7.7.

| Epochs | k Value | $\rho$ | number of cubes | overlap | Mapper Lens |
|---|---|---|---|---|---|
| {1,2,3,4,5,10, 15,20,40,60} | 100 | 0.05 | 30 | 66% (70% Layer 1) | 2 Component PCA |

Table 7.7: *Parameters used for the analysis of the 3-Layer CNN with Dropout*

Using these hyper parameter values the following results were found:

**Layer 1**

In layer 1, starting with the persistence diagrams, a clear cycle (i.e., element of $H_1$) does not begin to appear until epoch 20 where a cycle is found with persistence of 1. A clear cycle is found with increasing persistence in epochs 40 and 60 as well with a persistence value of 2.7 in epoch 40 and 3.2 in epoch 60. By looking at the Mapper diagrams, the Primary Circle is unclear at epochs 20 and 40 but becomes clear in epoch 60. It is reasonable that a persistent cycle will be clear in the persistence diagram but no cycle will appear in the Mapper diagram due to the fact that the Mapper diagram is created at a single threshold value on the data and thus if this value is not chosen correctly will result in no cycle being present. This is further complicated by the choice of clustering for the pullback of the cover in the Mapper algorithm, see Section 4.2.1. The implementation of the Mapper algorithm in Python (KepplerMapper) does not allow for user choice of the threshold parameter for building the simplicial complex.

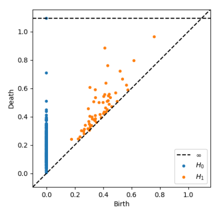
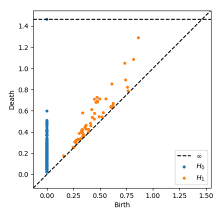
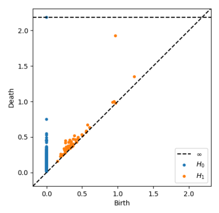
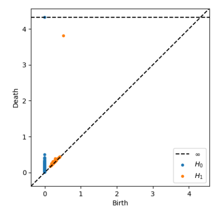
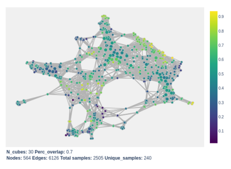
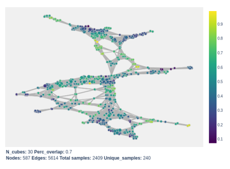
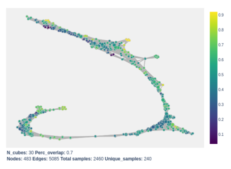
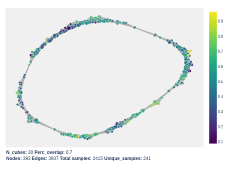
| 1 Epochs | 10 Epochs | 20 Epochs | 60 Epochs |
|---|---|---|---|
|  |  |  |  |
|  |  |  |  |

Table 7.8: *Top: Persistence Diagrams for the weights from the 1st layer of the 3-Layer CNN with Dropout and Data Augmentation Bottom: Mapper Diagrams for the weights from the 1st layer of the 3-Layer CNN with Dropout and Data Augmentation*

In any case it is clear to see that by epoch 60 an obvious persistent cycle appears in the weight data as is shown by the persistence diagram and that the same cycle appears in the Mapper diagram.

Looking at the Mapper diagram the nodes were traced back to their original weight values with the mean 3x3 weight patch plotted. The following function was used to compute the mean of the nodes.

Let $N_i = \{W_0, W_1, ...W_{m-1}\}$ be a node in the Mapper diagram with each weight vector $W_j = \{x_0, x_1, ...x_8\} \in \mathbb{R}^9$, then:

$$Mean(N_i)_j = \sum_{k=0}^{m} \frac{x_{kj}}{|N_i|} \text{ where } x_{kj} \text{ is the } j\text{-th element of } W_k \in N_i$$

The mean of each node was then plotted where $x_0$ is the upper right-hand component and the color is a greyscale value with higher values associated with lighter colors. Examining the mean of the nodes in the Mapper diagram in this way revealed that they were arranged in the Primary Circle pattern that was expected, i.e., rotating gradients around the circle. (see Figure 7.9).

It should be noted here that the exact layout of the Mapper diagram in Table 7.8 and Figure 7.9 are different because they are drawn by two different plotting mechanisms. In the table, the Mapper diagrams are produced using an external plotting function known as Plotly [51]; whereas, the Mapper diagram in Figure 7.9 is drawn using the kmapper [37] native function visualize. In both cases the differences arise from the difficulty in creating a geometric representation of an abstract simplicial complex. In the abstract simplicial complex, we have a set of simplexes which give rise to nodes and edges but there is no geometric information. Thus, different methods of creating a geometric representation can produce different layouts; however, both will have the same nodes and edges. While difficult to tell from the small pictures, in this paper, when inspected closely, the Mapper diagram in Table 7.8 and Figure 7.9 are in fact the same nodes and edges just with different layouts. From a topological perspective this is not an issue, as two topological structures are homotopic (i.e., same type or structure) if they are the same save for stretching, contracting, twisting, and translating.

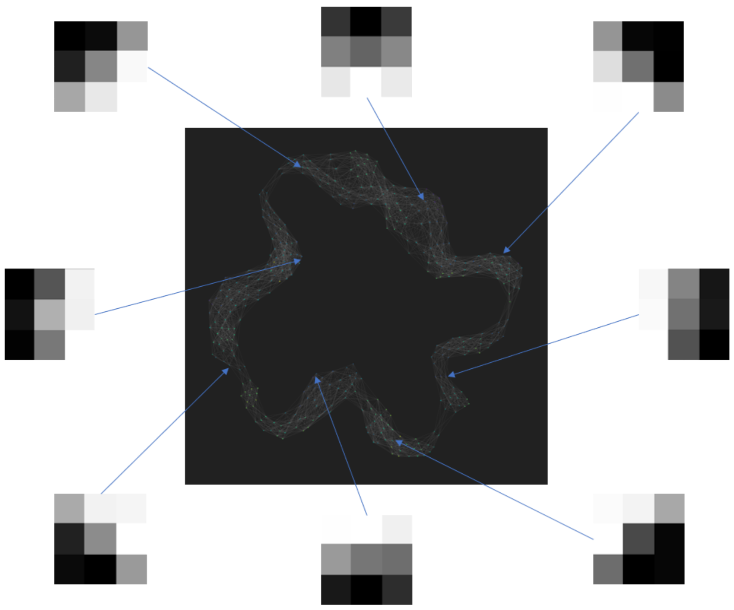


Figure 7.9: *Mapper Diagram from Layer 1 of the 3-Layer CNN with Dropout at 60 Epochs, with images of the means of nodes around the cycle showing the presence of the Primary Circle*

**Layer 2**

In layer 2, starting with the persistence diagrams, a clear cycle first appears in epoch 15 where a cycle is found with a persistence of 1.5. A clear cycle is found with increasing persistence in epochs 20, 40, and 60 as well with persistence values of 2.9 in epoch 20 and 4.1 in epoch 40 and 4.2 in epoch 60. By looking at the Mapper diagrams, the Primary Circle is quite clear in epochs 15, 20, 40, and 60. It is easier for the Mapper algorithm to detect a cycle in layers 2 and 3 because there are substantially more points which makes the cycle smoother and less likely to have jumps or missing sections due to a sparseness of data. In Layer 1 there are only 4800 points to start with which is reduced to 0.05*4800 = 240 points after the filtration. Whereas there are 102400 points in layer 2 which are reduced to 5120 points after the filtration. Images for the Layer 2 results are pictured in Table 7.9.

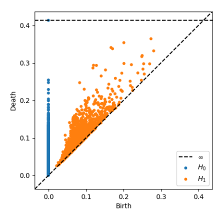
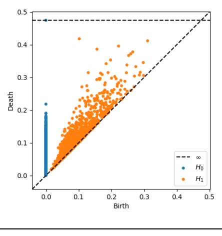
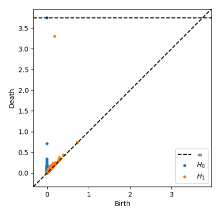
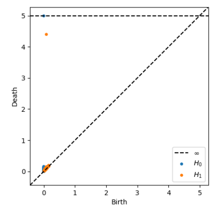
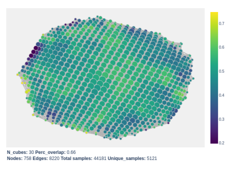
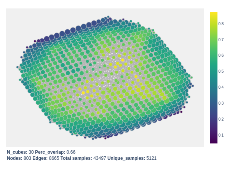
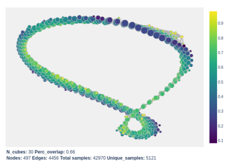
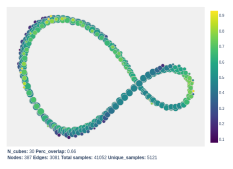
| 1 Epochs | 10 Epochs | 20 Epochs | 60 Epochs |
|---|---|---|---|
|  |  |  |  |
|  |  |  |  |

Table 7.9: *Top: Persistence Diagrams for the weights from the 2nd layer of the 3-Layer CNN with Dropout and Data Augmentation Bottom: Mapper Diagrams for the weights from the 2nd layer of the 3-Layer CNN with Dropout and Data Augmentation*

Looking further at the Mapper diagram from epoch 60 we can trace the nodes back to their original points using the same method that was used for Layer 1. Examining the mean of the nodes in the Mapper diagram in this way revealed that they were arranged in the Primary Circle pattern that was expected, i.e., rotating gradients around the circle. (see Figure 7.10).

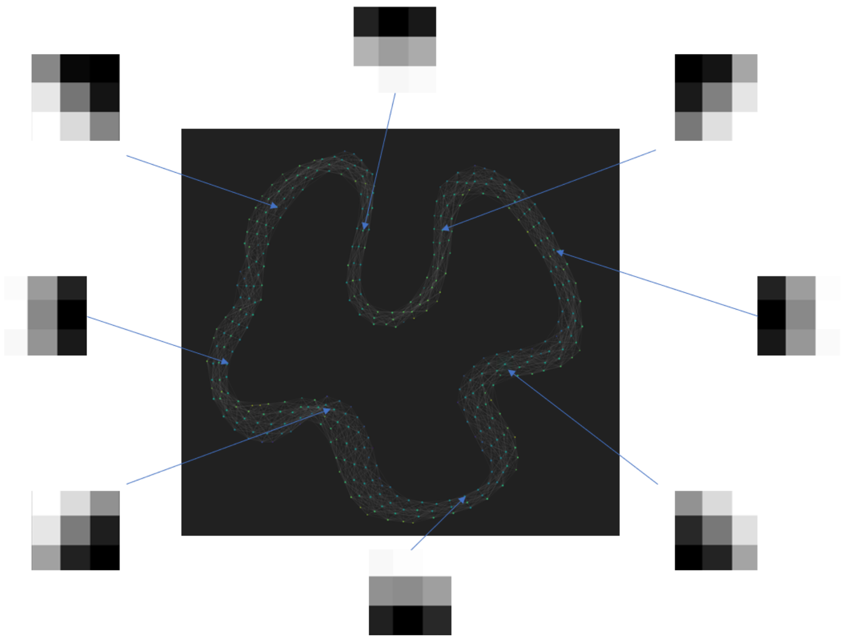


Figure 7.10: *Mapper Diagram from Layer 1 of the 3-Layer CNN with Dropout at 60 Epochs, with images of the means of nodes around the cycle showing the presence of the Primary Circle*

**Layer 3**

In layer 3, starting with the persistence diagrams, there is potentially a cycle appearing at epoch 20 but the persistence is too low to draw any conclusions and is made more unclear by the presence of another cycle of lower persistence. The assumption is that at this point Layer 3 has not correctly learned the structure of the data. By epoch 40, a single cycle becomes very clear with a persistence of 3.8. By epoch 60, the persistence increases to 4. The Mapper diagrams align with the persistence diagrams. In epoch 20, the Mapper diagram does not reveal a clear cycle although it can be noticed that one is forming. In epochs 40 and 60 a single cycle is very clear in the Mapper diagram. Images for the Layer 3 results are pictured in Table 7.10.

Looking further at the Mapper diagram from epoch 60 we can trace the nodes back to their original points using the same method that was used for Layer 1. Examining the mean of the nodes in the Mapper diagram in this way revealed that they were arranged in the Primary Circle pattern that was expected, i.e., rotating gradients around the circle. (see Figure 7.11).

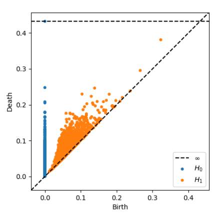
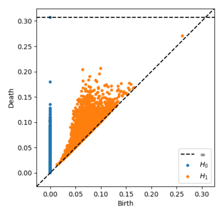
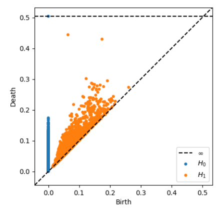
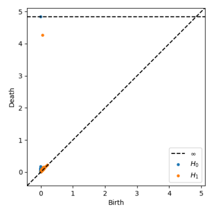
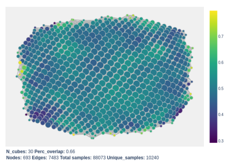
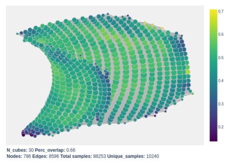
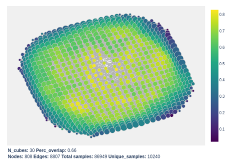
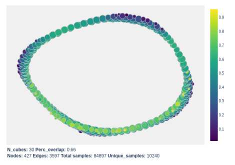
| 1 Epochs | 10 Epochs | 20 Epochs | 60 Epochs |
|---|---|---|---|
|  |  |  |  |
|  |  |  |  |

Table 7.10: *Top: Persistence Diagrams for the weights from the 3rd layer of the 3-Layer CNN with Dropout and Data Augmentation Bottom: Mapper Diagrams for the weights from the 3rd layer of the 3-Layer CNN with Dropout and Data Augmentation*

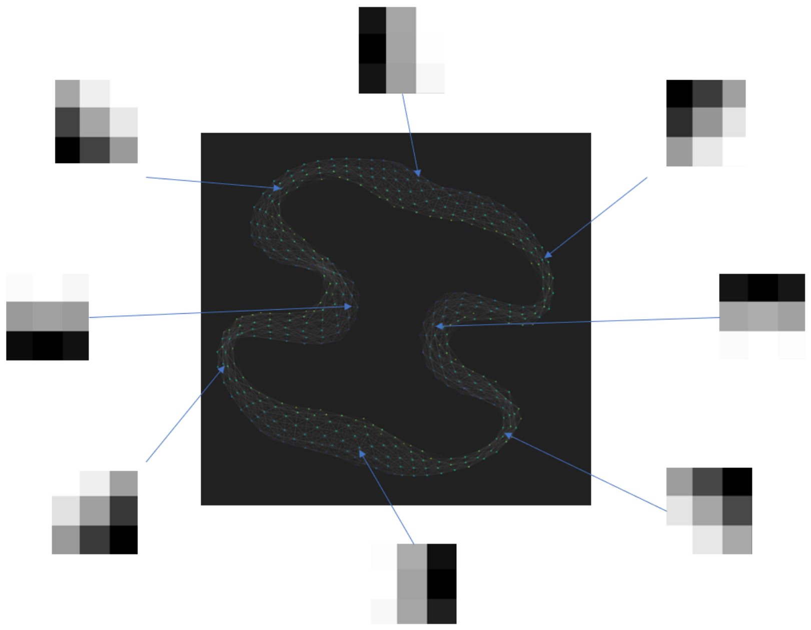


Figure 7.11: *Mapper Diagram from Layer 3 of the 3-Layer CNN with Dropout at 60 Epochs, with images of the means of nodes around the cycle showing the presence of the Primary Circle*

**Overfit Model**

We also attempted to analyze the first layer weights of the 3-Layer Model with dropout (but without data augmentation) referenced in Section 7.1 to see if the weights from this overfit model with not ideal accuracy had the same structure seen in the 3-Layer Model with Dropout and Data Augmentation analyzed above.

The Layer 1 weights were normalized using the same methods as above followed by Filtration(200, 30%) and then PCA (2 component) was applied to the normalized and filtered weights. Using this data, and the Scikit Learn and Giotto TDA packages, the persistence diagrams and Mapper diagrams were constructed for each of the epoch sets. The summary of this is shown in tables 7.11 and 7.12.

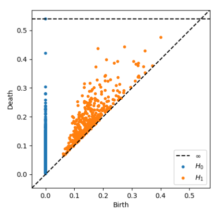
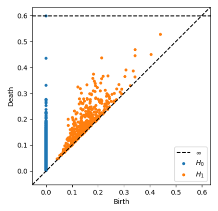
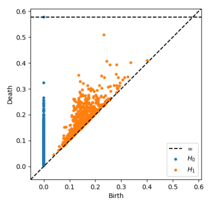
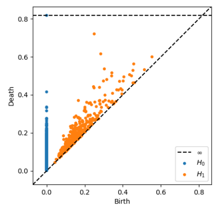
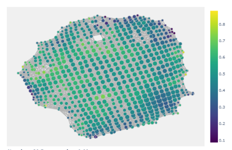
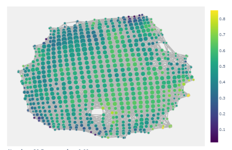
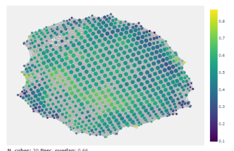
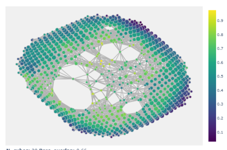
| 1 Epochs | 10 Epochs | 20 Epochs | 60 Epochs |
|---|---|---|---|
|  |  |  |  |
|  N_cubes: 30 Perc_overlap: 0.66<br>Nodes: 727 Edges: 7602 Total samples: 14871 Unique samples: 1738 |  N_cubes: 30 Perc_overlap: 0.66<br>Nodes: 724 Edges: 7566 Total samples: 14541 Unique samples: 1690 |  N_cubes: 30 Perc_overlap: 0.66<br>Nodes: 726 Edges: 7662 Total samples: 14239 Unique samples: 1652 |  N_cubes: 30 Perc_overlap: 0.66<br>Nodes: 785 Edges: 7803 Total samples: 13407 Unique samples: 1565 |

Table 7.11: *TDA Diagrams for epochs 1, 10, 20, and 60 of training the 3 Convolutional Layer with Dropout CNN*

Around 60 epochs a cycle did start to appear but it was very faint and the expected Primary Circle was very unclear in the Mapper diagram. As a result, we trained the model out to 80 and 100 epochs to see if more training caused the cycle to appear more clearly. While the cycle did become clearer in epochs 80 and 100 the persistence of the $H_1$ cycle is far lower than that found in the Layer 1 weights of the 3-Layer Model with Data Augmentation that did not overfit. In the overfit model, the cycle achieved a persistence value of 1.5 in epoch 80 and 2.1 in epoch 100. Further research could be performed to consider the Layer 2 and Layer 3 weights or the weights from epochs greater than 100, but this result was enough to show that overfitting can cause a model to not have as clear of a high persistence cycle as that found in ones without the problem of overfitting.

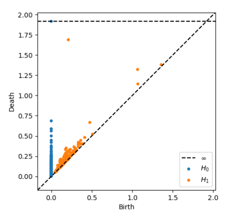
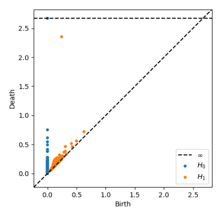
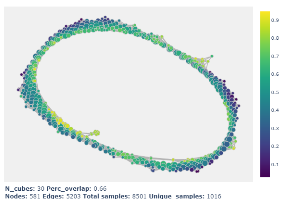
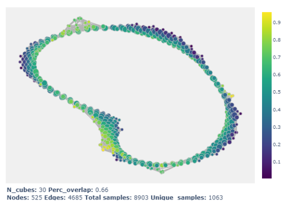
| 80 Epochs | 100 Epochs |
|---|---|
|  |  |
|  |  |

Table 7.12: *TDA Diagrams for epochs 80 and 100 of training the 3 Convolutional Layer with Dropout CNN*

### 7.4.3 4-Layer CNN

To show that the structure of the weight space is the same for different architectures and that it persists through more than just the first 3 layers, a second CNN was built with 4 convolutional layers. Similar to that of the 3-layer CNN with dropout the architecture can be seen in Table 7.13.

| Layer | Type | Input Dim. | Channels | Activation |
|---|---|---|---|---|
| Layer 1 | Convolutional | 200x200 | 32 | ReLU |
| Layer 2 | Max Pooling | 100x100 | 32 | n/a |
| Layer 3 | Dropout (40%) | 100x100 | 32 | ReLU |
| Layer 4 | Convolutional | 100x100 | 64 | ReLU |
| Layer 5 | Max Pooling | 50x50 | 64 | n/a |
| Layer 6 | Dropout (20%) | 50x50 | 64 | ReLU |
| Layer 7 | Convolutional | 50x50 | 128 | ReLU |
| Layer 8 | Max Pooling | 25x25 | 128 | n/a |
| Layer 9 | Dropout (40%) | 25x25 | 128 | ReLU |
| Layer 10 | Convolutional | 25x25 | 64 | ReLU |
| Layer 11 | Max Pooling | 12x12 | 64 | n/a |
| Layer 12 | Dropout (40%) | 12x12 | 64 | ReLU |
| Layer 13 | Flatten | | 9216 | n/a |
| Layer 14 | Fully Connected | | 128 | ReLU |
| Layer 15 | Dropout (50%) | | 128 | ReLU |
| Layer 16 | Fully Connected | | 1 | Standard Logistic |

Table 7.13: *Architecture of the 4 Convolutional Layer CNN*

The CNN was trained with data augmentation for 50 separate runs up to 60 epochs resulting in a mean accuracy across the runs of around 93.6% by epoch 60.

The same TDA methods were applied as with the 3-Layer CNN. The weights from each of the 4 convolutional layers were normalized and filtered using the process outlined in Section 4 with Filtration(100, 20%) on Layer 1 and Filtration(100, 5%) on layers 2,3, and 4, and then PCA (2 component) was applied to the normalized and filtered weights. Using this data, and the Scikit Learn package, the persistence diagrams and Mapper diagrams (cubes 30 and overlap 66% for Layer 1 and 70% for layers 2, 3, and 4) were constructed for each of the epoch sets. The results for epoch 60 are shown in Table 7.14. The full set of results can be found in the GitHub repository [61].

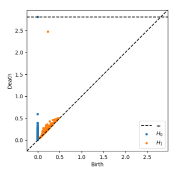
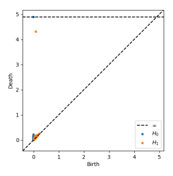
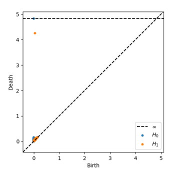
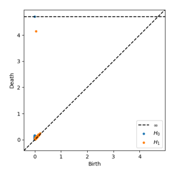
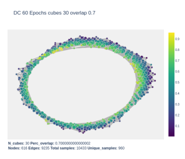
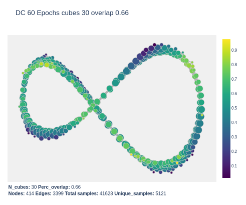
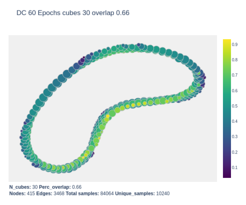
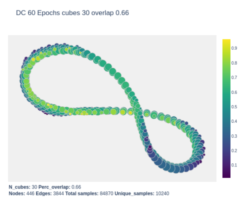
| Layer 1 | Layer 2 | Layer 3 | Layer 4 |
|---|---|---|---|
|  |  |  |  |
|  |  |  |  |

Table 7.14: *Top: Persistence Diagrams for the weights from the convolutional layers of the 4-Layer CNN with Dropout and Data Augmentation after 60 epochs Bottom: Mapper Diagrams for the weights from the convolutional layers of the 4-Layer CNN with Dropout and Data Augmentation after 60 epochs*

In Layer 1, the first clear cycle appears in epoch 40 with a persistence of 1.1, this increases to a persistence of 2.3 in epoch 60. In Layer 2, two cycles appear in epoch 15 one with a persistence of 1.6 and one with a low persistence of 0.7, this second cycle disappears in by epoch 20 and we are left with a single cycle of persistence 3.6, 3.9, and 4.1 in epochs 20, 40, and 60 respectively. In Layer 3, a cycle first appears in epoch 15 with persistence of 1, this increases to 3.2 in epoch 20 and 4 in epochs 40 and 60. In Layer 4, a cycle first appears in epoch 20 with a persistence of 1.3 which increase to 4 and 4.1 in epochs 40 and 60 respectively. Additionally, a very clear circle can be seen in the Mapper diagrams at epoch 60. We did not trace the weights back to verify that this is the Primary Circle (rotating gradients).

### 7.5 Dogs vs Cats Conclusion

From the work outlined in this section, it can be clearly seen that the convolutional weights of CNNs built for and trained on the Dogs vs Cats dataset take on some highly non-normal structure and that the data comes from a space with non-trivial geometry, specifically a space with a non-empty first Homology group. By using Voronoi Cells and the computation of the KL Distance, we were able to show that the weights are highly non-normal. Further, we were able to use TDA to show that the weights must come from some space with 0th and 1st homology groups resembling that of a circle. That is, the weights have a single high persistence cycle in $H_1$. This can be seen in the weights for all three layers of the 3-Layer CNN and all 4 convolutional layers of the 4-Layer CNN. Examining this structure closer also reveals, by use of the Mapper algorithm, that the structure of the weights resembles the Primary Circle. These results shows that the Carlsson-Gabrielsson hypothesis holds for the Dogs vs Cats dataset and thus for more complicated datasets than MNIST and CIFAR-10. For further discussion on these results see Section 8.

## 8 Discussion

Before making our conclusion in Section 9 we devote this section to a discussion in which we illustrate some observations from our results that are of particular interest and discuss some of the results which require further illumination or are unclear.

### 8.1 MNIST Results

#### 8.1.1 Verification of Carlsson Gabrielsson Results [18]

Carlsson and Gabrielsson demonstrated that the weights in CNNs built for three different simple image data sets (MNIST: 28x28 pixel grayscale, CIFAR-10: 30x30 pixel color, SVHN: 32x32 pixel color) have the expected structure to their convolutional weights, finding either a single cycle in $H_1$ or 3 cycles in $H_1$, i.e., similar to the $C^3$ topology discussed in Section 1. This shows that the weights from these CNNs appear to come from a space with a non-empty first homology group.

We were able to verify the presence of a single high persistence $H_1$ cycle in the 1st layer of a CNN built on the MNIST data set with the same architecture as used in [18]. We also found a high persistence $H_1$ cycle in the second layer weights which was not found by Carlsson and Gabrielsson as they used a different filtration. These results corroborate the work of Carlsson and Gabrielsson which was needed as the authors used a proprietary software package to achieve their results and therefore they are not readily available for verification. Some work has been done in [39] to replicate the results found by Carlsson and Gabrielsson, but the author of that work was unable to fully replicate their results. Expanding on the work of Carlsson and Gabrielsson, we were able to show the presence of a high persistence $H_1$ cycle in the second layer of a two convolutional layer CNN built on MNIST and were able to show the presence of a high persistence $H_1$ cycle in the first and second layers of a three convolutional layer CNN built on MNIST.

### 8.1.2 3rd Layer Cycle

In the 3rd layer of the 3-Layer CNN for MNIST no $H_1$ cycle could be detected. Analysis was performed using $k \in \{10, 20, ..., 100\}$ and $\rho \in \{0.05, 0.1, 0.2, 0.3\}$ but no high persistence $H_1$ cycles were found at any epoch. It is not clear why a cycle fails to appear in the 3rd layer of the 3-Layer MNIST CNN. It is hypothesized that the 3-layer CNN learns more specific features of the data set because the first two layers are able to learn the network with very high accuracy and so, as more layers are added, the network begins to learn more abstract or specific features of each image. Because the data set is simple enough, it is able to do this without overfitting. In order to verify that this is a feature of deeper networks built on this simplistic dataset another CNN with 4 convolutional layers was built and analyzed in the same way. This network had the same behavior as the 3-layer CNN in that the first two layers learned high persistence $H_1$ cycles while layers 3 and 4 did not. The results of this analysis can be found in Appendix A.

In contrast, a high persistence $H_1$ cycle is found in layers 3 and 4 of the 3 and 4-layer CNNs built on the Dogs vs Cats dataset and Carlsson and Gabrielsson also found $H_1$ cycles in deeper layers of their analysis of a CNN built on CIFAR-10 and their study of SVHN. Since all of these are more complicated datasets and ones for which a 2-layer CNN cannot achieve greater than 98% accuracy this aligns with the hypothesis of the 3-layer CNN for MNIST.

### 8.1.3 Persistence Evolution Through Training

While the Carlsson Gabrielsson analysis of MNIST CNNs focused only on CNNs at a single epoch, our analysis shows the progression of the weight structure across epochs. From this, an important observation is that, for the MNIST CNN, a cycle with highest persistence is achieved very early on in training, typically around epoch 5 for Layer 1 and epoch 15 for Layer 2. After this, the persistence of the cycle decreases through subsequent training. This seems counter-intuitive as it is expected that the cycle, if a true part of the geometric space from which the images were taken, should remain in the weights as they are trained and so a cycle should be found with increasing or steady persistence. Looking at the training accuracy it can be seen that the CNNs for MNIST achieve optimal accuracy very early in training, typically around the same number of epochs as the highest persistence cycle. After this point, accuracy either does not increase or increases only marginally while the loss function does increase. What this suggests is that the optimal CNN occurs around epoch 15, and then, as training continues, it begins to learn very specific features of each image. Because the MNIST dataset is relatively simple this does not produce gross overfitting which would be apparent if the loss function returned a large value (see Figure 6.3 and Table 6.5), rather, enough specific features of the images are able to be encapsulated by the weights that general structures of the data are not needed for classification. This case is not possible with more complicated data sets (like Dogs vs Cats or even CIFAR-10 as shown in [18]) and so this same result does not appear in the Dogs vs Cats analysis.

## 8.2 Dogs vs Cats

More importantly, the same methods were applied to a CNN built on the Dogs vs Cats dataset. This is a more complex data set with two orders of magnitude more pixels per image than studied by Carlsson and Gabrielsson. We were able to show a high persistence cycle exists and is a feature of the weights for all three convolutional layers of a 3 convolutional layer CNN and all 4 layers of a 4 convolutional layer CNN. This cycle appears with a high persistence in the persistent diagrams and also a circle appears in the Mapper diagrams. Additionally, in the Mapper diagrams, the nodes can be traced back to their original filter values revealing that the cycle appears to be the Primary Circle consisting of rotating gradients.

### 8.2.1 CNNs Learn a Cycle as Epochs Increase

In the Dogs vs Cats CNNs, it can be seen that in all three layers of the 3-Layer model and all 4 layers of the 4-Layer model, the persistence of the cycle in $H_1$ either gets larger as the number of epochs increases or grows to a certain persistence and then holds. Since the training accuracy is monotonically increasing through 60 epochs and the Cross-Entropy Loss matches between the test and training data sets through 60 epochs, this increase of persistence suggests that the cycle is a true structure of not only the data but also the space the weights lie in as the cycle only becomes clearer as the CNN training accuracy increases, i.e., the weights are tuned to better approximate the structure of the data.

As discussed above, this was not apparent in the MNIST CNNs because the MNIST dataset is fairly simple and so a CNN was able to learn the data set accurately in the first few epochs and then would either overfit the data or begin to encapsulate specific features of every image in the dataset rather than the global structure. With Dogs vs Cats, since it is a more complicated dataset, this was not possible. Exactly what features the MNIST CNNs were able to learn and why these better minimized the loss function used for training is not fully understood. Typically, in training a CNN, practitioners will train a CNN only until a "knee in the curve" appears, that is, once the accuracy or loss function levels out. This can prevent the problem of overfitting or specificity to the dataset. There are many open research questions here and a lot of room for further work in investigating what specific features are learned in these cases and why.

### 8.2.2 Cycle Weak In the Presence of Overfitting

As noted in Section 7.1, the first 3-Layer CNN for Dogs vs Cats had a problem with overfitting. In the analysis of this model, a persistent $H_1$ cycle was not found in the first layer until epoch 80 and even in epochs 80 and 100 cycles only appeared with a persistence of 1.5 and 2 respectively. This persistence is much lower than what is achieved in earlier epochs in Layer 1 of the model once the problem of overfitting was removed (persistence of 2.7 and 3.2 in epochs 40 and 60 respectively). This suggests that the true structure of the space in which the weights lie has a non-empty first homology group similar to that of the space of natural images, and thus CNN weights will approximate this structure as they learn the structure of the data. Overfitting means that the weights have learned more specific features of the images in the training data set but not in the test set and so the training set accuracy is high but the test set is not. If the true structure of the data contains a cycle, then this should be approximated better by a CNN without overfitting than one with overfitting as this structure appears in both the training and test data. It can still appear in CNNs with overfitting as it may be a filter that is used by the CNN to learn more specific features. In summary, the fact that the cycle has a higher persistence earlier on in training in the correct model (one without overfitting) gives further confidence in its true presence in the weight space, i.e., the space within which the weights lie.

### 8.2.3 Cycle Exists at Multiple Different Choices of Hyper-parameters

In Section 7.4.1, analysis is given for different choices of hyper-parameters. Specifically, on the 3-Layer CNN for Cats vs Dogs a set of persistence diagrams is generated for epoch 40 of each layer using the filtration parameters $\rho \in \{0.01, 0.02, 0.03, 0.04, 0.05, 0.1, 0.2\}$ and $k \in \{10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 150, 200, 250, 300, 350, 400, 450, 500\}$. With the exception of $\rho = 0.01$ and $\rho = 0.02$ for Layer 2, a high persistence cycle is found at every combination of

$\rho$ and $k$ value. What this suggests is that the cycle is (mostly) independent of choice for filtration values. Of course, if the filtration is chosen too low there will not be enough data to determine the structure of the data, and if there is no filtration then there will be too many noisy points which will make detection of the structure difficult. In general, the cycle appearing at varying filtrations means that this cycle is a true feature of the data and not an artifact of the filtration. The lack of a high persistence cycle at $\rho = 0.01$ and 0.02 is most likely due to these being very fine filtrations which do not leave enough data to adequately determine the structure in Layer 2.

#### 8.2.4 Primary Circle vs $C^3$

In the work of Carlsson and Gabrielsson [5, 18] much of their analysis found three cycles in $H_1$ rather than a single cycle, this gives rise to the $C^3$ model referenced in Section 1.2 which has three cyclic generators. In [3] Carlsson et. al. found that choice of $k$ can have an effect on whether 1 or 3 cycles are found in $H_1$ finding that when $k$ is smaller the 3 circle/Klein bottle topology is found; however, when k is increased the topology collapses to a single circle of linear gradients rotating around the circle. For both MNIST and CNN our work found only a single persistent cycle in $H_1$. The reason for this is two-fold. In general, 3 generators occur because one cycle is formed for each color value since the pixel input is actually a 3-tuple (red/green/blue). For MNIST, we found only one cycle because the MNIST images are greyscale and thus there is only a single color value. For Dogs vs Cats, the images are color images; however, we used a separate color channel for each color. Because of this, the weights were not applied to 3-tuples but rather a single hue value which made the output the same as the grayscale weights. This causes only a single generator to be discovered.

# 9 Conclusion

Given all of these results, what has actually been shown? Recall that the work of Mumford, Carlsson, et. al. showed that three main results hold:

1. The entire set of natural images is not characterizable in any meaningful way
2. The set of $n \times n$ high-contrast image patches lies on a high co-dimensional submanifold of $\mathbb{R}^{n^2}$
3. The set of high-contrast image patches resembles a 2-dimensional annulus, in particular, $H_1$ is nonempty

Turning to CNNs, the goal of the CNN is to learn features of the structure of the data set to which it is applied, i.e., characterize the ambient space from which the data is taken. Given that CNNs used for image processing apply an $n \times n$ filter to the image resulting in $n \times n$ response patches, then the weights used for these $n \times n$ filters should approximate the structure of high-variance $n \times n$ patches within the image set (high-variance because these are the patches in which most of the information about the images lie). Thus, given the prior results, the space of CNN convolutional weights should also approximate a 2-dimensional annulus, specifically $H_1$ nonempty. In the case of CNN weights, it is not necessary to restrict the study to high-contrast (high-variance) weight patches. This is because as the CNN learns, the weights will be tuned to the important features of the image set which means that low-contrast patches will be ignored since there is little to no information contained in low-contrast image patches.

This is precisely the hypothesis that Carlsson and Gabrielsson lay out in their work [18]. They demonstrate this by showing that for a CNN trained on certain data sets (MNIST, SVHN, CIFAR-10) the weights lie on a manifold with non-empty $H_1$.

Bearing this in mind, the ultimate goal that we set out to achieve was to further the theoretical understanding of CNNs built to classify natural images by using Topological Data Analysis methods to show that as it trains, a CNN encodes a structure similar to that of high-variance local image patches of natural images in its convolutional weights. This has been proposed in the Carlsson-Gabrielsson Hypotheses stated in Section 1 and demonstrated on some simple image sets [18] (MNIST, CIFAR-10, SVHN) but has not been proven or shown to be true for more complex datasets. Given the stochastic nature of CNNs it is highly unlikely that this property can ever be fully proven; therefore, we sought out to demonstrate that the Carlsson-Gabrielsson Hypothesis holds for more complex data sets, specifically for the Dogs vs Cats dataset.

In our work, we were able to replicate the results of Carlsson and Gabrielsson in [18] by performing TDA on several different CNN architectures trained using the MNIST dataset which showed that for the convolutional weights of the CNN there is a high persistence cycle in $H_1$ which, when observed using the Mapper algorithm, contains the Primary Circle (rotating gradients around a circle).

Further, and more importantly, we were able to show that a CNN built on a more complex dataset (Dogs vs Cats) also encodes this same structure (single high persistence cycle in $H_1$ and a clear Primary Circle when observed using the Mapper Diagram) in its convolutional weights and that this structure becomes clearer as training accuracy increases and becomes less clear or disappears altogether in the presence of overfitting. As a result, this work has gone a long way in demonstrating the validity of the Carlsson-Gabrielsson Hypothesis.

From a broader perspective, by showing that a CNN trained on a complex image set encodes a similar structure to that of the space from which the training images are taken we have been able to add further confidence in the theoretical robustness of the CNN model for use in image classification and have shed light on how a CNN is able to achieve good classification accuracy.

This also provides a potential avenue for use in validating CNNs and detecting malicious interference in the training data. As pointed out in Section 1, CNNs can be made to fail by making small changes to the image data which are almost imperceptible to humans. However, it is likely that this change will cause the images to no longer represent that set of natural images and as such the CNN may not encode the same structure as it does with a valid image set. If this is true then TDA can be applied to trained CNNs in order to detect whether the training data used was valid or if it had been modified. This is an area for further research.

In any case, what we have done is show that the weights of CNNs trained to perform classification on the Dogs vs Cats dataset are arranged in a structure which approximates a manifold with non-trivial geometry. Specifically, the weights lie on a space with a single $H_1$ cycle which appears to be made up of weights consisting of rotating gradients (the Primary Circle). This is a similar structure to that of high-variance local image patches of natural images. It is also the same structure as the neurons in the Primary Visual Cortex of the Mammalian Brain. This result shows that the CNN model for this type of image is robust. It also, together with the results of Carlsson and Gabrielsson [18], goes a long way in showing that this holds for any CNN designed to perform image classification, which would indicate that the CNN for natural image classification is theoretically robust.

# References

[1] Bro, Rasmus, and Age K. Smilde. Principal Component Analysis. Anal. Methods, vol. 6, no. 9, 2014, pp. 28122831., https://doi.org/10.1039/c3ay41907j.

[2] Brownlee, Jason. A Gentle Introduction to Cross-Entropy for Machine Learning. Machine Learning Mastery, 22 Dec. 2020, https://machinelearningmastery.com/cross-entropy-for-machine-learning/.

[3] Carlsson, Gunnar, et al. "On the local behavior of spaces of natural images." International journal of computer vision 76.1 (2008): 1-12.

[4] Carlsson, Gunnar. "Topology and data." Bulletin of the American Mathematical Society 46.2 (2009): 255-308.

[5] G. Carlsson and R. Gabrielsson, Topological Approaches to Deep Learning, Topological Data Analysis, Abel Symposia, vol 15, 2020

[6] Chazal, F., Michel, B.: An introduction to topological data analysis: fundamental and practical aspects for data scientists. arXiv:1710.04019 (2017)

[7] Conway, John Horton, and Neil James Alexander Sloane. Sphere packings, lattices and groups. Vol. 290. Springer Science & Business Media, 2013.

[8] Edelsbrunner, Herbert, and John L. Harer. Chapter 3 - Simplicial Complexes. *Computational Topology an Introduction, by Herbert Edelsbrunner and John L. Harer, American Mathematical Society, 2010*, pp. 5353.

[9] Edelsbrunner, Herbert, et al. Topological Persistence and Simplification. *Discrete Computational Geometry, vol. 28, no. 4, 2002*, pp. 511533., doi:10.1007/s00454-002-2885-2.

[10] Edelsbrunner, H., et al. Topological Persistence and Simplification. *Proceedings 41st Annual Symposium on Foundations of Computer Science, 2000*, doi:10.1109/sfcs.2000.892133.

[11] Elson, Jeremy, et al. Asirra. Proceedings of the 14th ACM Conference on Computer and Communications Security - CCS '07, 2007, https://doi.org/10.1145/1315245.1315291.

[12] Eisenbud, David. Commutative Algebra: with a View toward Algebraic Geometry. Springer, 1995.

[13] Ester, Martin, et al. "A density-based algorithm for discovering clusters in large spatial databases with noise." kdd. Vol. 96. No. 34. 1996.

[14] Fraleigh, John B. A First Course in Abstract Algebra. Pearson Education, 2003.

[15] Field, David J. "Relations between the statistics of natural images and the response properties of cortical cells." Josa a 4.12 (1987): 2379-2394.

[16] It Research Support Solutions Wiki. The Foundry [IT Research Support Solutions Wiki], https://wiki.itrss.mst.edu/dokuwiki/pub/foundry.

[17] K. Fukushhima, Neocognition: A self-organizing neural network for a mechanism of pattern recognition unaffected by shift in position, Biological Cybernetics, Vol. 36, Issue 4, 1980.

[18] Gabrielsson, Rickard Brel, and Gunnar Carlsson. "Exposition and interpretation of the topology of neural networks." 2019 18th ieee international conference on machine learning and applications (icmla). IEEE, 2019.

[19] Ghrist, Robert. Barcodes: The Persistent Topology of Data. *Bulletin of the American Mathematical Society, vol. 45, no. 01, 2007*, pp. 6176., doi:10.1090/s0273-0979-07-01191-3.

[20] giotto-tda: A Topological Data Analysis Toolkit for Machine Learning and Data Exploration, Tauzin et al, arXiv:2004.02551, 2020.

[21] Goldfarb, Daniel. "Understanding deep neural networks using topological data analysis." arXiv preprint arXiv:1811.00852 (2018).

[22] I. Goodfellow, J. Shlens and C. Szegedy, Explaining and Harnessing Adversarial Examples, International Conference on Learning Representations, 2015

[23] Goodman, Frederick M. Section 8.5. Algebra: Abstract and Concrete, by Frederick M. Goodman, 2.6 ed., SemiSimple Press, 2014, pp. 386391.

[24] Google Colab, Google, https://colab.research.google.com/.

[25] van Hateren, Johannes H. "Theoretical predictions of spatiotemporal receptive fields of fly LMCs, and experimental validation." Journal of Comparative Physiology A 171.2 (1992): 157-170.

[26] van Hateren, J. Hans, and Arjen van der Schaaf. "Independent component filters of natural images compared with simple cells in primary visual cortex." Proceedings of the Royal Society of London. Series B: Biological Sciences 265.1394 (1998): 359-366.

[27] Foundations of Neuroscience Subtitle:Open Edition Author:Casey Henley

[28] It Research Support Solutions Wiki. The Foundry [IT Research Support Solutions Wiki], https://wiki.itrss.mst.edu/dokuwiki/pub/foundry.

[29] D.H. Hubel and T.N. Wiesel, Receptive fields, binocular interaction and functional architecture in the cats visual cortex, Journal of Physiology, Vol. 160, Issue 1, 1962

[30] By: IBM Cloud Education. What Are Convolutional Neural Networks? IBM, https://www.ibm.com/cloud/learn/convolutional-neural-networks.

[31] ImageNet, https://www.image-net.org/.

[32] Jain A. K., Dubes R. C., Algorithms for clustering data. Prentice Hall Advanced Reference Series. Prentice Hall Inc., Englewood Cliffs, NJ, 1988.

[33] Johnson S. C., Hierarchical clustering schemes. Psychometrika 2 (1967), 241254.

[34] Dogs vs. Cats. Kaggle, https://www.kaggle.com/c/dogs-vs-cats.

[35] Chollet, Franois, and others. Keras. https://keras.io, 2015, https://keras.io.

[36] Khan, Salman, et al. A Guide to Convolutional Neural Networks for Computer Vision. Morgan & ; Claypool Publishers, 2018.

[37] Keplermapper 2.0.1 Documentation a Scikit-TDA Project. Kepler Mapper - KeplerMapper 2.0.1 Documentation, https://kepler-mapper.scikit-tda.org/en/latest/.

[38] Kullback, Solomon, and Richard A. Leibler. "On information and sufficiency." The annals of mathematical statistics 22.1 (1951): 79-86.

[39] Larsen, Erik. Topological Data Analysis on Convolutional Neural Networks. Norwegian University of Science and Technology, 2020.

[40] G. Lindsay, Convolutional neural networks as a model of the visual system: Past, present, and future, Journal of Cognitive Neuroscience, 2020

[41] Lloyd, Stuart. "Least squares quantization in PCM." IEEE transactions on information theory 28.2 (1982): 129-137.

[42] Loftsgaarden, Don O., and Charles P. Quesenberry. "A nonparametric estimate of a multivariate density function." The Annals of Mathematical Statistics 36.3 (1965): 1049-1051.

[43] A. Mahendran and A. Vedaldi., Understanding deep image representations by inverting them, IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015

[44] McCulloch, Warren; Walter Pitts (1943). "A Logical Calculus of Ideas Immanent in Nervous Activity". Bulletin of Mathematical Biophysics. 5 (4): 115133. doi:10.1007/BF02478259.

[45] Melcher, Kathrin, et al. A Friendly Introduction to [Deep] Neural Networks. KNIME, https://www.knime.com/blog/a-friendly-introduction-to-deep-neural-networks.

[46] Minsky, M.; S. Papert (1969). An Introduction to Computational Geometry. MIT Press. ISBN 978-0-262-63022-1.

[47] A.B. Lee, K. Pedersen and D. Mumford, The non-linear statistics of high-contrast patches in natural images, International Journal of Computer Vision, Vol. 54, Issue 1, 2003

[48] Munkres, J. R. Algebraic Topology. Addison-Wesley, 1984.

[49] Neyman, Jerzy, and Egon Sharpe Pearson. "IX. On the problem of the most efficient tests of statistical hypotheses." Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character 231.694-706 (1933): 289-337.

[50] J.A. Perea, A Brief History of Persistence, arXiv:1809.03624.

[51] Plotly. Plotly Python Graphing Library, https://plotly.com/python/.

[52] Reinagel, Pamela, and Anthony M. Zador. "Natural scene statistics at the centre of gaze." Network: Computation in Neural Systems 10.4 (1999): 341.

[53] Ripser.py 0.6.2 Documentation a Scikit-TDA Project. Setup - Ripser.py 0.6.2 Documentation, https://ripser.scikit-tda.org/en/latest/.

[54] Ruderman, Daniel L. "The statistics of natural images." Network: computation in neural systems 5.4 (1994): 517.

[55] Saha, Sumit. A Comprehensive Guide to Convolutional Neural Networks-the eli5 Way. Medium, Towards Data Science, 17 Dec. 2018, https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[56] Sasirekha, K., and P. Baby. "Agglomerative hierarchical clustering algorithm-a." International Journal of Scientific and Research Publications 83.3 (2013): 83.

[57] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

[58] K. Simonyan, A. Vedaldi and A. Zisserman. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps, ICLR, 2014

[59] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

[60] Singh, Gurjeet, Facundo Mmoli, and Gunnar E. Carlsson. "Topological methods for the analysis of high dimensional data sets and 3d object recognition." PBG@ Eurographics 2 (2007).

[61] Wagenknecht, A. "Existence-of-the-Primary-Circle-in-Convolutional-Neural-Networks" (Version 1.0.0) [Computer software].

https://github.com/adam-wagen/Existence-of-the-Primary-Circle-in-Convolutional-Neural-Networks (2023)

[62] LeCun, Yann and Cortes, Corinna and Burges, CJ, "MNIST handwritten digit database" ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist (2010)

[63] M.D. Zeiler and R. Fergus, Visualizing and Understanding Convolutional Networks, Computer Vision ECCV, 2014

[64] Ziou, Djemel, and Salvatore Tabbone. "Edge detection techniques-an overview." Pattern Recognition and Image Analysis C/C of Raspoznavaniye Obrazov I Analiz Izobrazhenii 8 (1998): 537-559.

[65] A. Zomorodian and G. Carlsson, Computing Persistent Homology, *Discrete and Computational Geometry 33,* 2 (2005) 249-247

# A Computation of Persistent Homology an Example

In this section we will take a look at two different filtrations and apply the method used in Section 2.3.5 for computing persistent homology over a field. In our case, we will compute both examples over the field $\mathbb{Z}_2$, but of course the method is the same over any field.

## A.1 Example 1

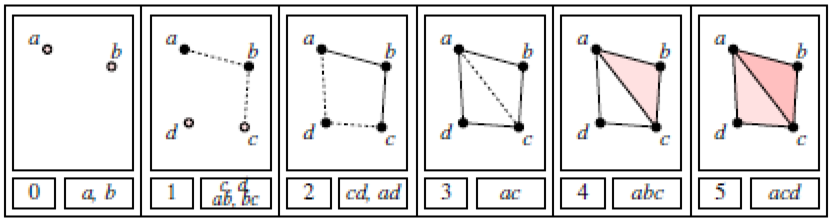


Figure A.1: Filtration 1, taken from [65]

Consider the filtration above taken from [65]. The simplices and degrees from this filtration are:

| a | b | c | d | ab | bc | cd | ad | ac | abc | acd |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 | 1 | 2 | 2 | 3 | 4 | 5 |

Using this table then we get the following matrix representation $M_1$ for $\partial_1$:

$$M_1 = \left[\begin{array}{c|ccccc} & ab & bc & cd & ad & ac \\ \hline d & 0 & 0 & t & t & 0 \\ c & 0 & 1 & t & 0 & t^2 \\ b & t & t & 0 & 0 & 0 \\ a & t & 0 & 0 & t^2 & t^3 \end{array}\right] \Rightarrow \left[\begin{array}{c|ccccc} & cd & bc & ab & ad & ac \\ \hline d & t & 0 & 0 & t & 0 \\ c & t & 1 & 0 & 0 & t^2 \\ b & 0 & t & t & 0 & 0 \\ a & 0 & 0 & t & t^2 & t^3 \end{array}\right] \Rightarrow$$

$$\left[\begin{array}{c|ccccc} & cd & bc & ab & ad-cd & ac-t^2\cdot bc \\ \hline d & t & 0 & 0 & 0 & 0 \\ c & t & 1 & 0 & t & 0 \\ b & 0 & t & t & 0 & t^3 \\ a & 0 & 0 & t & t^2 & t^3 \end{array}\right] \Rightarrow$$

$$\left[\begin{array}{c|ccccc} & cd & bc & ab & ad - cd - t\cdot bc & ac - t^2\cdot bc - t^2\cdot ab \\ \hline d & t & 0 & 0 & 0 & 0 \\ c & t & 1 & 0 & 0 & 0 \\ b & 0 & t & t & t^2 & 0 \\ a & 0 & 0 & t & t^2 & 0 \end{array}\right]$$

$$\Rightarrow \tilde{M}_1 = \left[\begin{array}{c|ccccc} & cd & bc & ab & ad - cd - t\cdot bc - t\cdot ab & ac - t^2\cdot bc - t^2\cdot ab \\ \hline d & t & 0 & 0 & 0 & 0 \\ c & t & 1 & 0 & 0 & 0 \\ b & 0 & t & t & 0 & 0 \\ a & 0 & 0 & t & 0 & 0 \end{array}\right]$$

Thus we get a basis $\{z_1, z_2\} = \{ac - t^2\cdot bc - t^2\cdot ab, ad - cd - t\cdot bc - t\cdot ab\}$ for $Z_1$.

Now as described in the previous section we begin to calculate $\tilde{M}_2$ by representing $M_2$ relative to the standard bases for $C_2$ and $C_1$ and so we get:

$$M_2 = \left[\begin{array}{c|cc} & abc & acd \\ \hline ac & t & t^2 \\ ad & 0 & t^3 \\ cd & 0 & t^3 \\ bc & t^3 & 0 \\ ab & t^3 & 0 \end{array}\right]$$

Now we remove the rows corresponding to the nonzero columns in $\tilde{M}_1$ and are left with:

$$\tilde{M}_2 = \left[\begin{array}{c|cc} & abc & acd \\ \hline z_1 & t & t^2 \\ z_2 & 0 & t^3 \end{array}\right] \Rightarrow \left[\begin{array}{c|cc} & abc & acd - t\cdot abc \\ \hline z_1 & t & 0 \\ z_2 & 0 & t^3 \end{array}\right]$$

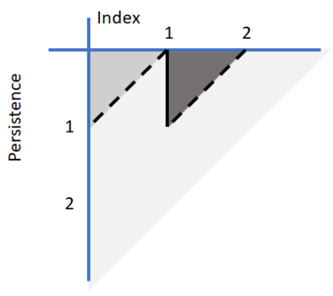
where $z_1$ and $z_2$ are defined as above.

Since there are no 3-simplices in our filtration this concludes the description of the persistent homology. Lastly, we use Theorem 2.13 to obtain the $\mathcal{P}$-interval in each dimension.

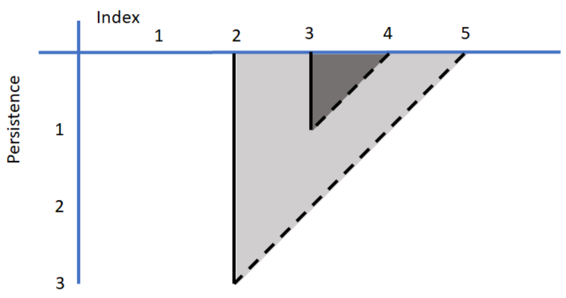
For dimension 1 we get:
$d \to (1, 2)$
$c \to (1, 1)$
$b \to (0, 1)$
$a \to (0, \infty)$

And for dimension 2 we have:
$z_1 \to (3,4)$
$z_2 \to (2,5)$

Plotting these we get:



$\mathcal{P}$-intervals for dimension 1



$\mathcal{P}$-intervals for dimension 2

Figure A.2:

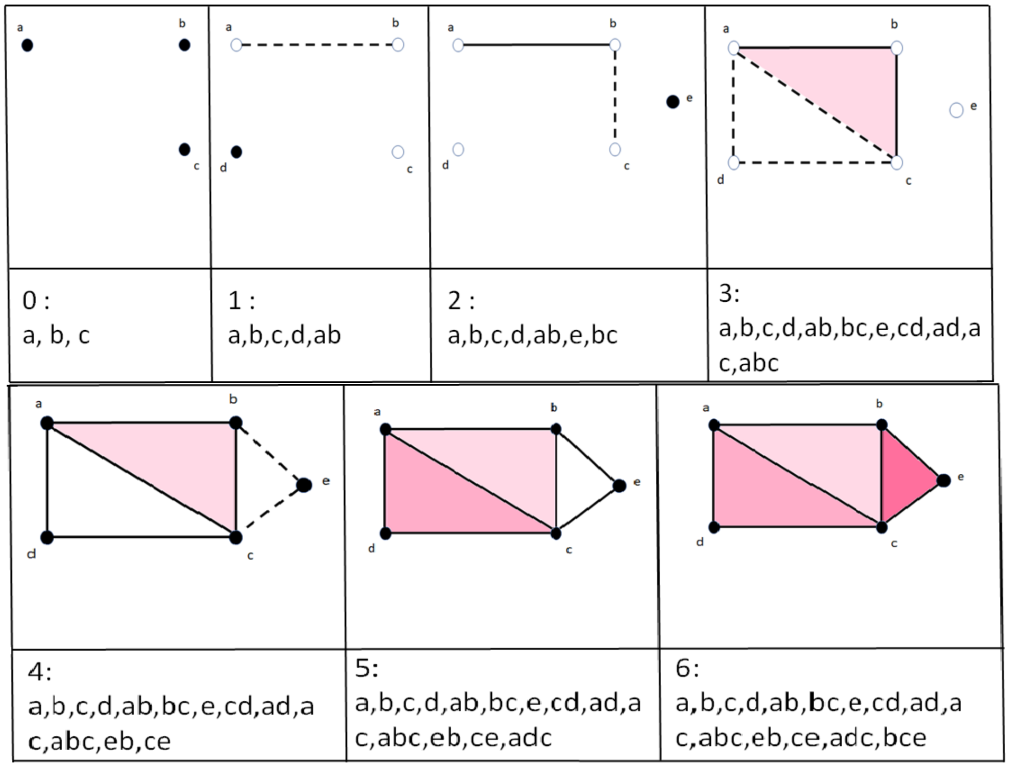
We now recall Lemma 2.9, which says that the $l$ level $p$ persistent Betti number for the $k$-th dimension is the number of triangles containing the point $(l,p)$.Using this we get the following persistent Betti numbers:

$\beta_1^{1,0} = \beta_1^{0,0} = 2$
$\beta_1^{i,p} = 1$ for $p \geq 0, i \geq 0$ and $(i,p) \notin \{(1,0),(0,0)\}$
Otherwise $\beta_1^{i,p} = 0$

$\beta_2^{3,0} = 2$
$\beta_2^{2,1} = \beta_2^{4,0} = \beta_2^{2,2} = \beta_2^{2,0} = \beta_2^{3,1} = 1$
Otherwise $\beta_2^{i,p} = 0$

## A.2 Example 2

Consider the following filtration.



a
b
c
d
e
0 : a, b, c
1 : a,b,c,d,ab
2 : a,b,c,d,ab,e,bc
3: a,b,c,d,ab,bc,e,cd,ad,ac,abc
4: a,b,c,d,ab,bc,e,cd,ad,ac,abc,eb,ce
5: a,b,c,d,ab,bc,e,cd,ad,ac,abc,eb,ce,adc
6: a,b,c,d,ab,bc,e,cd,ad,ac,abc,eb,ce,adc,bce

Figure A.3: Filtration 2

The simplices and degrees from this filtration are:

| a | b | c | d | ab | e | bc | cd | ad | abc | ac | be | ce | acd | bce |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 2 | 2 | 3 | 3 | 3 | 3 | 4 | 4 | 5 | 6 |

Using this table we get the following matrix representation $M_1$ for $\partial_1$:

$$M_1 = \left[\begin{array}{c|ccccccc} & ab & bc & cd & ad & ac & be & ce \\ \hline e & 0 & 0 & 0 & 0 & 0 & t^2 & t^2 \\ d & 0 & 0 & t^2 & t^2 & 0 & 0 & 0 \\ c & 0 & t^2 & t^3 & 0 & t^3 & 0 & t^4 \\ b & t & t^2 & 0 & 0 & 0 & t^4 & 0 \\ a & t & 0 & 0 & t^3 & t^3 & 0 & 0 \end{array}\right] \Rightarrow \left[\begin{array}{c|ccccccc} & be & cd & bc & ad & ac & ab & ce \\ \hline e & t^2 & 0 & 0 & 0 & 0 & t^2 & t^2 \\ d & 0 & t^2 & 0 & t^2 & 0 & 0 & 0 \\ c & 0 & t^3 & t^2 & 0 & t^3 & 0 & t^4 \\ b & t^4 & 0 & t^2 & 0 & 0 & t^4 & 0 \\ a & 0 & 0 & 0 & t^3 & t^3 & 0 & 0 \end{array}\right] \Rightarrow$$

$$\left[\begin{array}{c|ccccccc} & be & cd & bc & ad-cd & ac-t\cdot bc & ab & ce-be \\ \hline e & t^2 & 0 & 0 & 0 & 0 & 0 & 0 \\ d & 0 & t^2 & 0 & 0 & 0 & 0 & 0 \\ c & 0 & t^3 & t^2 & t^3 & 0 & 0 & t^4 \\ b & t^4 & 0 & t^2 & 0 & t^3 & t & t^4 \\ a & 0 & 0 & 0 & t^3 & t^3 & t & 0 \end{array}\right] \Rightarrow$$

$$\left[\begin{array}{c|ccccccc} & be & cd & bc & ad-cd-t\cdot bc & ac-t\cdot bc & ab & ce-be-t^2\cdot bc \\ \hline e & t^2 & 0 & 0 & 0 & 0 & 0 & 0 \\ d & 0 & t^2 & 0 & 0 & 0 & 0 & 0 \\ c & 0 & t^3 & t^2 & 0 & 0 & 0 & 0 \\ b & t^4 & 0 & t^2 & t^3 & t^3 & t & 0 \\ a & 0 & 0 & 0 & t^3 & t^3 & t & 0 \end{array}\right] \Rightarrow$$

$$\tilde{M}_1 = \left[\begin{array}{c|ccccccc} & & & & & ac-t\cdot bc & ad-cd & ce-be \\ & be & cd & bc & ab & -t^2\cdot ab & -t\cdot bc-t^2\cdot ab & -t^2\cdot bc \\ \hline e & t^2 & 0 & 0 & 0 & 0 & 0 & 0 \\ d & 0 & t^2 & 0 & 0 & 0 & 0 & 0 \\ c & 0 & t^3 & t^2 & 0 & 0 & 0 & 0 \\ b & t^4 & 0 & t^2 & t & 0 & 0 & 0 \\ a & 0 & 0 & 0 & t & 0 & 0 & 0 \end{array}\right]$$

Thus we get a basis $\{z_1, z_2, z_3\} = \{ce - be - t^2 \cdot bc, ac - t \cdot bc - t^2 \cdot ab, ad - cd - t \cdot bc - t^2 \cdot ab\}$ for $Z_1$.

Now as described in the previous section we begin to calculate $\tilde{M}_2$ by representing $M_2$ relative to the standard bases for $C_2$ and $C_1$ and so we get:

$$M_2 = \left[\begin{array}{c|ccc} & abc & acd & bce \\ \hline ce & 0 & 0 & t^2 \\ be & 0 & 0 & t^2 \\ ac & 1 & t^2 & 0 \\ ad & 0 & t^2 & 0 \\ cd & 0 & t^2 & 0 \\ bc & t & 0 & t^4 \\ ab & t^2 & 0 & 0 \end{array}\right]$$

Now we remove the rows corresponding to the nonzero columns in $\tilde{M}_1$ and are left with:

$$M_2 = \left[ \begin{array}{c|ccc} & abc & acd & bce \\ \hline z_1 & 0 & 0 & t^2 \\ z_2 & 1 & t^2 & 0 \\ z_3 & 0 & t^2 & 0 \end{array} \right]$$

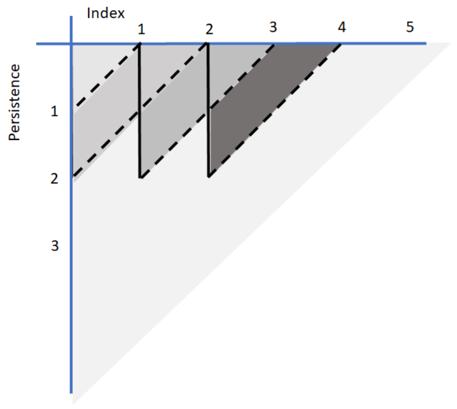
where $z_1, z_2$, and $z_3$ are defined as above.

Since there are no 3-simplices in our filtration this concludes the description of the persistent homology. Lastly we use Theorem 2.13 to obtain the $\mathcal{P}$-interval in each dimension.

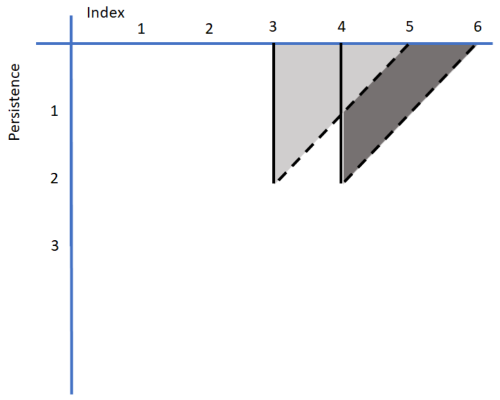
For dimension 1 we get:
$e \to (2, 4)$
$d \to (1, 3)$
$c \to (0, 2)$
$b \to (0, 1)$
$a \to (0, \infty)$

And for dimension 2 we have:
$z_1 \to (4, 6)$
$z_2 \to (3, 3)$
$z_3 \to (3, 5)$

Plotting these we get:



$\mathcal{P}$-intervals for dimension 1



$\mathcal{P}$-intervals for dimension 2

Figure A.4:

We now recall Lemma 2.9, which says that the $l$ level $p$ persistent Betti number for the $k$-th dimension is the number of triangles containing the point $(l, p)$.

Using this, we get the following persistent Betti numbers:

$\beta_1^{1,0} = \beta_1^{2,0} = \beta_1^{0,0} = 3$
$\beta_1^{3,0} = \beta_1^{0,1} = \beta_1^{1,1} = \beta_1^{2,1} = 2$
$\beta_1^{i,p} = 1$ for $p \geq 0, i \geq 0$ and $(i,p) \notin \{(0,0),(1,0),(2,0),(3,0),(0,1),(1,1),(2,1)\}$
Otherwise $\beta_1^{i,p} = 0$

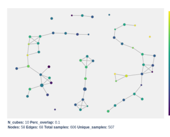
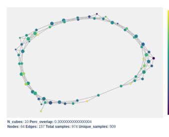
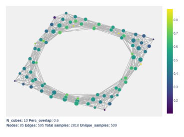
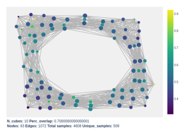
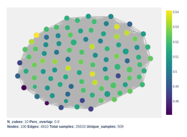
$\beta_2^{4,0} = 2$
$\beta_2^{3,0} = \beta_2^{5,0} = \beta_2^{3,1} = \beta_2^{4,1} = 1$
Otherwise $\beta_2^{i,p} = 0$

## B Dogs vs Cats Mapper Hyper-parameter Analysis

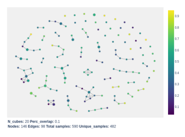
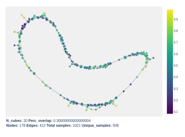
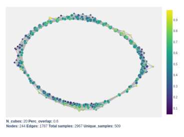
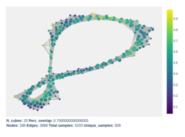
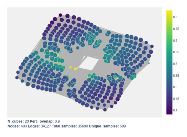
Below is the full set of results from the Mapper hyper-parameter testing referenced in Section 7.4.1. These runs were made using using number of cubes $\in \{10, 20, 30, 40, 50, 60\}$ and overlap $\in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$ $k = 100$ and $\rho = 0.05$.

### Layer 1

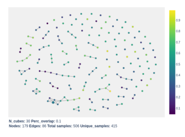
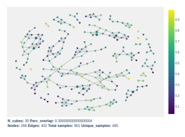
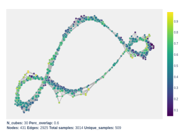
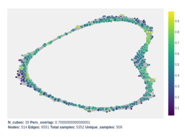
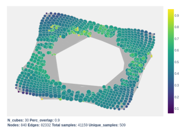
Cubes = 10

| Overlap 0.1 | Overlap 0.3 | Overlap 0.6 | Overlap 0.7 | Overlap 0.9 |
|---|---|---|---|---|
|  |  |  |  |  |

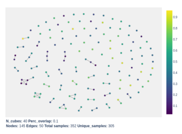
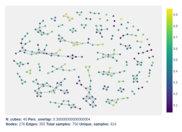
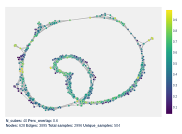
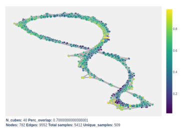
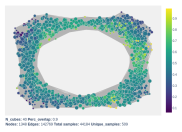
Cubes = 20

| Overlap 0.1 | Overlap 0.3 | Overlap 0.6 | Overlap 0.7 | Overlap 0.9 |
|---|---|---|---|---|
|  |  |  |  |  |

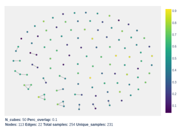
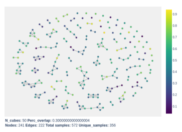
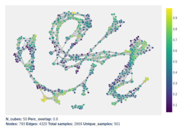
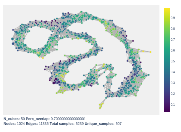
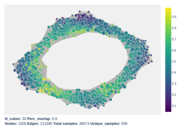
Cubes = 30

| Overlap 0.1 | Overlap 0.3 | Overlap 0.6 | Overlap 0.7 | Overlap 0.9 |
|---|---|---|---|---|
|  |  |  |  |  |

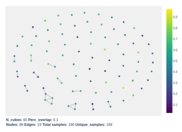
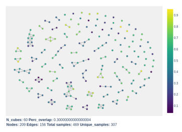
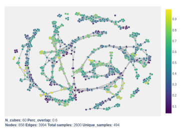
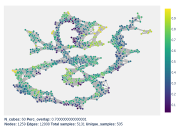
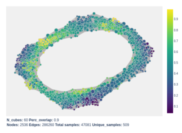
Cubes = 40

| Overlap 0.1 | Overlap 0.3 | Overlap 0.6 | Overlap 0.7 | Overlap 0.9 |
|---|---|---|---|---|
|  |  |  |  |  |

Cubes = 50

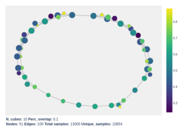
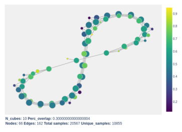
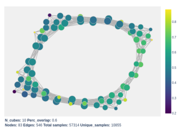
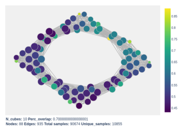
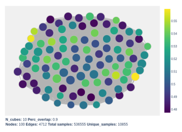
| Overlap 0.1 | Overlap 0.3 | Overlap 0.6 | Overlap 0.7 | Overlap 0.9 |
|---|---|---|---|---|
|  |  |  |  |  |

Cubes = 60

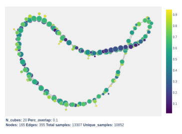
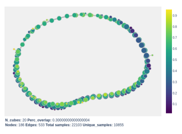
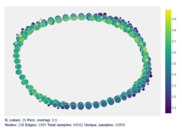
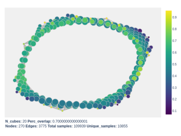
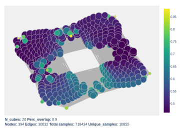
| Overlap 0.1 | Overlap 0.3 | Overlap 0.6 | Overlap 0.7 | Overlap 0.9 |
|---|---|---|---|---|
|  |  |  |  |  |

## Layer 2

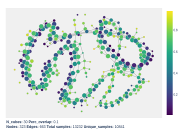
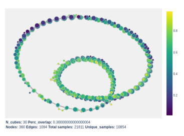
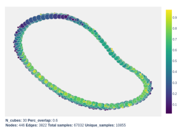
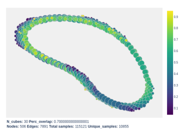
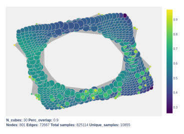
Cubes = 10

| Overlap 0.1 | Overlap 0.3 | Overlap 0.6 | Overlap 0.7 | Overlap 0.9 |
|---|---|---|---|---|
|  |  |  |  |  |

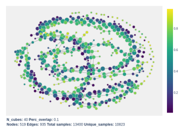
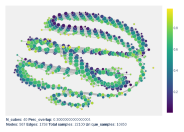
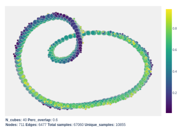
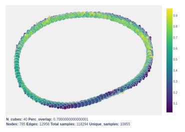
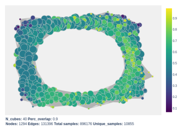
Cubes = 20

| Overlap 0.1 | Overlap 0.3 | Overlap 0.6 | Overlap 0.7 | Overlap 0.9 |
|---|---|---|---|---|
|  |  |  |  |  |

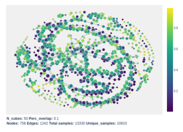
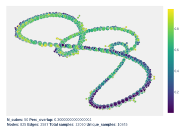
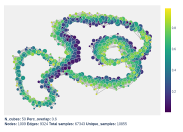
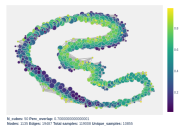
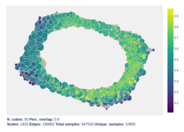
Cubes = 30

| Overlap 0.1 | Overlap 0.3 | Overlap 0.6 | Overlap 0.7 | Overlap 0.9 |
|---|---|---|---|---|
|  |  |  |  |  |

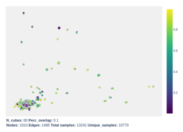
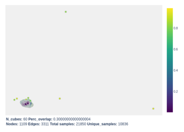
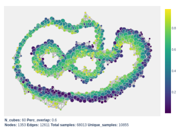
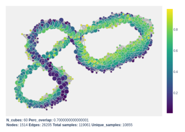
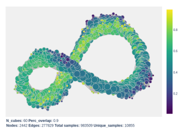
Cubes = 40

| Overlap 0.1 | Overlap 0.3 | Overlap 0.6 | Overlap 0.7 | Overlap 0.9 |
|---|---|---|---|---|
|  |  |  |  |  |

Cubes = 50

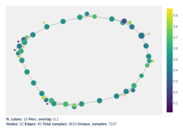
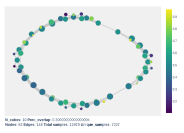
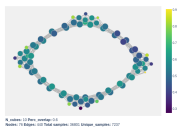
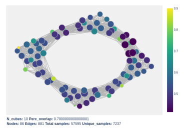
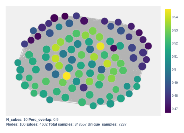
| Overlap 0.1 | Overlap 0.3 | Overlap 0.6 | Overlap 0.7 | Overlap 0.9 |
|---|---|---|---|---|
|  |  |  |  |  |

Cubes = 60

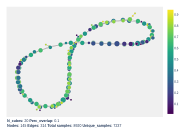
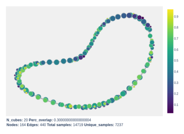
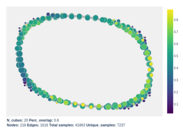
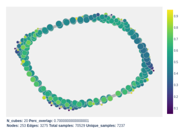
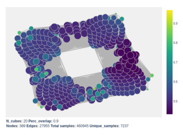
| Overlap 0.1 | Overlap 0.3 | Overlap 0.6 | Overlap 0.7 | Overlap 0.9 |
|---|---|---|---|---|
|  |  |  |  |  |

**Layer 3**

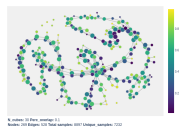
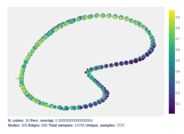
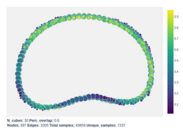
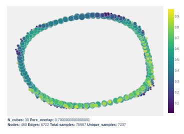
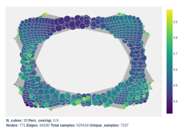
Cubes = 10

| Overlap 0.1 | Overlap 0.3 | Overlap 0.6 | Overlap 0.7 | Overlap 0.9 |
|---|---|---|---|---|
|  |  |  |  |  |

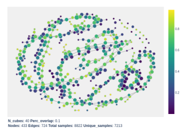
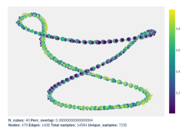
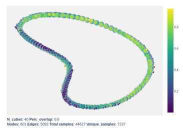
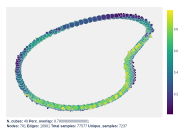
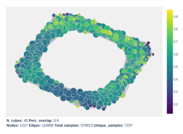
Cubes = 20

| Overlap 0.1 | Overlap 0.3 | Overlap 0.6 | Overlap 0.7 | Overlap 0.9 |
|---|---|---|---|---|
|  |  |  |  |  |

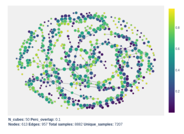
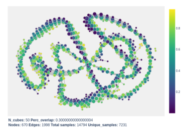
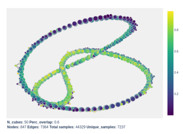
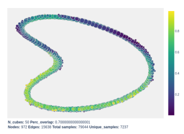
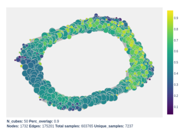
Cubes = 30

| Overlap 0.1 | Overlap 0.3 | Overlap 0.6 | Overlap 0.7 | Overlap 0.9 |
|---|---|---|---|---|
|  |  |  |  |  |

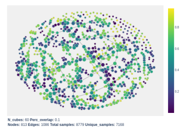
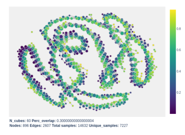
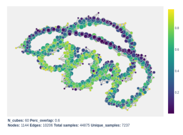
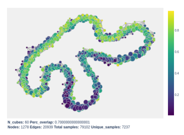
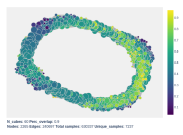
Cubes = 40

| Overlap 0.1 | Overlap 0.3 | Overlap 0.6 | Overlap 0.7 | Overlap 0.9 |
|---|---|---|---|---|
|  |  |  |  |  |

Cubes = 50

| Overlap 0.1 | Overlap 0.3 | Overlap 0.6 | Overlap 0.7 | Overlap 0.9 |
|---|---|---|---|---|
|  |  |  |  |  |

Cubes = 60

| Overlap 0.1 | Overlap 0.3 | Overlap 0.6 | Overlap 0.7 | Overlap 0.9 |
|---|---|---|---|---|
|  |  |  |  |  |

# C 4-Layer CNN for MNIST

## Layer 1

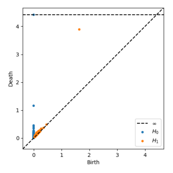
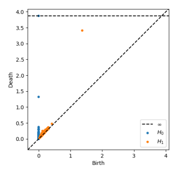
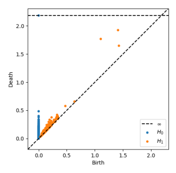
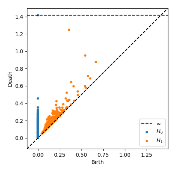
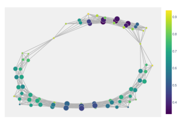
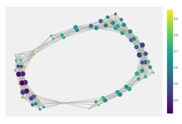
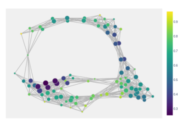
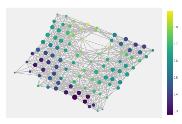
| 1 Epochs | 10 Epochs | 20 Epochs | 60 Epochs |
|---|---|---|---|
|  |  |  |  |
|  |  |  |  |

Table C.1: *Top: Persistence Diagrams for the weights from the 1st layer of the 4-Layer MNIST CNN Bottom: Mapper Diagrams for the weights from the 1st layer of the 4-Layer MNIST CNN*

## Layer 2

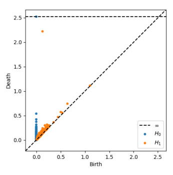
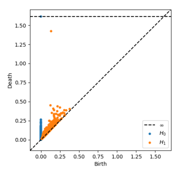
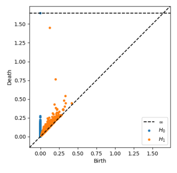
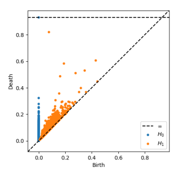
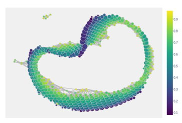
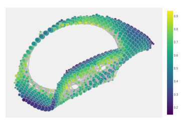
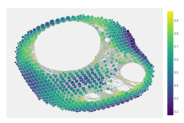
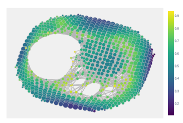
| 1 Epochs | 10 Epochs | 20 Epochs | 60 Epochs |
|---|---|---|---|
|  |  |  |  |
|  |  |  |  |

Table C.2: *Top: Persistence Diagrams for the weights from the 2nd layer of the 4-Layer MNIST CNN Bottom: Mapper Diagrams for the weights from the 2nd layer of the 4-Layer MNIST CNN*

## Layer 3

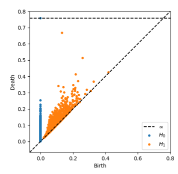
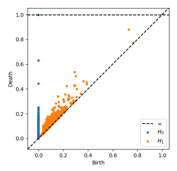
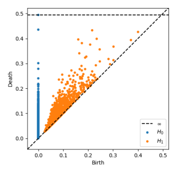
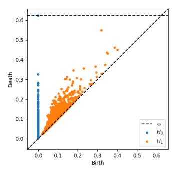
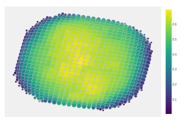
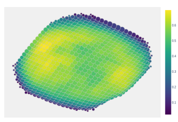
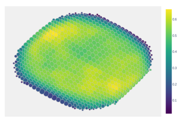
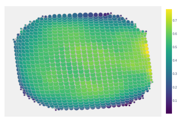
| 1 Epochs | 10 Epochs | 20 Epochs | 60 Epochs |
|---|---|---|---|
|  |  |  |  |
|  |  |  |  |

Table C.3: *Top: Persistence Diagrams for the weights from the 3rd layer of the 4-Layer MNIST CNN Bottom: Mapper Diagrams for the weights from the 3rd layer of the 4-Layer MNIST CNN*

## Layer 4

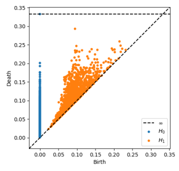
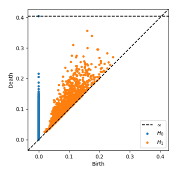
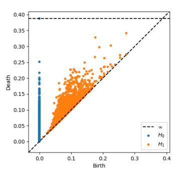
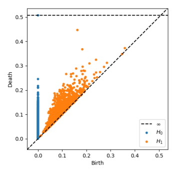
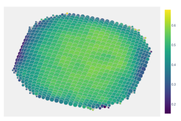
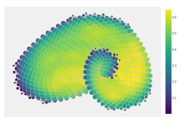
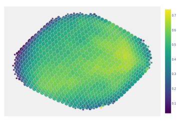
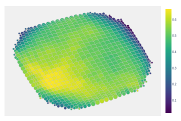
| 1 Epochs | 10 Epochs | 20 Epochs | 60 Epochs |
|---|---|---|---|
|  |  |  |  |
|  |  |  |  |

Table C.4: *Top: Persistence Diagrams for the weights from the 4th layer of the 4-Layer MNIST CNN Bottom: Mapper Diagrams for the weights from the 4th layer of the 4-Layer MNIST CNN*