Topological Data Analysis of Convolutional Neural Networks Using Depthwise Separable Convolutions

Eliot Courtois

University of Missouri-St. Louis, ecvd6@mail.umsl.edu

Follow this and additional works at: https://irl.umsl.edu/dissertation

Part of the Data Science Commons, and the Geometry and Topology Commons

Recommended Citation

https://irl.umsl.edu/dissertation/1322

This Dissertation is brought to you for free and open access by the UMSL Graduate Works at IRL @ UMSL. It has been accepted for inclusion in Dissertations by an authorized administrator of IRL @ UMSL. For more information, please contact marvinh@umsl.edu.
Topological Data Analysis of Convolutional Neural Networks Using Depthwise Separable Convolutions

Eliot Courtois

M.S., Actuarial Science, Maryville University, 2018

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

August
2023

Advisory Committee

Adrian Clingher, Ph.D.
Chairperson

Haiyan Cai, Ph.D.

David Covert, Ph.D.

Qingtang Jiang, Ph.D.
6 Applying TDA Tools to a Trained CNN, Prior Related Research
6.1 Topology and Natural Image Patch Distributions ........................................ 91
6.2 Carlsson and Gabrielsson’s Methods and Findings ........................................ 94

7 Our Methods
7.1 CNN Architecture for Our Work ................................................................. 100
7.2 Training Process and Extracting the Needed \( R^9 \) Points ................................. 102
7.3 Preprocessing and Specifications for TDA Procedures ..................................... 104
7.4 Technical Breakdown .................................................................................... 107

8 Our Results
8.1 Summary of Observations by Layer ............................................................... 110
   8.1.1 \( SC_4 \) ..................................................................................................... 110
   8.1.2 \( SC_3 \) ..................................................................................................... 111
   8.1.3 \( SC_2 \) ..................................................................................................... 112
   8.1.4 \( SC_1 \) ..................................................................................................... 114
   8.1.5 \( C_1 \) ..................................................................................................... 117
8.2 Global Summary of Findings ............................................................................ 119
List of Figures

1.1 Hubel-Wiesel Pathway vs. CNN model, from [25]. .............................................. 8
2.1 2-node input layer, 2 3-node hidden layers, single-node output layer .................... 10
2.2 Network from Figure 2.1 with weights and nodes filled in before the activation step, described in Section 2.1.3. ................................................................. 12
2.3 visualizations of several commonly-used single-node activation functions. ................. 14
2.4 Figure 2.1’s network with all values filled in for the nodes and weights, post-activation where each non-input value has a ReLU function applied. .................................. 15
2.5 example accuracy and loss plots for an image-processing CNN taking in training (blue curve) and validation (orange curve) data. ................................................. 22
2.6 diagram summarizing the addition of a batch norm operation to a NN layer, from [28]. 24
2.7 example architecture roadmap for a C-class classification CNN, from [33]. ................. 26
2.8 visualization of a $3 \times 3 \times 1$ kernel operating with a $4 \times 4 \times 1$ patch of pixels .......... 26
2.9 Figure 2.8 with a max pooling operation added .................................................... 27
2.10 visualization of how the convolution operation extends to multi-channel layers. ........ 28
3.1 visualization of the two steps of a 32-filter $3 \times 3$ separable convolution operation with valid padding applied to an example $X \times X \times 16$ layer to produce a $X - 2 \times X - 2 \times 32$ layer. 29
3.2 Example $4 \times 4$ set of input nodes with 3 channels. ........................................... 31
3.3 $3 \times 3 \times 3$ set of parameters to be applied per-channel for Step 1, the spatial correlation step. 31
3.4 Output of applying the filters in Figure 3.3 to the nodes in Figure 3.2 ....................... 32
3.5 $4 \times 1 \times 1$ sets of parameters to be applied in Step 2, the cross-channel convolution step. 32
3.6 Figure 3.5’s Step 2 filters, re-displayed in different colors to match the outputs in Figure 3.7. ............................................................................................................ 32
3.7 The final output of applying the parameters in Figures 3.3 and 3.6 to the input from Figure 3.2. ............................................................................................................. 33
3.8 Standard convolution image from Figure 2.10. .................................................... 33
3.9 Separable convolution image from Figure 3.1 ....................................................... 34
3.10 Example Inception module with distinct combinations of $1 \times 1$ and $3 \times 3$ standard convolutions, from [8]. ................................................................. 36
3.11 Simplified Inception example with the same combination of convolutions repeated for each of its three operations, from [8]. ....................................................... 36
3.12 Restructured Inception module, functionally equivalent to Figure 3.11, from [8]. ....... 37
3.13 Logical extension of Figure 3.12, applying a $3 \times 3$ convolution to each output channel of the initial $1 \times 1$ convolution, from [8]. ....................................................... 37
4.1 visual demonstration of a continuous transformation (in fact, a homeomorphism) between a coffee mug and a doughnut. ....................................................... 40
4.2 open sets in $S^1$ as defined by the subspace topology from $\mathbb{R}^2$. ......................... 42
4.3 open sets in $S^2$ as defined by the subspace topology from $\mathbb{R}^3$. ......................... 42
4.4 an example of an open set in a space consisting of two cubes glued together along a face. 45
4.5 visualization of the pre-image of an open interval in $\mathbb{R}$ through the simple projection map $\pi$ described in Example 4.4.4. ....................................................... 46
4.6 a simple 2-dimensional CW complex. ...................................................................... 47
4.7 Visualization of combining $D^0$ with $D^1$ or $D^2$ to form $S^1$ and $S^2$ respectively. 49
4.8 Figure 4.6 with labels to illustrate the CW pair $(X, A)$. ........................................ 49
4.9 illustration of two-point intersections between $S^1$ and elements from $\mathbb{R}^3$. 
4.10 illustration of two-point intersections between $S^2$ and elements from $\mathbb{R}^3$. 
4.11 attaching a copy of $S^1$ to every point on an original instance of $S^1$ (e.g. the inner ring in this figure) forms the standard hollow torus in $\mathbb{R}^3$. 
4.12 visualization of the quotient map on a rectangle (i.e. $I \times I$) for defining a torus through edge identification, from [24]. 

5.1 In this example, the homology class $\gamma$ is born at $H_p(K_t)$ and dies entering $H_p(K_f)$. Diagram from [10]. 
5.2 (left) persistence barcodes of dimension 0 and 1. Note the three longer bars in the dimension-1 barcodes, corresponding to three distinct 1-dimensional holes. (right) a persistence diagram of dimension 0 (red) and 1 (green). There is one significant 1-dimensional homology class, given the single point seen far above the diagonal. 

6.1 The three-circle space, as pictured in [7]. 
6.3 Natural Image data as viewed in [7] using $k = 300, p = .3$. 
6.4 the primary and secondary circles of $3 \times 3$ image patches, from [6]. 
6.5 arrangement of image-patch primary and secondary circles using identified points on a rectangle in the form of a Klein bottle, from [7]. 
6.6 Carlsson and Gabrielsson’s CNN Architecture, from [12]. 
6.7 circular Mapper diagram generated from MNIST-trained CNN kernels, from [6].
6.8 Persistence barcode for the kernels forming Figure 6.7, from [6]................. 95
6.9 Layer 2’s Mapper diagram, from [6]..................................................... 96
6.10 harder-to-interpret Mapper generated by thoroughly trained first-layer CNN filters on grayscale CIFAR-10 images, from [6]................................. 96
6.11 circular Mapper diagram with a cluster of nodes representing a non-edge “bullseye” pattern in second-layer CNN filters trained on grayscale CIFAR-10 images, from [6]...................... 97
6.12 first- and second-layer Mapper diagrams for grayscale image-trained networks, arranged by number of batch iterations, from [6]........................................ 97
6.13 circular Mapper as generated by the first-layer kernels of CIFAR-10 color-image trained CNNs, from [6]................................................................. 98
6.14 persistence barcodes and Mapper diagram showing three intersecting circles in the space of first-layer CNN filters trained on color CIFAR-10 images, from [6]...................... 98
6.15 first- and second-layer Mapper diagrams for color image-trained networks, arranged by number of batch iterations, from [6]........................................ 99

7.1 sample of x-ray images from the dataset with their corresponding labels: NORMAL and PNEUMONIA................................................................. 101
7.2 listing of parameter amounts in each layer of the network described in Table 7.1........................................................................ 102
7.3 listing of parameter amounts in each layer of the network described in Table 7.1, but with standard convolutions in place of separable convolutions........... 102
7.4 Example validation accuracy plot, taken at the end of each epoch, of one of our CNNs trained on the X-Ray dataset......................................................... 103
7.5 Average testing accuracies, taken every 5 epochs, from our set of 518 CNNs........ 103
7.6 Mapper diagram constructed from kernels in the fifth and final convolutional layer of our CNNs, along with the accompanying persistence diagram.................. 107

8.1 visualizations of four edge-detecting kernels constructed from taking the means of all 9-dimensional data points in four regions of the Mapper shown in Figure 8.1 (left), taken from the fourth separable layer of our 518 CNNs with a $p = .11$ density filtration........ 110
8.2 Mapper diagram used to derive the kernels in Figure 8.1, along with the persistence diagram to verify the clear appearance of a single circle in the point cloud........ 110
8.3 Persistence and Mapper diagram taken from $SC_4$ at 5 epochs of training........ 111
8.4 Persistence and Mapper diagram taken from $SC_4$ at 5 epochs of training and density filtration $p = .14$................................................................. 111
8.5 Persistence and Mapper diagrams for $SC_3$ at 50 epochs of training and density filtration $p = .11$................................................................. 111
8.6 Persistence and Mapper diagram taken from $SC_3$ at 50 epochs of training and density filtration $p = .01$................................................................. 112
8.7 Persistence and Mapper diagrams for $SC_2$ at 5 epochs of training and density filtration $p = .02$................................................................. 112
8.8 Persistence and Mapper diagrams for $SC_2$ at 50 epochs of training and density filtration $p = .02$................................................................. 113
8.9 Persistence and Mapper diagrams for $SC_2$ at 5 epochs of training, density filtration $p = .08$................................................................. 113
8.10 Persistence and Mapper diagrams (10 cubes) for $SC_2$ at 50 epochs of training, density filtration $p = .08$................................................................. 114
8.11 Persistence and Mapper diagrams for $SC_1$ at 5 epochs of training, density filtration $p = .07$................................................................. 114
8.12 Persistence and Mapper diagrams for $SC_1$ at 5 epochs of training, density filtration $p = .14$................................................................. 115
8.13 Persistence and Mapper diagrams for $SC_1$ at 20 epochs of training, density filtration $p = .07$................................................................. 115
8.14 Persistence and Mapper diagrams for $SC_1$ at 20 epochs of training, density filtration $p = .14$................................................................. 116
8.15 Persistence and Mapper (10 cubes) diagrams for $SC_1$ at 50 epochs of training, density filtration $p = .06$................................................................. 116
8.16 Persistence and Mapper (10 cubes) diagrams for $SC_1$ at 50 epochs of training, density filtration $p = .1$................................................................. 116
8.17 Persistence and Mapper diagrams for $C_1$ at 5 epochs of training, density filtration $p = .05$................................................................. 117
8.18 Persistence and Mapper diagrams for $C_1$ at 40 epochs of training, density filtration $p = .05$................................................................. 117
8.19 Persistence and Mapper diagrams for $C_1$ at 20 epochs of training, density filtration $p = .05$................................................................. 118
8.20 Persistence and Mapper diagrams for $C_1$ at 50 epochs of training, density filtration $p = .05$................................................................. 118
8.21 Persistence and Mapper diagrams for $C_1$ at 5 epochs of training, density filtration $p = .1$. 119
8.22 Persistence and Mapper diagrams for $C_1$ at 35 epochs of training, density filtration $p = .1$. 119
8.23 Persistence and Mapper (30 cubes) diagrams for $C_1$ at 40 epochs of training, density filtration $p = .1$. 120
8.24 visualizations of four edge-detecting kernels constructed from taking the means of all 9-dimensional data points in four regions of the Mapper shown in Figure 8.20 (right), taken from the standard convolution layer of our 518 CNNs with a $p = .05$ density filtration. 120
Chapter 1

Introduction

In the field of machine learning, in particular, the subfield of deep learning, a popular model for predictions related to image inputs is the Convolutional Neural Network (or CNN). The introduction of the CNN was a major step in the development of machine learning algorithms aiming to process and make predictions based on image data. The CNN, as well as the larger class of predictive models it belongs to, the artificial neural networks (ANNs or NNs), while known for being able to fit a wide variety of functions and having a significant scalability advantage over many other predictive models, are notoriously difficult to analyze and understand holistically.

Once a NN’s architecture is set (as described in the following chapter), the model is fit, or trained, by initializing a set of random parameters, then, based on how it performs on training data, making small changes to these parameters through an optimization process, generally some variation on gradient descent, which we will expand upon in the following chapter. However, there have been some major innovations in the training process of NNs that have complicated the analysis of them as a predictive model. Examples we will address include stochastic gradient descent, batch normalization, and $L_1$ or $L_2$ regularization. Such additions have shown to have advantages including decreasing the amount of time needed to train a model and enhancing its ability to perform on previously unseen data. However, there are some interesting places where such techniques can go wrong; for example, stochastic gradient descent has the potential to cause instability in the training process of a network. Currently, there is no standard theoretical procedure for analyzing the performance of the training process, so it tends to be judged only empirically, testing it directly on new data points and calculating a loss or cost metric.

A paper essential in motivating the questions resulting eventually in the CNN was by neuroscientists Hubel and Wiesel [16], discussing the primary visual cortex (or V1). This region belongs to the larger mammalian visual cortex (or MVC), and acts as the base-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. Especially pertinent to future applications in machine learning was the finding that these cells detect edges and lines. Further layers of the MVC are capable of performing more abstract tasks in the visual process.

Fukushima would later develop the Neocognitron neural network [11], consisting of neurons referred to as S-cells and C-cells. These parts of the network were intended to replicate the action of, respectively, the simple and complex neurons found in Hubel and Wiesel’s research. As the years passed and machine-learning advances were made, these S-cells and C-cells developed into the two fundamental processes in CNN models, known respectively as convolution layers and pooling operations. We see this comparison visualized in Figure 1.1.

In a neural network model, inputs are processed to generate predictions through a series of computational steps, called layers, which we will break down in the following chapter. A CNN is then characterized by the types of computations occurring in its preliminary layers, known, as mentioned above, as convolutional layers. In a common CNN structure, the aforementioned pooling layers following convolutional layers look to create a lower-dimensional projection of the information contained in
the convolutional layer’s output. With respect to the convolutional layers, one might want to see how similar the convolution process is to that of the primary visual cortex, i.e. whether the parameters contained conform basic line and edge patterns in images. As is the focus of our work, one method to exploring this question comes in the form of applying techniques from Topological Data Analysis (TDA).

We can trace the use of topology with image data to an earlier work by Lee, Mumford, and Pedersen [23], which investigates the distribution of 3x3-pixel patches within a natural image dataset. They found that high-density collections of these patches pointed to an interesting nonlinear geometry: in particular, their findings invited the use of topological manifolds to understand the image patch distribution. Gunnar Carlsson applied the TDA techniques of persistent homology and Mapper analysis to the image patches in [7], which directly yielded cyclical patterns suggesting an underlying distribution similar to a curious geometric object: the Klein bottle. Carlsson and Richard Gabrielsson’s analysis of CNNs in [12] and [6] using similar techniques uncovered a similar distribution, suggesting that the CNNs they studied were, in a sense, learning topological information, in particular the first homology group of the image patch space through their trained parameters.

Our work aims to expand this use of TDA to an alternate CNN construction, with layers each employing a depthwise separable convolution, or simply a separable convolution. This construction requires fewer parameters to be trained and requires fewer multiplication/addition operations than the CNNs discussed in the work of Carlsson and Gabrielsson, which instead use a more prevalent convolution technique we refer to here as a standard convolution. The standard convolution will be introduced in the following chapter, and the separable convolution will be introduced in Chapter 3.

In this dissertation, we describe the fundamentals of these CNN models, detail the foundations of algebraic topology needed to motivate and develop the persistent homology and Mapper analysis tools, discuss prior work involving application of these TDA tools to a collection of CNNs only standard convolutions in its convolutional layers, and finally report our own findings involving the application of these same techniques to CNNs with primarily separable convolutions. We trained a collection of such separable-convolution CNNs on a set of chest X-Rays and found that, despite the smaller number of parameters, they are able not only to perform well on our chosen dataset, but also have the capability to learn similar topological information to their standard-convolution counterparts.
Chapter 2

Neural Network Overview

Artificial neural networks are a prominent object study in the field of deep learning, a subfield of machine learning. These are multi-layer parametric models that take in some input array, possibly multidimensional, and apply a series of transformations, resulting in an output, or prediction. This output may take the form of a single number or a list of multiple values. In particular, the type of networks centered on in this dissertation are classification networks. We will review prior work carried out by researchers Gunnar Carlsson and Richard Gabrielsson that return a vector of 10 real values summing to 1; these 10 values correspond to probabilities that an input belongs to each of 10 potential classes. On the other hand, the networks we have studied directly are binary classification networks, which return a single probability between 0 and 1: in this case there are two potential classes, and output probabilities close to 0 sort the corresponding input into one class while probabilities close to 1 sort the corresponding input into the opposing class. Each step, including the input and output, is referred to as a layer. These transformations are determined by a set of parameters at each layer, and the total number of parameters in a complete model can often number in the millions. These steps are expanded upon from a linear algebra perspective in Section 2.1.1.

The parameters in a neural network are initialized randomly, then a set of labeled data points are given to the random model input vectors; this set of vectors is known as the training dataset. Once these input vectors have been processed by the untrained network, the predictions for each vector as generated by the network are compared to the labels. Such a comparison is done through a loss (or error) function, and the model’s parameters are each shifted slightly, with the object of decreasing (and ideally minimizing) the loss function, through the process of gradient descent. This process is detailed in Section 2.2.3. The shifts to be applied to each parameter are calculated from the gradient of the loss function, and computing the gradient is done through a procedure called backpropagation.

We will expand on the process summarized above, and what types of operations are present inside the neural network itself, by introducing the feed-forward neural network (FFNN). Note that all networks we discuss directly in this dissertation will be of this feed-forward type. Further details, in addition to the fundamentals we describe in this chapter, can be found in [27].

2.1 Introducing the Feed-Forward NN

A neural network may be represented as a directed graph arranged into several disjoint subsets of nodes, arranged into columns from left to right, as pictured in Figure 2.1. Each such subset represents to a collection of values called a layer, and each edge corresponds to an operation, as specified later, being performed on its initial node of this edge to contribute to the value of its terminal node. This figure represents the skeleton of a feed-forward and fully-connected neural network: in particular,

- a feed-forward neural network’s graph has edges that point only to the right, and most notably, there are no cycles (in contrast to another type of model called the recurrent neural network, or RNN), and
in a fully-connected neural network graph, each node (except those in the rightmost column) is connected via an edge to all nodes in the column directly to its right.

Figure 2.1: 2-node input layer, 2 3-node hidden layers, single-node output layer

We now specify the nature of these layers in the model.

**Definition 2.1.1 (Layers).** In a neural network, a layer (which we will notate $L^{(i)}$) is a vector in $\mathbb{R}^{n^{(i)}}$, where $i \in \{1, 2, \ldots, m\}$, with $m$ being the number of layers in the network. In particular, if $1 < i \leq m$, then $L^{(i)}$ is a function of $L^{(i-1)}$ as determined by a set of parameters $\theta$. The first (leftmost) layer is known as the input layer, the last (rightmost) layer is known as an output or readout layer, and the layers in between are called hidden layers.

As we will see, this characterization of the layers as vectors is very intentional, as a foundational computation used to determine the values of $L^{(i)}$ from $L^{(i-1)}$ is a linear transformation, in particular multiplication with a $n^{(i)} \times n^{(i-1)}$ matrix of parameters. We follow with a more description of feed-forward neural networks, then move to an intersecting class of networks, the convolutional neural network (or CNN), our major object of study.

### 2.1.1 General Architecture of the FFNN

Let $L^{(i)}$ represent each the $i^{th}$ layer of an $m$-layer FFNN, with $L^{(1)}$ as the input layer and $L^{(m)}$ as the output layer. Additionally, let $n^{(i)}$ represent the number of nodes in the $i^{th}$ layer and $L^{(i)}_{j}$ represent the $j^{th}$ node in the $i^{th}$ layer. As mentioned previously, Figure 2.1 is an example of a fully connected neural network, as each node in $L^{(i)}$ is connected by an edge to each node in $L^{(i+1)}$ for all $i \in \{1 \ldots m - 1\}$. If one or more of these connections between two nodes of $L^{(i)}$ and $L^{(i+1)}$ were to be deleted, we would say that $L^{(i+1)}$ is not fully connected.

Now that we have graph representation and general description of a small example network, let’s break down the forward propagation process of a neural network, i.e. the processing of a single unlabeled
data point, starting with the input layer.

**Definition 2.1.2** (Forward propagation). *Forward propagation* of a neural network is the process, starting with an \( \mathbb{R}^{n^{(1)}} \) vector, of populating all of the network’s nodes from left to right, finishing with the output, contained in the layer \( L^{(m)} \).

In this process, note that the number of nodes in the input layer **must** be set to equal the number of features in the dataset, so that we can directly input an observation to the first layer, entry by entry; thus we have \( n^{(1)} \) features. When forward propagation is completed for a single observation, each node of the network will contain a number, and each layer will have been filled in order from left to right. Recall that an edge drawn to connect two nodes in two consecutive layers indicates that the value in the node on the edge’s left side is used to calculate the value in the node on the edge’s right side. Thus, in a fully connected neural network, each node in \( L^{(i)} \) is used to calculate each node in \( L^{(i-1)} \) for all \( i \in \{2 \ldots m\} \). The size \( n^{(i)} \) of each hidden layer (i.e. with \( i \in \{2 \ldots m-1\} \)) may be determined completely at will (subject to computational time and cost), as a functional neural network can theoretically be built with hidden layers of any combination of sizes, regardless of the structure of the data or labeling method.

Similar to how the input structure determines the number of nodes \( n^{(i)} \) to use in the input layer, the output structure (as determined by the type of prediction we wish to carry out via the network) determines the number of nodes \( n^{(m)} \) to use in the output. For example, the binary classification (recall: 2-class categorization) neural network is of type \( n^{(m)} = 1 \), since the final layer is to contain only a single probability. By contrast, Carlsson and Gabrielsson’s 10-class neural networks are of type \( n^{(m)} = 10 \).

We now move on to describe the full process of using the nodes of layer \( L^{(i)} \) to populate the successive layer \( L^{(i+1)} \): first a linear transformation is applied to \( L^{(i)} \), and this is followed by a non-linear (or identity) activation function applied to each of the nodes before continuing with any successive linear transformations. As we will see, there are additional steps which may be introduced, which allows for a vast degree of customization when building a neural network.

### 2.1.2 Inter-Layer Calculations: Linear Transformations

The standard process of calculating layer \( L^{(i+1)} \) from layer \( L^{(i)} \) first involves a linear transformation. This linear transformation is characterized by a \( n^{(i+1)} \times (n^{(i)} + 1) \) matrix of weights and biases, and is applied, more precisely, to a slightly modified version of \( L^{(i)} \).

**Notation 2.1.3.** We notate the vector \((1,L^{(i+1)}_1,L^{(i+1)}_2,\ldots,L^{(i+1)}_{n^{(i)}})\) as \( L^{(i)*} \).

**Definition 2.1.4** (weights, biases, trained parameters). In a neural network, *weights* are values in \( \mathbb{R} \) to be multiplied directly by node values of \( L(i) \) when computing \( L(i+1) \). We will notate the weight applied to \( L^{(i)}_k \) when computing \( L^{(i+1)}_j \) as \( w^{(i)}_{j,k} \), i.e. \( L^{(i+1)}_j = w^{(i)}_{j,k} \). *Biases*, by contrast, contribute to the linear transformation on \( L^{(i)*} \) but are not multiplied by node values from \( L^{(i)} \), and we will notate the bias applied to \( L^{(i)*} \) in computing \( L^{(i+1)}_j \) as \( b^{(i)}_j \).

Both weights and biases fall under the general class of *trainable parameters* in a neural network. Trainable parameters refer to any parameter in the neural network which may be updated as the network processes input from the training dataset; often, the weights and biases in particular will be initialized randomly before any data is processed by the network. Note that such parameters are included in \( \theta \), the object storing all parameters of the network.

The aforementioned weight-bias matrix for \( L^{(i)} \), using the notation from Definition 2.1.4, is of the form
\[
\mathbf{w}^{(i)} = \begin{pmatrix}
\mathbf{b}_1^{(i)} & w_{1,1}^{(i)} & w_{1,2}^{(i)} & \cdots & w_{1,n^{(i)}}^{(i)} \\
\mathbf{b}_2^{(i)} & w_{2,1}^{(i)} & w_{2,2}^{(i)} & \cdots & w_{2,n^{(i)}}^{(i)} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
\mathbf{b}_{n^{(i)+1}}^{(i)} & w_{n^{(i)+1},1}^{(i)} & w_{n^{(i)+1},2}^{(i)} & \cdots & w_{n^{(i)+1},n^{(i)}}^{(i)}
\end{pmatrix},
\tag{2.1}
\]

and the result of the matrix-vector multiplication \(\mathbf{w}^{(i)}L^{(i)}\) is then a base point to work from when populating \(L^{(i+1)}\). We will refer to the result of this operation as the \textit{pre-activation} node vector \((z_1^{(i+1)} \ldots z_{n^{(i)+1}}^{(i+1)})\) for \(L^{(i+1)}\). In particular, for all \(k \in \{1 \ldots n^{(i)}\}\) and \(j \in \{1 \ldots n^{(i)+1}\}\), we have

\[
z_j^{(i+1)} = \sum_{k=1}^{n^{(i)}} w_{j,k}^{(i)} a_k^{(i)} + b_j^{(i)},
\tag{2.2}
\]

where \(a_k^{(i)}\) is the post-activation value of \(L_k^{(i)}\).

Given a weight for each edge of Figure 2.1, we may use this linear operation to populate this simple network’s second layer with pre-activation values. Note that we will need to apply an additional transformation (the \textit{activation function}, as described in the next section) to this second layer before continuing with our computations and finishing off our population of the network. The result of such a linear transformation applied to the first layer of a neural network is pictured in Figure 2.2. Note the addition of a node with a value of 1 with no connections to the first layer in order to represent in the bias of .4 that will be used in calculating the 1-node output layer of the network.

![Figure 2.2](image-url)

Figure 2.2: Network from Figure 2.1 with weights and nodes filled in before the activation step, described in Section 2.1.3.

We now describe the final major step before \(L^{(i+1)}\) is finalized: applying the activation function.

### 2.1.3 Intra-Layer Calculations: Activation Functions

As we will see, introducing non-linearity is an essential part of enabling a neural network to accommodate a flexible range of predictive modeling situations. On a data-point level, i.e. during a single iteration of
forward propagation, this non-linearity is accomplished via the activation function.

**Definition 2.1.5** (activation function). For a node $L_j^{(i)}$ in a neural network, the activation function $g^{(i)} : \mathbb{R}^n(i) \rightarrow \mathbb{R}^n(i)$ defines the post-activation vector $a^{(i)} = g^{(i)}(z^{(i)})$ of the $j^{th}$ node of $L(i)$.

We will refer to the function defining the post-activation value $a_j^{(i)}$ of the node $L_j^{(i)}$ as $g_j^{(i)} : \mathbb{R}^n(i) \rightarrow \mathbb{R}$.

We use $z^{(i)}$ to describe the vector of pre-activation values of the nodes in $L(i)$, and we call the pre-activation value of its $j^{th}$ node $z_j^{(i+1)}$. Note that each node will have an activation function. (even if this activation is chosen to be the identity function): let $g_j^{(i)}$ be the activation function for the $i^{th}$ layer’s $j^{th}$ node and let $a_j^{(i)} = g_j^{(i)}(z_j^{(i)})$. A standard choice for this activation function is the Rectified Linear Unit, or ReLU, function $r(x) = \max(x, 0)$.

Another activation function that has been widely used is the sigmoid function $s(x) = \frac{1}{1+e^{-x}}$, but it has been found that many iterations of this activation function throughout a neural network can cause the standard model-fitting process of gradient descent to become impractically slow. More details on gradient descent and this will be discussed later, but this vanishing gradient problem is beyond the scope of this dissertation. One context where one can often expect to find sigmoid activation is in the output layer of a binary classification network, including the networks we have used for TDA work. A clear reason for using such an activation is that $s(x)$ is a bijection between $\mathbb{R}$ and $(0,1)$, making it an ideal choice when the goal is to output an estimate between 0 and 1.

As an aside, there are activation functions that operate using multiple nodes in a layer as an input. This property is present in the softmax activation function $s_k : \mathbb{R}^k \rightarrow \mathbb{R}^k$, where we write $s_k(x) = (s_k(x_1 \ldots x_k)_k$, and define

$$s_k(x_1 \ldots x_k)_i = \frac{e^{x_i}}{\sum_{j=1}^k e^{x_j}} \quad (2.3)$$

for $i \in \{1 \ldots k\}$. In particular, we will see this function used in the final layer of a multi-class neural network, with examples including the 10-class networks analyzed by Carlsson and Gabrielsson. Since the output of this function is a vector in $\mathbb{R}^k$ with entries that add to 1, we can interpret the output nodes in this layer as being a list of percentage confidence levels that an input falls into each of $k$ class.

Figure 2.3 graphs several commonly used single-node activation functions:

- the sigmoid function $g(x) = \frac{1}{1+e^{-x}}$ (top left),
- the ReLU (rectified linear unit) function $g(x) = \max(0, x)$ (top right),
- the hyperbolic tangent or tanh function $g(x) = \frac{e^x-e^{-x}}{e^x+e^{-x}}$ (bottom left), and
- the linear or identity function $g(x) = x$.

Note that these are nondecreasing functions. This gives us an important interpretation for the use of these particular activation functions. If a large value enters a node (pre-activation), we can consider this analogous to a highly active neuron, and so a relatively large value should also leave the node (post-activation), and vice versa.

Of the three nonlinear activation functions pictured above, the ReLU is unique in that it has a "threshold" interpretation for a neuron’s degree of activation. If a non-negative number enters a node with a ReLU activation function, then the exact same value leaves the node post-activation, to be used in further network calculations. However, if a negative value enters this node, then it is sent to zero. In this way, we can think of a node with a ReLU activation function as "inactive" when fed a negative number and "active" when fed a positive value.

On the other hand, the sigmoid and tanh activation functions work in a very similar manner to one another. Both functions send highly positive pre-activation values to post-activation values near
Figure 2.3: visualizations of several commonly-used single-node activation functions.
1 ("highly active"), and both send highly negative pre-activation values to post-activation values near some other number (0 for the sigmoid, -1 for the tanh, we might call these "highly inactive"). For pre-activation values closer to 0, there is a smooth interpolation between these two extremes, representing a more "moderately active" neuron.

2.1.4 Synthesizing these Steps into a Cohesive Model

Further calculations in the forward propagation process continue iteratively: for each node $L_j^{(i)}$ in the network (minus the nodes in the output layer), we apply the following two steps.

1. $z_j^{(i+1)} = \sum_{k=1}^{n^{(i)}} w_{j,k} a_k^{(i)} + b_j^{(i)}$
2. $a_j^{(i+1)} = g_j^{(i+1)}(z_j^{(i+1)})$

Once $a_j^{(m)}$ is calculated for all $j \in 1 \ldots n^{(m)}$, then these values ($a_1^{(m)}, \ldots, a_{n^{(m)}}$) are taken as the output vector for forward propagation for the input values $(x_1, \ldots, x_{n^{(1)}})$. A completed diagram of the Figure 1 network is shown in Figure 2.4, using a ReLU activation function in the second and third (output) layers.

![Figure 2.4: Figure 2.1's network with all values filled in for the nodes and weights, post-activation where each non-input value has a ReLU function applied.](image)

The networks we analyzed using TDA techniques also have a single output node, but its value is restricted to the interval $(0, 1)$ by the sigmoid function. If we were to apply this sigmoid activation to the Figure 2.4 output node, the resulting value would be $\frac{1}{1+e^{-1.5}} \approx 0.8176$.

Neural networks can be used to model both classification and regression problems. In future sections, we will be focusing on the classification case, as the networks we observe will be built to classify images. The networks we focus on in our TDA work, as mentioned before, are binary classification networks, and are trained on a collection of chest X-Ray images in order to sort them into pneumonia and non-pneumonia classes. Additionally, there are popularly-used datasets for more general classification networks; in particular, the MNIST [9] dataset is a collection of hand-drawn digits (assigned to one of ten categories, 0 through 9), and the CIFAR-10 [21] dataset is a collection of images each representing one...
of ten categories: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. For classifying images of this type (where the input layer consists of pixel values in an image), the output layer will have 10 nodes, each representing “confidence” that an image will fall into each category: As such, the activation function used for this output layer will be the softmax function $s_{10}$.

## 2.2 Training and Validating a Neural Network

### 2.2.1 Neural Networks as a Parametric Predictive Model

Neural networks are a complex type of predictive model, and specifically, the neural networks we will discuss in this paper are parametric models. Such parametric models are functions $f_\theta : X \rightarrow Y$, along with a method for defining its performance on a dataset given a parameter set $\theta \in \Theta$, where $\Theta$ is the parameter space of the model characterized by the choice of $f_\theta$. Such a dataset consists of individual observations, or labeled data points, which are pairs from $X \times Y$; a member of $X$ specifies the input features for each observation, and a member of $Y$ specifies the target or label for each observation.

In all examples of such models that follow in this dissertation, $X = \mathbb{R}^k$, and the input $(x_1 \ldots x_k)$ is a vector of $k$ input features used to predict a target variable $y \in Y$. The goal is then to produce a reasonable estimate $\hat{y} = f_\theta(x_1 \ldots x_k)$ for a given data point with predictor values $x_1 \ldots x_k$ and label/target value $y$.

**Remark 2.2.1.** For a traditional image recognition neural network, as we will describe in future sections, $k$ represents the number of pixels in the input image (multiplied by the number of color channels if this image is not grayscaled), $x_i$ denotes an intensity measure for the $i^{th}$ pixel, and $y$ represents the category the picture is assigned to.

Finally, $\Theta$ has a large potential for variability depending on the type of model being discussed: for example, as we will see, linear regression models choose their parameters from $\mathbb{R}^{k+1}$ (with $k$ again being the number of input features), while for neural network models, $\Theta$ is instead a Cartesian product of matrix spaces $\mathbb{R}^{n(i+1) \times n(i)}$, where the $i^{th}$ matrix in an element of $\Theta$ specifies the collection of parameters to use in the layer $L^{(i)}$.

**Remark 2.2.2.** The $i^{th}$ matrix in an element of $\Theta$ is $n^{(i+1)} \times (n^{(i)} + 1)$, so the total number of weights and biases in a neural network with $m$ layers is

$$\sum_{i=1}^{m-1} (n^{(i)} + 1)n^{(i+1)}.$$

Two major types of general predictive models (and in particular neural networks) are classification and regression models. Classification models are those fit to data points labeled with one of a finite number of classes $c$. A standard method for encoding such a category is one-hot encoding, where, for an observation $(x, y)$ in the $i^{th}$ class, $y \in \{0, 1\}^c$ and contains zeroes in every entry with the exception of a 1 as its $i^{th}$ entry. For a classification model with $c$ potential classes, $Y = \{(y_1 \ldots y_c) \in [0, 1]^c | \sum_{i=1} y_i = 1\}$. For a binary classification problem, such as the one for which we fit our CNNs, we can equivalently say $Y = \mathbb{R}$, since all points in $\mathbb{R}^2$ satisfying the constraints for $Y$ are of the form $(y, 1 - y)$. In both cases, the output of the predictive model is understood as a vector of probabilities assigning a confidence value that an observation with predictors $x_1 \ldots x_k$ is a member of each class $i \in \{1 \ldots k\}$. All examples of neural networks described in this dissertation will be such classification networks. On the other hand, regression models are of the form $Y = \mathbb{R}^l$ (in many cases $Y = \mathbb{R}$), and their goal is to predict a measurable value rather than to place an object into one of a finite number of classes.

We will now compare the fitting, or training, process a neural network to that of the classic linear regression

$$f_\theta(x_1 \ldots x_k) = \theta_0 + \theta_1 x_1 + \ldots + \theta_k x_k.$$  \hspace{1cm} (2.4)
As we will see, training a neural network is a complex and sometimes imprecise process when compared to that of the linear regression, a simpler and more easily and directly interpretable model. To illustrate this, we will directly compare and contrast how general neural networks and linear models process a set of data to arrive at a fitted model \( f(x_1 \ldots x_k) \approx y \).

The fundamental similarity between linear models and neural networks (besides the use of linear transformations in neural networks) is in their status as parametric predictive models: again, the motivation in using both is to find an optimal selection of parameters \( \theta = (\theta_1 \ldots \theta_p) \) to be used in the model \( f_{\theta}(x_1 \ldots x_k) \approx y \). Thus, we need to define what it means for a parameter selection \( \theta \) to be optimal, i.e. for such a model to produce estimates \( \hat{y} \) close to the true target values \( y \) across a training dataset. This is done by minimizing a loss function \( J(\theta) \), which uses the predictors \( x = (x_1 \ldots x_k) \) and target \( y \) from each observation in the data as fixed values.

**Definition 2.2.3** (loss, loss function). The loss associated with the \( i^{th} \) observation from a dataset of \( n \) observations, given a set of parameters \( \theta \), will be notated as \( j_i(\theta) \), and the loss function associated with the entire dataset for \( \theta \) is

\[
J_{x,y}(\theta) = \sum_{i=1}^{n} j_i(\theta).
\]

Elsewhere, we shorten the notation \( J_{x,y}(\theta) \) to \( J(\theta) \).

### 2.2.2 Loss Functions For Regression and Classification Models

#### A Simple Example: OLS Regression

The goal of our loss function \( J(\theta) \) is to measure the model’s performance for a particular parameter choice \( \theta \), and in the usual formulation, large values of \( J(\theta) \) translate to poor choices of \( \theta \) for the model. Thus, under this definition, finding an optimal model is a question of choosing a \( \theta \) that minimizes this loss function. For example, a common choice for \( J(\theta) \) when fitting a linear model is the sum of squared errors

\[
SSE = \sum_{i=1}^{n} (f_{\theta}(x_{1i} \ldots x_{ki}) - y_i)^2.
\]

or the mean squared error

\[
MSE = \frac{1}{n} \sum_{i=1}^{n} (f_{\theta}(x_{1i} \ldots x_{ki}) - y_i)^2,
\]

where \( n \) is the number of observations used to fit the model, \( x_{ji} \) is the \( j^{th} \) predictor from the \( i^{th} \) observation, and \( y_i \) is the target variable from the \( i^{th} \) observation. Note that the parameter selection for minimizing these two choices of \( J(\theta) \) will be the same, as they differ only by a factor of \( n \).

Using this strategy for determining a properly fitting linear model is known as Ordinary Least Squares (OLS) linear regression, and there is a known directly-computable solution, given a set of \( n \) observations each consisting of a set of predictors \( (x_{1i} \ldots x_{ki}) \) and a target value \( y_i \) to be predicted. The existence of this directly computable solution points to the main difference in the training processes between a linear regression and a neural network. For an OLS linear regression, this is a convex problem with a directly computable solution via the normal equation.

\[
\theta = (X^TX)^{-1}(X^Ty).
\]

In Equation (2.8) above, \( X \) is a matrix of predictors and \( y \) is a vector of labels. In particular,

\[
X = \begin{pmatrix}
1 & x_{1,1} & x_{1,2} & \cdots & x_{1,k} \\
1 & w_{2,1}^{(i)} & w_{2,2}^{(i)} & \cdots & w_{2,k}^{(i)} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & w_{n,1}^{(i)} & w_{n,2}^{(i)} & \cdots & w_{n,k}^{(i)}
\end{pmatrix},
\]

(2.9)
where \( x_{i,j} \) is the \( i^{th} \) observation’s \( j^{th} \) predictor value, and

\[
y = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix},
\] (2.10)

where \( y_i \) is the label for the \( i^{th} \) observation.

When we use a nonlinear activation function (such as the sigmoid, ReLU, or tanh), we are introducing a necessary part of the neural network structure that separates it functionally from a linear regression model. Since composition of multiple linear maps creates a single linear map, a neural network whose activations were only linear would itself represent a linear transformation of the input. Thus, since we can directly calculate optimal parameters for Ordinary Least Squares regression, creating a regression neural network and fitting it using an approximation method such as gradient descent would be useless, at least with regard to a Mean Square Error loss function.

### Classification Loss Functions

Recall that the neural networks in our work are for a binary classification problem, and so their output is a single probability associated with class 1, as opposed to class 0. A common choice for \( J(\theta) \) in this context (and the one we use) is then the binary cross-entropy function

\[
J(\theta) = -\sum_{i=1}^{n} [y_i \ln(\hat{y}_i) + (1 - y_i) \ln(1 - \hat{y}_i)].
\] (2.11)

Here \( y_i \) is 1 if the \( i^{th} \) observation is actually in the \( j^{th} \) class, and 0 otherwise (regardless of the model’s prediction). \( \hat{y}_i \) is a number between 0 and 1 representing the model’s confidence that the \( i^{th} \) observation is class 1. Thus, this loss function particularly penalizes low-confidence predictions for the correct class.

While it turns out that optimizing this loss function is also in fact a convex problem, there is no closed-form solution for its minimum. Thus, such a binary classification problem necessitates using an approximation method such as gradient descent (described in Section 2.2.3) or a higher-order method such as Newton-Raphson (not detailed or used in this dissertation).

We will also be encountering networks discussed in previous works involving TDA approaches to NN study; these will also be classification networks, but each of their inputs (a member of either the MNIST or CIFAR-10 dataset as discussed earlier) is to be placed into one of ten different categories. For a more general classification problem, one can extend the above binary cross-entropy function to the multi-class cross-entropy function

\[
J(\theta) = -\sum_{i=1}^{n} \sum_{j=1}^{c} y_{ij} \ln(\hat{y}_{ij}).
\] (2.12)

Here \( y_{ij} \) is 1 if the \( i^{th} \) observation is in the \( j^{th} \) category, and 0 otherwise; and \( \hat{y}_{ij} \) is a number between 0 and 1 representing the model’s confidence that the \( i^{th} \) observation is in the class \( j \). Like the binary cross-entropy function, this loss function particularly penalizes low-confidence predictions for the correct class, as the summed terms for all incorrect classes are 0.

### 2.2.3 Backpropagation and Gradient Descent

For a neural network, the fitting, or training process is not as simple or precise of a process as OLS regression, despite the network’s usage of linear transformations from layer to layer. Due to the addition of nonlinear activation functions within its layers, the model \( f_{\theta}(x_1 \ldots x_k) \) is significantly more complex and flexible than in the linear model case. Accordingly, the process of minimizing our choice of \( J(\theta) \)
generally takes a different approach, as, especially for large neural networks with many layers and varied computations, it is not reasonable to attempt directly computing a global minimum for \( J(\theta) \).

**Remark 2.2.4.** As we will see, such a process will be based on approximations rather than direct computation of an optimal value. These approximations will be based on repetitive iterations of forward propagation, with each iteration taking in a *batch*, or subset, of the training dataset. Each such iteration is followed by an update of all trained parameters in the neural network, typically resulting in an improved value of \( J(\theta) \).

This is where the ideas of *deep learning* truly take root, as these updates are taking place with respect to a large number of parameters, easily numbering in the millions. When we fit a neural network to a dataset, instead of directly computing an optimal choice of parameters, we start the model from a randomly generated selection of \( \theta \) and use a *gradient descent* algorithm to adjust our parameters and guide them toward a significantly improved value of \( J(\theta) \), perhaps approximating a local minimum. For this descent process, it is more helpful to consider \( \theta \in \mathbb{R}^p \) to be a vector listing every trained parameter in our network, rather than the list of matrices considered earlier.

**Definition 2.2.5 (gradient descent).** A *gradient descent* algorithm is a sequence \((\theta_1, \theta_2, \ldots)\) in \( \mathbb{R}^p \) and where, for \( i \in \{1, 2, \ldots\} \), a function \( J: \mathbb{R}^p \to Y \), and a sequence \( \alpha_i \) in \( \mathbb{R} \),
\[
\theta_{i+1} = \theta_i - \alpha_i \nabla J(\theta_i).
\] (2.13)

**Remark 2.2.6.** The sequence \((\alpha_i)\) is a set of *hyperparameters*, i.e. preset values that do not change during training, with each \( \alpha_i \) known as the *learning rate* at descent iteration \( i \). We may have \( \alpha_i = \alpha \) for some \( \alpha \in \mathbb{R} \), i.e. constant throughout the learning process. Alternatively (as in our work), \((\alpha_i)\) may be a decreasing sequence in order to reflect finer adjustments in the trained parameters further into training.

This often-used method of updating \( \theta_i \) to \( \theta_{i+1} \) after running forward propagation with a batch of data comes from a basic property of the gradient: the fact that, for small changes in the input, the negative gradient represents the direction of steepest descent for the function in question. For neural networks, we view the loss function as \( J(\theta) \), leaving the data points fed to the network as fixed, and using the \( \theta \) as its multivariate input. The goal is therefore, for each iteration of gradient descent, to add some multiple of \(-\nabla J\) to the parameter vector \( \theta \).

As noted in Definition 2.2.5, using gradient descent necessitates calculating the gradient of the loss function with respect to \( \theta \), i.e. the partial derivative \( \frac{\partial J}{\partial \theta_i} \) for every trained parameter in the network, including all weights and biases.

**Remark 2.2.7.** Weights and biases are two prominent examples of trained parameters in a neural network, but trained parameters are not limited to weights and biases. Later, we will describe a process called *batch normalization* which introduces additional trained parameters.

The process used to calculate these partial derivatives starts in the final, \( m^{th} \) layer of the network and proceeds backward toward the beginning of the network, in accordance with the multivariate chain rule, and is thus called *backpropagation*. As mentioned previously, the networks we will discuss in the rest of this dissertation are classification networks with a sigmoid or softmax activation function in the final layer and a binary or multi-class cross-entropy loss function, so this is the framework we will be using to discuss backpropagation. The following outline of backpropagation will use several details and conventions from [22], with notations changed to fit our previous discussions, for instance using \( m \) to represent the final layer rather than \( L \) and \( J \) to represent the loss function rather than \( C \).

We now present the first step of backpropagation followed by a breakdown of notation. Note that for the rest of this backpropagation section, we will be referring to \( J \) when denoting the loss associated
with a single observation of data. When running an iteration of backpropagation and gradient descent on a full batch of data, the loss function will be simply the sum of all loss values for the batch, and thus the gradient of the loss function will be, similarly, the sum of all gradient values for the batch.

\[
\delta^{(m)} = \nabla_{a^{(m)}} J \odot \sigma'(z^{(m)}) \tag{2.14}
\]

The symbol \( \delta^{(m)} \) refers to a measure of error associated specifically with the output layer, specifically the vector of partial derivatives \( \left( \frac{\partial L}{\partial z_1^{(m)}}, \ldots, \frac{\partial L}{\partial z_n^{(m)}} \right) \). Recall we have used \( n(i) \) as the total number of nodes in the \( i \)th layer and \( z_j^{(i)} \) to describe the preactivation value in the \( j \)th node of the \( i \)th layer. As a shorthand we will write \( z^{(i)} = (z_1^{(i)}, \ldots, z_n^{(i)}) \). The right side of this equation represents the gradient of the loss function \( J \) with respect to the vector \( a^{(m)} \) of post-activation values in the nodes of the output layer, multiplied elementwise (denoted by the Hadamard product \( \odot \)) by the vector of partial derivatives of the activation function \( \sigma \) with respect to the final layer’s pre-activation vector of nodes \( z \). From this point forward, we will use \( \hat{y} \) in place of \( a^{(m)} \) to represent the fact that this is the vector of final estimates produced by the neural network for an input \( x \).

Note that for a binary classification network with sigmoid activation, we have only one output node. Thus we can note a convenient property of the sigmoid function, the fact \( \sigma \) that \( \sigma'(z) = \sigma(z)(1 - \sigma(z)) \), and thus \( \sigma'(z^{(m)}) = (a^{(m)})(1 - a^{(m)}) \) or, equivalently, \( \hat{y}(1 - \hat{y}) \) for the model’s probability estimate \( \hat{y} \) that the given observation falls into class 1. Additionally, given a binary cross-entropy loss function, we have

\[
\frac{dJ}{d\hat{y}} = \frac{1 - y}{1 - \hat{y}} - \frac{y}{\hat{y}} \tag{2.15}
\]

i.e. \( -\frac{1}{\hat{y}} \) if the true labeled class \( y = 1 \) and \( \frac{1}{1 - y} \) if \( y = 0 \). Therefore, applying the (in this case, univariate) chain rule

\[
\delta^{(m)} = \frac{dJ}{dz^{(m)}} = \frac{dJ}{d\hat{y}} \cdot \frac{d\hat{y}}{dz}, \tag{2.16}
\]

we have

\[
\delta^{(m)} = (1 - y)\hat{y} - y(1 - \hat{y}), \tag{2.17}
\]

i.e. \( \hat{y} \) if \( y = 0 \) and \(- (1 - \hat{y}) \) if \( y = 1 \). In particular, we get a positive partial derivative if \( y = 0 \), which implies that parameter shifts will aim to decrease \( \hat{y} \) toward 0. On ther other hand, we get a negative partial derivative if \( y = 1 \), and so parameter shifts are likely to increase \( \hat{y} \) toward 1.

The softmax activation in a multi-class neural network makes for a more general situation, as each output node of the softmax function requires an input from every pre-activation node. Thus, to start backpropagation with a softmax activation in the final layer, we need to employ the multivariable chain rule as follows to calculate \( \delta^{(m)} \).

\[
\frac{\partial J}{\partial z_1^{(m)}} = \sum_{j=1}^{n} \frac{\partial J}{\partial \hat{y}_j} \frac{\partial \hat{y}_j}{\partial z_1^{(m)}} \tag{2.18}
\]

Recall that the multi-class cross-entropy loss function for a single data point is equal to \( ln(\hat{y}_c) \), where \( \hat{y}_c \) is the probability estimate by the network corresponding to the true labeled category \( c \) of the input data point. Thus \( \frac{\partial J}{\partial \hat{y}_c} = \frac{1}{\hat{y}_c} \), and \( \frac{\partial J}{\partial \hat{y}_i} = 0 \) for \( i \neq c \).

Applying the multivariable chain rule as discussed above in Equation (2.18) results in the following:

\[
\delta^{(m)} = \left( \frac{1}{\hat{y}_1}, \ldots, \frac{1}{\hat{y}_c} - \frac{\hat{y}_c}{\hat{y}_c}, \ldots, \frac{1}{\hat{y}_n^{(m)}} \right). \tag{2.19}
\]
Since the softmax function guarantees $0 \leq \hat{y}_i \leq 1$ for $1 \leq i \leq n^{(m)}$, we have a negative partial derivative $\frac{\partial J}{\partial z^{(m)}_i}$ for the correct category $c$ and a positive partial derivative $\frac{\partial J}{\partial z^{(m)}_i}$ for all classes $j$ where $j \neq c$. This is consistent with our intuition of the loss function, as higher confidence in correct category predictions should result in a lower loss function value, similar to the binary case.

Now that we have dealt with the calculation of $\delta^{(m)}$, the error associated with the final layer, for our special multi-class NN architecture, the remaining steps of backpropagation will be much more standard and streamlined. In particular, they will be independent of the number of classes the network uses to sort observations. We can use the following recursive operation to find the error $\delta^{(i)}$ associated with the $i^{th}$ layer.

$$\delta^{(i)} = (\mathbf{w}^{(i+1)})^T \delta^{(i+1)} \odot g'(z^{(i)})$$

(2.20)

Note that we use the current (pre-update) values of the weights spanning $i^{th}$ and $(i+1)^{th}$ layer of the network to traverse from the $(i+1)^{th}$ layer’s nodes to the $i^{th}$ layer, and so we use the transpose of the weight matrix to pair up weights and biases between these two layers with their corresponding pre-activation values $z^{(i+1)}_j$ in the $(i+1)^{th}$ layer. This iterated process carries us from the end of the network to the beginning of the network, and then we conclude our gradient calculation with the following step. The following expression gives us a matrix of partial derivatives corresponding to each entry in the $(l)^{th}$ layer’s weight-bias matrix $\mathbf{w}^{(l)}$.

$$\nabla \mathbf{w}^{(l)} = \delta^{l+1}(a^{(l)})^T$$

(2.21)

Here, $a^l$ is the vector of post-activation values in the $(l)^{th}$ layer. Once this process of calculating the partial derivatives is complete for every weight in the network, the gradient descent process is ready to begin. We restate the calculation to be used for updating below, noting once again that $\theta$ contains all trainable parameters, including all parameters in each of the weight bias matrices $\mathbf{w}^{(l)}$ for each corresponding layer $L^{(l)}$.

$$\theta_{i+1} = \theta_i - \alpha \nabla J$$

2.2.4 Evaluating a Neural Network’s Generalizability: Training and Validation Sets

When observing the effectiveness of a neural network on a particular dataset, we start by partitioning the data into two subsets: one for training and one for testing, also known as a holdout partition or holdout dataset. The training set is what is used to fit the model, while the testing set is left out until the model is fully trained. We then test the accuracy of this fitted model on the unseen testing data to see how well the fitted model generalizes to this new data. Generally, the training set is significantly larger than the testing set, since we want to fit the model to most of the available data before assessing it on unseen data. We might additionally perform a further division of the training set into a true training set and a validation set, but we will stay with the former case for this discussion. Such a validation set might be used to check for a possible overtraining or overfitting of the model to the training set, a phenomenon we will discuss in more detail later. As is the case for the CIFAR-10 image dataset, sometimes the partitioning has been done for us, as this dataset consists of 60,000 images for training and 10,000 for testing.

Once the training and testing sets have been established, and the network’s weights have been randomly initialized, the training process begins. We set a batch size, which refers to the number of data points to be sent through the network before updating the network’s parameters. The first batch of data points (or images, in the case of the image processing networks we will be discussing) is then sent through the network, followed by an iteration of gradient descent and an update in the weights, as detailed in Section 2.2.3. Besides single batch iterations, a larger unit of training is called the epoch. This refers to the amount of training done when the entire training dataset has been processed a single time by the model: neural networks are generally trained for numerous epochs before they are able to perform satisfactorily and consistently even on the training dataset itself.
At the start of training, we specify a number of epochs or batches to quantify the total number of data points to be given to the network. Once this number of training iterations is completed, the model is done training to our specifications. Note that because we are going through a descent process from a randomly initialized model, we have no guarantee that the model has converged to a minimum of \( J(\theta) \), local or global.

After this fitting process is done, we then evaluate the model’s performance on the testing (unseen) data using \( J(\theta) \) or possibly another metric. For classification neural networks (as we’ll be discussing), the natural choice of a performance metric is classification accuracy, i.e. the percent of testing images placed in the correct category. It is customary to track the improvement in accuracy over the course of training, as well as the loss function \( J(\theta) \) with respect to the training set: there are other loss metrics that may be employed, but those are beyond the scope of this dissertation. Figure 2.5 shows an example of a plot of these two training metrics for a classification network training on an X-Ray image dataset.

![Model accuracy and loss plots](image)

Figure 2.5: example accuracy and loss plots for an image-processing CNN taking in training (blue curve) and validation (orange curve) data.

We see that in this plot, the training and validation accuracy has both improved without dropping off in the long run. By contrast, if we see a decrease in validation accuracy during late stages of training, we say that the model is overfitting to the training dataset.

### 2.3 Some Mathematically Puzzling Elements of Training a Neural Network

While gradient descent is a very useful technique for approximating a local minimum of our loss function \( J \) (i.e. the global minimum in the case where \( J \) is a convex function, such as the mean square error or binary cross-entropy loss function), the neural network has seen a number of developments that drastically complicate the training process from a mathematical perspective. We describe some prominent examples in this section. **Stochastic** and **mini-batch** gradient descent have shown to speed up the training process while violating the basic notion of gradient descent that the \( x \) in \( J(x,y) \) should remain consistent throughout all iterations of gradient descent. **Batch normalization** regulates the post-activation values appearing in nodes throughout the network by adding mean and variance parameters for each layer. **Regularization** and **dropout**, on the other hand, regulate the values of the trained parameters themselves by reducing the amount of volatility in parameter changes, each through different means.

Such procedures which complicate direct mathematical explanations for the superior performance of these neural network models are what have led us to use mostly empirical, reactive measures, such as testing accuracy and testing loss metrics, when evaluating the efficacy of these models. The lack of an objective theoretical measure independent of how a model performs specifically on its training and test datasets has led to the use of exploratory measures such as TDA when investigating parameter patterns appearing in trained networks.
2.3.1 Stochastic and Mini-Batch Gradient Descent

If we use the entire training dataset as a single batch, i.e. update the weights through gradient descent precisely once per epoch, we are carrying out batch gradient descent. This is the only way to guarantee that the entire dataset’s true loss function \( J(\theta) \) is used in the gradient descent process. As long as a local minimum for \( J(\theta) \) exists and we use an appropriate learning rate, this procedure will lead us to local minimum, i.e. the model will converge to a set of parameters such that \( J(\theta) \) is close to a local minimum.

For the two alternatives to batch gradient descent defined below, the full value of \( J(\theta) \) is not calculated before each iteration of gradient descent. Instead, gradient descent occurs each time a limited batch of data of size \( n_{\text{batch}} \) is evaluated by the NN. Thus, with each iteration of gradient descent, the function used in backpropagation is not \( J(\theta) \), but a function using a reduced number of data points, which we will call \( J_{\text{batch}}(\theta) \).

A batch size of 1 leads to what is called stochastic gradient descent (SGD), where parameters are updated each time an individual data point is evaluated by the model. Recall that \( J(\theta) \) is calculated for the entire dataset by summing loss, or error, values \( j_i(\theta) \) for each individual observation. By contrast, stochastic gradient descent calculates the loss for a single observation, then uses the resulting gradient \( \nabla j_i \) to update the network: i.e. \( J_{\text{batch}}(\theta) = j_i(\theta) \). This procedure, while sometimes effective in reducing the time required to train a network to reach impressive results, is less reliable than batch gradient descent. This is because the weight shifts dictated by SGD change depending on the order the observations are given to the network: since the data point used to calculate \( \nabla j_i \) is different from one descent iteration to the next, we cannot be entirely confident that a NN using SGD will converge at all.

Finally, a batch size strictly in between 1 and the entire training set gives us a hybrid process called mini-batch gradient descent. For this process,

\[
J_{\text{batch}}(\theta) = \frac{1}{n_{\text{batch}}} \sum_{i=1}^{n_{\text{batch}}} j_i(\theta).
\]

Here, the fact that the training process is still stochastic, as in SGD, leads to some risk of nonconvergence. However, also similar to SGD, when mini-batch gradient descent produces a well-fitting model, it does so more quickly than batch gradient descent. Since we are using more data points per descent iteration than in SGD, there is less risk of a nonconverging model. This is the approach employed by Carlsson and Gabrielson in their training of CNNs, so their findings suggest that through TDA, we can still observe the learning process of a CNN, despite the stochastic nature of mini-batch gradient descent.

2.3.2 Regularization and Dropout

As just mentioned, a potential issue in training neural networks is overfitting, a problem where a model fits well to a training data set, but deteriorates in performance when operating on unseen testing data. A number of regularization strategies have been employed as measures against overfitting: these procedures aim to avoid extreme (far from 0) parameter values, and thus decrease the potential for over-reliance on single nodes of the network.

\( l_1 \) and \( l_2 \) Regularization: A traditional regularization technique applied to various predictive models is to increase the loss function as each parameter’s absolute value increases. This is most commonly seen in \( l_1 \) and \( l_2 \) norm regularization. When applied to a standard OLS linear regression, \( l_1 \) norm regularization produces a LASSO regression and \( l_2 \) norm regularization produces a ridge regression, as described in [18].

Remark 2.3.1. In this subsection we use \( w_{k,j}^{(i)} \) to denote the \( i^{th} \) layer’s \( k^{th} \) bias parameter \( b_{k}^{(i)} \).

In a neural network, such norm-based regressions can be applied layer-by-layer, as in Keras and other NN implementations. Using an initial loss function \( L(\theta) \), we can define the new loss function using \( l_1 \) regularization as
\[ J_{L1}(\theta) = J(\theta) + \sum_{i=1}^{m-1} \left( \lambda^{(i)} \sum_{j=1}^{n^{(i)}} \sum_{k=1}^{n^{(i+1)}} |w^{(i)}_{k,j}| \right) \]  

where \( m \) is the number of layers in the network, \( \lambda^{(i)} \) is a regularization factor set for the \( i^{th} \) layer, and \( n^{(i)} \) is the number of nodes in the \( i^{th} \) layer. Likewise, the \( l_2 \) regularization loss function can be defined as

\[ J_{L2}(\theta) = J(\theta) + \sum_{i=1}^{m-1} \left( \lambda^{(i)} \sum_{j=1}^{n^{(i)}} \sum_{k=1}^{n^{(i+1)}} (w^{(i)}_{k,j})^2 \right). \]  

**Dropout**: Another commonly used technique with a similar effect is dropout. This procedure also speeds up the training process of large networks. When applying dropout to a neural network, we set a "keep" probability \( p^{(i)} \) to each layer. For each observation in the training set, this is the probability that each node in the \( i^{th} \) layer will be used in forward propagation. Thus, for each observation, there is a probability of \( 1 - p^{(i)} \) for each node in the \( i^{th} \) layer of being "dropped out" of the network. This has the effect of thinning out the network for each observation, reducing the problem of training a single large network multiple times to a problem of training multiple smaller networks. We can intensify this thinning-out effect in the \( i^{th} \) layer by decreasing the "keep" probability \( p^{(i)} \).

**2.3.3 Batch Normalization**

Another related technique that is often used after each layer calculation is batch normalization. In demonstrating some of the ideas set forth in the next several sections, we will use a dataset of chest X-Rays from pneumonia-afflicted patients and healthy patients and train a neural network to classify images into pneumonia and non-pneumonia classes. Each layer will make use of an iteration of batch normalization (or batch norm) before the linear calculations for the next layer proceed.

Batch normalization is a commonly used process in NN architectures generally applied on a layer-by-layer basis, and has been shown to improve NN performance. It was proposed in [17] as a solution to a phenomenon known as internal covariate shift or ICS, which refers to alterations in the distribution of node values as a result of the gradient descent process’s updates to parameters preceding said nodes in a network. However, [28] argues that ICS and training performance have no established positive relationship, and further, that the connection between batch norm and ICS is overall not a strong one. Most relevant to our investigation, similar to tools such as SGD, L1/L2 regularization, and dropout, there seems to be no consensus on an exact mechanism by which batch norm improves NN performance, although [28] cites a "smoothing effect" through "reparametrizing the underlying optimization problem."

Below, Figure 2.6 shows a general outline of the batch norm process taking place after a NN layer. We can see that there is a normalization done: as in a statistical normalization, mean-centering and scaling a list of values so that the new mean is 0, variance 1. Following this is a second scaling process (by a value \( \gamma \)) and a shifting process applied to the result (by a value \( \beta \)).

![Figure 2.6: diagram summarizing the addition of a batch norm operation to a NN layer, from [28].](image)

The first shift-scale operation (normalization), taking a vector (layer of nodes) \( y = (y_1 \ldots y_n) \) to another vector \( \hat{y} = (\hat{y}_1 \ldots \hat{y}_n) \), is done by applying the operation \( \hat{y}_i = \frac{y_i - \mu}{\sigma} \), where \( \mu \) is the mean of all values of \( y \), taken across the entire batch of data currently being considered, and \( \sigma \) is the corresponding standard deviation. Thus, across each batch, after this first step of batch norm, each node in the layer
will have a mean of 0 and a variance of 1. When implementing a NN in Keras with batch norm, the mean and variance with respect to each node is treated as a non-trained parameter.

The second shift-scale operation sets the "true" mean and standard deviation for each node across the batch, with $\beta$ representing the mean post-batch-norm, and $\gamma$ representing the standard deviation post-batch-norm. This operation applies the operation $z_i = \gamma \hat{y}_i + \beta_i$ in order to transform the vector $\hat{y} = (\hat{y}_1 \ldots \hat{y}_n)$ into $z = (z_1 \ldots z_n)$. Unlike $\mu_i$ and $\sigma_i$, these are trained parameters just like the standard weights and biases in the linear operations of a NN. They are initialized randomly at the start of training and are then updated through gradient descent.

Now that all operations used in our observed NNs have been covered, we now turn to the special type of neural network researched by Carlsson and Gabrielsson: the CNN.

### 2.4 Special Model: The CNN

The convolutional neural network is a special type of model that follows the same general operation structure, from input to output, as any other neural network. It contains multiple layers, and there are parameters that transform a vector (often called a tensor in CNN work) from layer to layer. However, as we will see, instead of connecting every node between consecutive layers using trainable parameters, CNNs set many of these weights in each linear transformation to 0 and set many of the others to be equal to one another. This allows us to concentrate and speed up the training process by requiring fewer independent trainable parameters. In addition to speeding up the training process, the specific structure and arrangement of the parameters in a CNN make it intuitive for use with image data, as it aims to capture the appearance of specific patterns in different locations of an input image.

As described by Gunnar Carlsson, convolutional neural networks pare down the collection of unique weights needed for training by imposing a sense of geometry and locality on the nodes of each layer. In a traditional image-processing, the nodes of a convolutional layer (the type of layer that characterizes a CNN) are arranged into a $H \times W \times C$ block. In particular, when $C = 1$, we have a $H \times W$ block of nodes, resembling a singular grid of pixels. This hints at a popular use of CNNs: image processing. Image processing will be the singular function of a CNN that we focus on here, but there are other uses, including analysis of and predictions involving video data. As we will discuss later, the particular strategy used to pare down the trainable parameters in a CNN are specifically intended to work in an analogous manner to mammalian visual processing.

Figure 2.7 displays a diagram indicating the fundamental process of a 10-category image classifier CNN from start to finish. Its input layer contains the original image to be classified, which is taken as the input for the first layer. Note that this example is a grayscale image with height and width both equal to 28 pixels, so its dimension is $28 \times 28 \times 1$, where the single channel represents the fact that we have only one value characterizing each pixel. In a red-green-blue color scheme, this third dimension might have size 3. The first (and usually multiple subsequent) layers of computations in a CNN are specifically characteristic of the CNN structure and are called convolutional layers. The convolution operations carried out by these layers are generally followed up by a summary operation called pooling. These steps will be expanded on following the diagram.

In what we call a standard convolutional layer, the trainable parameters have a "shape" of their own. They come in the form of $f \times m \times n \times C$ blocks of weights known as filters or kernels, where $C$ matches the channel (third) dimension of the previous layer. When building the CNN, we choose $f$, $m$, and $n$ as controllable properties of the network. Often $m = n$, with a popular choice for both $m$ and $n$ being 3. For convolutional layers, the values in these kernels are initialized randomly, and as we feed training data to the network, the weights in these kernels are updated through gradient descent similar to any other neural network. As used in a CNN with the above grid construction of both layers and kernels, a convolution refers to the following:
Definition 2.4.1 (Standard Convolution). Let $x_{i,j,c}$ be the node in the $i^{th}$ row, $j^{th}$ column, $c^{th}$ channel of a convolutional layer. Then the output node $y_{i',j',c'}$ of the standard $m \times n$ standard convolution operation is given by

$$y_{i',j',c'} = \sum_{i'=i'}^{i'+m-1} \sum_{j'=j'}^{j'+n-1} \sum_{c'=c'}^{C} w_{i'-i'+1,j'-j'+1,c,c'} \cdot x_{i,j,c}$$

where $w_{a,b,c,c'}$ is the weight in the $a^{th}$ row, $b^{th}$ column of its $m \times n$ kernel, taking inputs in channel $c$ to outputs in channel $c'$. 

Intuitively, we can describe the action of one of these kernels as ”sliding” across the nodes of a convolutional layer, and computing a dot product (or convolution) of the weights in the kernel with the values in each node being “covered” by the kernel. We can specify the stride, or number of pixels traveled, both horizontally and vertically along each layer, by the kernel before another convolution is computed, when constructing the architecture for the CNN.

Figure 2.8 shows an example of a $3 \times 3 \times 1$ kernel acting on a $4 \times 4 \times 1$ collection of nodes, perhaps pixel values in a grayscale image. Note that when a kernel with more than one parameter acts directly on a set of input nodes in this manner, the corresponding set of output nodes will be smaller by comparison to the input set. In the case below, we have 16 input nodes, but only 4 $3 \times 3$ patches to which we can apply the kernel, resulting in 4 output nodes arranged as a $2 \times 2$ grid. Using the practice of zero padding, or adding zeroes to the border regions of a set of input nodes, we can reduce or even avoid this loss of nodes in subsequent convolutional layers.

For a quick example, the first node in the $2 \times 2$ output is $.5$ as a result of applying the standard convolution operation:

$$0(.7) + 0(.7) + 0(.7) + 0(.1) + 1(.1) + (-1)(.1) + 0(-.5) + (-1)(-.5) + 0(-.5) = .5$$

Once our kernels are finished operating on the input nodes for the convolutional layer, the result will be an $x \times y \times f$ collection of new nodes, where $xy$ is the number of convolutions performed and $f$ is the
number of kernels defined in that layer of the architecture. We can think of this as a new image consisting of \( xy \) pixels and having \( f \) different channels (imagine that these pixels are defined using \( f \) different colors, where \( f = 3 \) is analogous to a red-green-blue color scheme). Although this color comparison is apt for intuitively understanding the structure of an intermediate convolutional layer of a CNN, note that it is common to construct convolutional layers that contain 16, 32, or more channels.

Once the convolution operation is complete, it is generally followed up by a pooling layer, which summarizes the information present in the recently computed set of nodes. A pooling step accomplishes this by breaking up the set of convolution output nodes into subsets, or pools, based on locality and computing a summary value for each pool. Pools can be of whatever dimensions the user wishes (contained within the layer of course) and we can introduce a stride hyperparameter similar to that of the convolutional layers themselves.

**Definition 2.4.2 (Pooling).** An \( m \times n \) pooling operation is a function \( p : \mathbb{R}^{mn} \rightarrow \mathbb{R} \), where the input is arranged as an \( m \times n \) grid, or pool.

Two common pooling strategies are max pooling and average pooling, which work as one might expect. Max pooling, as shown in Figure 2.9, locates the maximum of each specified pool, while average pooling computes the arithmetic average of each specified pool. The chosen summary value is then sent to the next layer as an individual node; in this way, pooling significantly pares down the number of nodes to be carried forward through the CNN. Generally, pools will be non-overlapping, and a choice often used in constructing a pooling layer is that of \( 2 \times 2 \) pools with a stride of 2. This is especially useful for layers of convolution output nodes with even height and weight dimensions, as it allows a summary value to be computed for each and every non-overlapping \( 2 \times 2 \) patch of nodes in the layer.

![Figure 2.9: Figure 2.8 with a max pooling operation added](image)

Applying convolutional kernels to layers with more than one channel works very similar to the one-channel case, in that we intuitively “slide” a kernel across the input nodes and compute a dot product with all values in the node. However, in the multi-channel case, the kernel will also have multiple channels: in particular, the depth dimension of the kernel will match that of the set of input nodes. From here, to carry out a single convolution, we use all the values in the 3-dimensional kernel, and we compute our dot product with respect to all nodes in a common \( m \times n \) region of every channel in the set of input
nodes. A single one of these convolutions across every channel of the input node set will result in the calculation of a single output node for the layer.

Figure 2.10 shows a demonstration of applying $32 \times 3 \times 3 \times 16$ kernels to an $x \times x \times 16$ set of input nodes. As noted above, this third depth dimension of the kernels and the input node set must always match in a traditional CNN. Note the arrows drawn from the kernels to the output node set in the diagram. We can think of the first kernel as transforming the input node set into the first channel of the output node set, then iterate through our remaining kernels until we reach the $32^{nd}$ and final kernel, which transforms the input nodes into the final channel of the output set. This is the process applied in most intermediate convolutional layers of most CNNs, and in the first convolutional layer of RGB color image processing CNNs, as RGB images have three color channels.

Figure 2.10: visualization of how the convolution operation extends to multi-channel layers.

After the convolution and pooling layers detailed in the architecture are finished, the nodes are then flattened and sent to a more traditional neural network layer. In this sense, once we reach the end of the final convolution-pooling iteration, the geometry originally imposed on our nodes is no longer useful for the remaining steps of the network. These remaining steps follow the traditional fully connected procedure from Section 2.1, complete with the usual linear transformations with trained parameters connecting pairs of nodes, until the output layer is reached. For CNNs trained on the MNIST and CIFAR-10 datasets, this output layer will be a vector of size 10, generally with softmax activation, similar to the one defined in Section 2.1.
Chapter 3

Separable Convolutions

Over the past several years, one significant group of CNNs to emerge is the MobileNets, whose characteristic separable convolution operation is detailed in [15]. Separable convolution, also known as depthwise separable convolution, is a special case of standard convolution, defined using a significantly smaller (in most cases) set of trained parameters than a standard convolution as described in Definition 2.4.1. \( n \times n \) separates the \( n \times n \) convolution process into two major steps, as visualized below in Figure 3.1.

![Figure 3.1: visualization of the two steps of a 32-filter 3 x 3 separable convolution operation with valid padding applied to an example X x X x 16 layer to produce a X - 2 x X - 2 x 32 layer.](image)

For a \( C \)-channel layer of input nodes to be transformed to a \( C' \)-channel layer of output nodes, the first step applies \( C \times n \times n \times 1 \) convolutional filters, each filter corresponding to one of the channels. The result of this first step can be thought of as an intermediate \( C \)-channel layer of nodes to be used in the second step. The second step then applies standard convolution to this intermediate layer, using \( C' \times 1 \times 1 \times C \) convolutional filters. Thus, the separable convolution process decouples two major tasks we ask of standard convolution kernels, cross-channel and per-channel convolutions, assigning each task to a much smaller collection of trained parameters. Further, as we’ll see, substituting separable convolutions in place of standard convolutions drastically reduces total number of trained parameters in a convolutional layer.

This alternate convolution process will our focus for applying the Topological Data Analysis techniques described later in this dissertation. As we will detail, prior work by Carlsson and Gabrielsson applied the techniques of persistent homology and Mapper complexes to the kernels of standard convo-
utional layers to reveal significant patterns, while we have sought out similar patterns in the depthwise (Step 1) convolutional kernels of these special separable layers.

The separable convolution process is detailed more rigorously, with summation notation, below:

**Definition 3.0.1 (Separable Convolution).** Let $x_{i,j,c}$ be the node in the $i^{th}$ row, $j^{th}$ column, $c^{th}$ channel of a convolutional layer. Then, to perform separable convolution with an input of $C$ channels and an output of $C'$ channels, we apply two steps of computation.

**Step 1: Depthwise Convolution**
This step produces an intermediate layer of nodes to be passed to Step 2. The node $x'_{i',j',c}$ in this intermediate layer is given by

$$x'_{i',j',c} = \sum_{i' = i}^{i'} \sum_{j' = j}^{j'} w_{i-i'+1,j-j'+1,c} \cdot x_{i,j,c}$$

where $w_{a,b,c}$ is the weight in the $a^{th}$ row, $b^{th}$ column, and $c^{th}$ channel of an $n \times n \times 1$ Step-1 convolutional filter.

**Step 2: $1 \times 1 \times C$ Cross-Channel Convolution**
This step applies a standard $1 \times 1$ convolution to the intermediate layer as determined by Step 1. Thus the output node $y_{i',j',c'}$ of the separable $n \times n$ convolution operation is given by

$$y_{i',j',c'} = \sum_{c' = 1}^{C} w_{c,c'} \cdot x'_{i',j',c}$$

where $w_{c,c'}$ is the weight taking inputs in channel $c$ to outputs in channel $c'$.

Let’s now present a claim made earlier as a formal proposition.

**Proposition 3.0.1.** As described in Definition 3.0.1, a separable convolution layer is a special case of a standard convolution layer as applied to the layer’s input nodes.

**Proof.** Combine the step-1 and step-2 sums from Definition 3.0.1 by expanding the term $x'_{i',j',c}$:

$$y_{i',j',c'} = \sum_{c' = 1}^{C} w_{c,c'} \cdot x'_{i',j',c} = \sum_{c' = 1}^{C} (w_{c,c'} \cdot \sum_{i = i'}^{i'+n-1} \sum_{j = j'}^{j'+n-1} w_{i-i'+1,j-j'+1,c} \cdot x_{i,j,c})$$

$$= \sum_{c = 1}^{C} \sum_{i = i'}^{i'+n-1} \sum_{j = j'}^{j'+n-1} w_{c,c'} \cdot w_{i-i'+1,j-j'+1,c} \cdot x_{i,j,c}$$

This is equivalent to a standard convolution where an $n \times n \times C$ kernels applied to the input nodes $x_{i,j,c}$, where each weight $w_{i-i'+1,j-j'+1,c}$ is replaced with the product of separable weights $w_{c,c'} \cdot w_{i-i'+1,j-j'+1,c}$.

An example $3 \times 3$ full separable convolution layer with a $4 \times 4 \times 3$ section input and $4 \times 1 \times 1$ kernels is detailed below.
Example 3.0.2. Start with a sample of input nodes. We’ll picture this sample as a small $4 \times 4$ section of a much larger 3-channel input layer. From this point we will describe computational specifics in terms of the input nodes displayed in Figure 3.2. We will color-code the channels by a standard approach, representing the depth dimension of our nodes as a red, green, and blue color channel, and each node as a pixel intensity value from 0 to 1.

![Figure 3.2: Example $4 \times 4$ set of input nodes with 3 channels.](image)

In step 1 of separable convolution, the depthwise convolution, we apply a single $3 \times 3$ convolution to each channel from the input, the example kernel we’ll use is in Figure 3.3. Recall that cross-channel convolution is saved for step 2, so we color-code the kernel as well, representing the fact that each channel of the kernel is only applicable to its corresponding color channel in the $4 \times 4 \times 3$ input.

![Figure 3.3: $3 \times 3 \times 3$ set of parameters to be applied per-channel for Step 1, the spatial correlation step.](image)

Since this first convolution operation is carried out only on a per-channel basis, the output will have a depth dimension matching the size of the input nodes and the step-1 kernel. Thus, in this example, we can still conceive of the output of per-channel convolution in terms of RGB color channels, as pictured in Figure 3.4. Note that this computation is done with valid padding, i.e. no zero padding.

The 3-channel step-1 output in Figure 3.4 then undergoes the step-2 cross-channel convolution step. Recall that the number of convolutions specialized in the CNN architecture specifically affects this step by determining the number of $1 \times 1 \times C$ (in this case depth 3) kernels to apply to this intermediate collection of values. This example’s cross-channel kernels are pictured in Figures 3.5 and 3.6. Figure 3.5 is color-coded to correspond to the RGB channels in Figure 3.4, and Figure 3.6 is color-coded to correspond to the final output channels in Figure 3.7.
Recall that this last cross-channel convolution is simply a $1 \times 1$ standard convolution, and so the 4 filters pictured in Figures 3.4 and 3.6 each correspond to an output channel to be used as direct inputs to future pooling, normalization, or convolution operations. The outputs for this example are pictured in Figure 3.7. Note that the height and width dimensions of this final output are the same as in Figure 3.4, since $1 \times 1$ convolution does not reduce output size, even under valid padding.
3.1 Advantage: Cutting Down the Number of Trained Parameters

As mentioned previously, there is a potentially dramatic decrease in the number of trained parameters necessary when replacing standard convolution layers with separable convolution layers. [15] poses some practical motivations for such an innovation, noting that when used in fields such as robotics and autonomous vehicle design, CNNs must be built to operate quickly and on systems with limited memory.

This difference between the two convolution styles can be seen plainly by breaking down the dimensions of a convolution kernel in each case.

Example 3.1.1. Figure 3.8 re-displays the image from Figure 2.10 for use with this example. Here we see an example of standard $3 \times 3$ convolutional layer with 32 kernels operating on a $x \times x \times 16$ input and valid padding, resulting in a $(x - 2) \times (x - 2) \times 32$ output.

In this particular example, the 32 kernels contain, in total, $3 \cdot 3 \cdot 16 \cdot 32 = 4608$ trained parameters. Let’s now contrast this with $3 \times 3$ separable convolution on the same $x \times x \times 16$ collection of input nodes and the same output depth of 32, as re-displayed from Figure 3.1 in 3.9.

Since Step 1’s depthwise convolution involves only a single $3 \times 3 \times 16$ kernel, where each channel of the kernel only performs a convolution with its corresponding channel in the input, this first step contributes only $3 \cdot 3 \cdot 16 = 144$ trained parameters to the process.

Meanwhile, Step 2’s $32 1 \times 1$ convolutions contribute an additional $16 \cdot 32 = 512$ trained parameters to this separable convolution layer. Before any additional parameters are added due to bias, batch normalization, or any other reason, the total trained parameter count in the layer is

$$3 \cdot 3 \cdot 16 + 16 \cdot 32 = 16 \cdot (3 \cdot 3 + 32) = 656,$$

resulting in a ratio of separable convolution parameters to standard convolution parameters of about 0.1613.
Generalizing to a convolution operation and input/output channels of arbitrary size, let $c$ be the number of input channels in a convolutional layer, $f$ the number of output channels, and let $m$ and $n$ be the dimensions of the convolution operation. (Recall that in many practical applications we default to a square kernel, i.e. $n \times n$ convolution.)

In a standard convolution such as in Figure 3.8, the number of trained parameters in the layer will be the product

$$c \cdot m \cdot n \cdot f.$$

On the other hand, in a separable convolution as in Figure 3.9, the number of trained parameters for the depthwise step is

$$c \cdot m \cdot n,$$

and for the second $1 \times 1$ convolution step we require

$$c \cdot f$$

trained parameters, resulting in a total of

$$c \cdot m \cdot n + c \cdot f = c \cdot (m \cdot n + f)$$

trained parameters for the full separable convolutional layer.

Comparing these two parameter totals by a ratio of separable convolution parameters to standard convolution parameters yields

$$\frac{c \cdot (m \cdot n + f)}{c \cdot m \cdot n \cdot f} = \frac{(m \cdot n + f)}{m \cdot n \cdot f} = \frac{1}{f} + \frac{1}{mn},$$

and so, further, with an arbitrarily large combination of convolution size and number of output channels comes an arbitrarily small ratio of required trained parameters in a separable convolution layer to that of a standard convolution layer.
One might expect, because of the decrease in the number of trained parameters from a standard CNN layer to a separable layer, that the training process should be sped up in real time when replacing standard convolutional layers with those employing a separable convolution approach. However, despite the potentially extreme decrease in trained parameters, [15] has a notable omission when comparing MobileNets with separable convolution layers to other types of CNNs, particularly VGG 16, which employs only standard convolutions in its first several layers.

This paper describes the massive distinctions between these network types in terms of
1. the total number of parameters, and
2. the number of multiplication and addition operations (or "mult-adds") necessary to generate a classification based on input data in a trained network,

but lacks any discussion of a difference in runtime of the training operation. [32] notes that there is no guarantee on an improvement in runtime of the training process when switching from standard to separable convolutions, as hardware inefficiencies may be present. In particular, as we have observed in our work (as well as many forums on GitHub and StackOverflow), there is a well-documented pattern of TensorFlow, an often-used neural-network training package, creating separable convolution layers that train much slower than standard convolution layers of the same dimension.

While this is an unfortunate issue, it was certainly not prohibitive to our work, as we used a relatively small and simple CNN architecture (5 convolution layers). From many initializations of networks with this architecture, we were able to obtain a large set of trained depthwise kernels, giving us a large set of data for applying the aforementioned TDA techniques.

3.2 Inception and Xception

[Inception paper: going deeper with convolutions]

When it comes to computations inside a CNN’s convolutional layers, a close cousin of separable convolution is known as Inception, a general procedure with quite flexible hyperparameters, by comparison to standard or separable convolution. In fact, as we will see, [8] describes Inception as a generalization of standard and separable convolution processes, in particular an "intermediate step." As we will note, the separable convolutions described in [8] have some notable distinctions from the type introduced in Definition 3.0.1, but the fundamental concept of decoupling inter- and intra-channel operations is conserved.

As presented in [30] and [8], the Inception framework for convolution layers in CNNs is a generalization of a standard convolution that allows for multiple different convolution-related operations to be carried out, followed by a concatenation operation which consolidates the final form of the output.

Definition 3.2.1. An Inception module is a CNN layer that performs \( k \) different standard convolution operations on a \( H \times W \times D \) collection of input nodes. For \( i \in \{1, 2, \ldots, k\} \), the \( i^{th} \) convolution can be of any chosen size \( m_i \times n_i \) (with \( 0 \leq m_i \leq H \) and \( 0 \leq n_i \leq W \)) and produce any number \( f_i \) of output channels. However, each output must be of the same height \( H' \) and width \( W' \). The outputs of each of the \( k \) convolutions are then concatenated channel-wise, producing an output with \( \sum_{i=1}^{k} f_i \) channels.

An example of such an Inception module is shown in Figure 3.10. Note here that the average pooling operation is precisely a standard convolution with an untrained kernel where all parameter entries are \( \frac{1}{mn} \) for an \( m \times n \) average pool. Additionally, pooling operations inside an Inception module tend to use a stride parameter of 1 rather than the traditional \( n \times n \) pool with stride \( n \).
Figure 3.10: Example Inception module with distinct combinations of $1 \times 1$ and $3 \times 3$ standard convolutions, from [8].

[8] proposes a special case of the Inception module, called Xception, that is extremely similar to the separable convolution operation, with a few differences that vary in significance. As we will see, one visual difference is the order of operations, as the $1 \times 1$ standard convolution is performed in advance of the $m \times n$ depthwise convolution. This paper gives an illustrative example, shown in Figure 3.11 starting by drawing up an Inception module where each of the convolutions is a distinct composition of $1 \times 1$ convolution with $3 \times 3$ convolution.

Figure 3.11: Simplified Inception example with the same combination of convolutions repeated for each of its three operations, from [8].
Equivalently, we can reformulate the multiple instances of $1 \times 1$ standard convolution as a single larger $1 \times 1$ standard convolution. This can be done by considering $k$ $1 \times 1$ convolutions done in parallel, each with $f_i$ output channels, and replacing the operation with a single $1 \times 1$ convolution with $\sum_{i=1}^{k} f_i$ output channels. This reformulation is visualized in Figure 3.12.

![Figure 3.12: Restructured Inception module, functionally equivalent to Figure 3.11, from [8].](image)

From this point, the Xception module is built by, once the $1 \times 1$ convolutions have been merged into the aforementioned single large convolution operation, applying $f \ m \times n$ convolution kernels, but restricting each kernel to its own distinct channel of nodes. The example from [8] is shown in Figure 3.13. This Xception operation contains the same cross-channel vs channel-wise decoupling seen in separable convolution. However, the operation is done in reverse order, as the $f \ 1 \times 1$ kernels are applied to the input layer, producing the required number of $H \times W$ channels to continue to the channel-wise convolution step.

![Figure 3.13: Logical extension of Figure 3.12, applying a $3 \times 3$ convolution to each output channel of the initial $1 \times 1$ convolution, from [8].](image)
It should be noted that [8] also notes that a non-linear activation should be applied in between the two major convolution steps. In other words, the depthwise convolution is applied to a non-linearly transformed output of the $1 \times 1$ convolution step. This intermediate nonlinearity is a crucial distinguishing factor between separable convolutions and Xception modules.

Alternate structures such as Inception and Xception provide any intriguing possibility for future work. As will be presented in later chapters, interesting topological patterns (which we will later detail through the lens of nontrivial homology) were observed first during the training of standard CNNs, and we also observed such patterns in our study of separable CNN layers. However, we have not yet investigated whether similar findings can be seen in these alternate CNN constructions.
Chapter 4

Motivating TDA from Algebraic Topology Foundations

For a given set of data points, one way to describe and make inferences about the set is to apply topological techniques. The goal of these types of techniques is to ascribe particular geometric properties to our dataset, and these techniques fall under the practice of topological data analysis (or TDA). Two such techniques that have been used in researching the "shape" of weights in trained CNNs are persistent homology and the Mapper algorithm.

Before elaborating on the specific TDA techniques used, we will start with some preliminary information about topological spaces, then move to the tools of algebraic topology that will be applied, in order to motivate the use of these techniques. Our main reference texts for the fundamental concepts of this chapter are [14] and [24]. We also use several examples from (and highly recommend!) two online courses offered by Gabriel Islambouli¹ and Marius Furter², which cover selected material from [14] and [24], respectively.

4.1 General Motivations of Topology and Algebraic Topology

In the field of topology, the main objects under consideration are topological spaces, and the focus is on defining and distinguishing objects based on continuous transformations, also known as continuous maps or deformations. In an intuitive sense, continuous maps take "nearby" points to other "nearby" points, and we will make this notion more rigorous later on. Specifically, there is a special notion of continuous equivalence called homeomorphism that, from a topological perspective, renders two spaces functionally identical. These terms will be defined more rigorously as we walk through the foundational concepts in defining topological spaces.

A commonly-referenced example of such an equivalence is demonstrated by the quip "A topologist cannot distinguish between a doughnut and a coffee cup." Figure 4.1 shows a model from Henry Segerman’s "Topology Joke" video³ showing a visualization of a such a transformation.

Qualities of such a space that remain constant under homeomorphism are known as topological invariants. This prompts the question: if finding a special type of transformation between two objects renders them equivalent, then what tools can we use to conclude that two spaces are not equivalent? Specifically in the doughnut/coffee cup example, a prominent central similarity between these two objects is the existence of exactly one "hole," and the other differences in shape between the two objects are considered irrelevant when attempting to distinguish the two as topological spaces.

¹https://www.youtube.com/playlist?list=PLLevbjS8851N39Nk8ve13FE5tMDZejStE
²https://www.youtube.com/playlist?list=PLd8NhPjkXPliJunShxtDNMuFsmZPeHpm-0
³https://www.youtube.com/watch?v=9MlqyTr6-TpA
Algebraic topology is a subfield concerned with using a group theory perspective to find topological invariants. Clearly, if when analyzing two topological spaces $X$ and $Y$, we find a topological invariant that differs between $X$ and $Y$, then $X$ and $Y$ must be distinct topologically. Two major types of such invariants deal with homotopy equivalence and homology, which we will be expand upon later.

Consider the comparison between a sphere in $\mathbb{R}^3$ and a torus $\mathbb{R}^3$, i.e. the donut shape seen at the bottom of Figure 4.1. An algebraic topology tool that we can use to distinguish between these two surfaces is the fundamental group, which applies a group structure to the set of all loops in a space. As we will see when covering the fundamental group with more rigor, this fundamental group is different between the sphere and the torus. This difference, in short, is that all loops on the surface of the sphere can be contracted continuously to a point.

However, there is a limit to the fundamental group’s power to distinguish topological spaces. For example, a sphere is not topologically equivalent to a single point, but the fundamental group does not distinguish the two. On the other hand, homology groups can be used to distinguish these two objects.

4.2 Continuity, Topological Spaces, Open and Closed Sets

Let’s start our discussion of continuous maps using the familiar $\epsilon - \delta$ definition from analysis in $\mathbb{R}^n$:

\textbf{Definition 4.2.1.} A map $f : \mathbb{R}^n \to \mathbb{R}^k$ is continuous if for every $\epsilon > 0$ and $x_0 \in \mathbb{R}^n$, there exists a $\delta > 0$ such that $x \in B_\delta(x_0)$ implies $f(x) \in B_\epsilon(f(x_0))$, where $B_\delta(x_0)$ is the open ball about $x_0$ with radius $\delta$.

We can restate this definition equivalently using a perspective more specifically relevant to general topological spaces.

\textbf{Definition 4.2.2.} A map $f : \mathbb{R}^n \to \mathbb{R}^k$ is continuous if the pre-image $f^{-1}$ of any union of open balls is also a union of open balls.
Let’s look at a very quick example and non-example using maps in \( \mathbb{R} \). Note that open balls in \( \mathbb{R} \) are simply open intervals \((a, b)\).

**Example 4.2.3.** Consider the function \( f : \mathbb{R} \to \mathbb{R} \) given by \( f(x) = 3x \). Since \( f^{-1}(a, b) = (a/3, b/3) \), this is a continuous function.

**Example 4.2.4.** Consider the function \( f : \mathbb{R} \to \mathbb{R} \) given by

\[
f(x) = \begin{cases} 
  x, & \text{if } x < 0 \\
  x + 1, & \text{if } x \geq 0 
\end{cases}
\]

Note that \( f^{-1}(.5, 1.5) = [0, .5) \). Thus, the pre-image of the open interval \((.5, 1.5)\) is not a union of open intervals, showing that \( f \) is not continuous.

Now we will generalize the notion of open sets to general topological spaces, including those beyond \( \mathbb{R}^n \), starting with a definition of such a space.

**Definition 4.2.5.** A topological space is a set \( X \) equipped with a special collection of subsets \( \tau \). \( \tau \) must satisfy the following properties:

1. \( \emptyset \in \tau \) and \( X \in \tau \).
2. Any arbitrary union of elements from \( \tau \) is also in \( \tau \).
3. Any finite intersection of elements from \( \tau \) is also in \( \tau \).

The set \( \tau \) is called a topology on \( X \) and elements of \( \tau \) are called open sets.

Verifying that considerations 1 and 2 apply to open balls in \( \mathbb{R}^n \) is rather straightforward, and there is a classic example that demonstrates why we specify that intersections are finite in consideration 3.

**Example 4.2.6.** In \( \mathbb{R} \), the intersection \( \bigcap_{n=1}^\infty (1 - 1/n, 1 + 1/n) = \{1\} \), which is not an open set.

Finally, we introduce the concept of a basis in a topological space. This a useful notion for defining certain topological spaces.

**Definition 4.2.7.** Starting from a topological space \( X \) with topology \( \tau \), a collection of subsets \( B \) is called a basis if

1. \( B \subseteq \tau \).
2. Any open set in \( X \) is precisely a union of elements of \( B \).

This leads us to a few classic examples of topological spaces.

**Example 4.2.8.** Let \( X \) be a set equipped with a metric \( d \), i.e. \( d(x, y) \in \mathbb{R} \) defines the distance between two points \( x, y \in X \). Define the open ball \( B_r(x) \) as \( \{ y \in X : d(x, y) < r \} \). Then \( B = \{ B_r(x) : x \in X, r \in (0, \infty) \} \) is a basis for a topology on \( X \). We may refer to this as the metric topology induced on \( X \) by \( d \). In many contexts, one uses \( X = \mathbb{R}^n \) and \( d(x, y) = ||y - x||_2 \).
Example 4.2.9. For any set $X$, the **trivial topology** is $\tau = \{X, \emptyset\}$. Here, no subsets of $X$ are considered open other than the full space $X$ and the empty set.

Example 4.2.10. For any set $X$, the **discrete topology** is $\tau = \mathcal{P}(X)$, the power set of $X$. Here, all subsets of $X$ are considered open. This is another extreme but relatively uninteresting case, similar to 4.2.9.

A richer, more interesting example of defining a topology starts from a larger topological space $X$ and using this space to derive a topology for a subset $S \subseteq X$. This is known as the **subspace topology**.

**Definition 4.2.11.** Start with a topological space $X$ with topology $\tau_X$ and consider a subset $S \subseteq X$. The **subspace topology** on $S$ is

$$\tau_S = \{U \subseteq S : U = S \cap V \text{ for } V \in \tau_X\}.$$  

Two intuitive examples of such a topology involve topologizing the unit circle as a subspace of $\mathbb{R}^2$ and the unit sphere as a subspace of $\mathbb{R}^3$.

Example 4.2.12. Define the **unit circle** as $S^1 = \{(x,y) \in \mathbb{R}^2 : x^2 + y^2 = 1\}$. Then the subspace topology on $S^1$ as a subspace of $\mathbb{R}^2$ defines open sets in $\tau_{S^1}$ as all unions of "bent open intervals," as depicted in Figure 4.2 by the green regions of the circle on the right.

![Figure 4.2: open sets in $S^1$ as defined by the subspace topology from $\mathbb{R}^2$.](image)

Example 4.2.13. Define the **unit sphere** as $S^2 = \{(x,y,z) \in \mathbb{R}^2 : x^2 + y^2 + z^2 = 1\}$. Then the subspace topology on $S^2$ as a subspace of $\mathbb{R}^3$ defines open sets in $\tau_{S^2}$ as all unions of "open patches," as depicted in Figure 4.3 by the green regions of the sphere on the right.

![Figure 4.3: open sets in $S^2$ as defined by the subspace topology from $\mathbb{R}^3$.](image)
A potentially unintuitive example is given by the subspace topology of the positive numbers as a subspace of \( \mathbb{R} \).

**Example 4.2.14.** Let \( S = [0, \infty) \). When we apply the subspace topology using \( \mathbb{R} \) as our special superset of \( S \), then open sets of \( S \) are defined as intersections between \( S \) and open intervals \((a, b)\) in \( \mathbb{R} \). More specifically, open sets in \( S \) are of the form \([0, b)\) or \((c, d)\). Thus there are sets that are not open in \( \mathbb{R} \) that are open in \( S \).

An important counterpart to open sets in topology is the closed set. Note that closed sets and open sets are not a dichotomy, as certain sets can be both open and closed.

**Definition 4.2.15.** In a topological space \( X \), a closed set is the complement \( X \setminus U \) of an open set \( U \subseteq X \).

It follows from Definitions 4.2.15 and 4.2.5 that...

**Theorem 4.2.1.** The following are properties of closed sets:

1. \( \emptyset \) and \( X \) are closed sets (note they are also open).
2. Any arbitrary intersection of closed sets is a closed set.
3. Any finite union of closed sets is a closed set.

Now that open and closed sets are thoroughly defined, we define continuity of a map between general topological spaces, not just \( \mathbb{R}^n \) to \( \mathbb{R}^k \) with the standard metric topology.

**Definition 4.2.16.** Define topological spaces \( X \) and \( Y \). A map \( f : X \to Y \) is called continuous if, for all open sets \( U \subseteq Y \), \( f^{-1}(U) \) is an open set of \( X \).

Based on this definition, we can create continuous maps in spaces other than \( \mathbb{R}^n \).

**Example 4.2.17.** Let \( X \) be any topological space with the discrete topology. Recall this means all subsets of \( X \) are open. As a result, any map \( f : X \to Y \) to any topological space \( Y \) is continuous.

However, the trivial topology has an opposing result involving continuous maps, as illustrated by the following example.

**Example 4.2.18.** Let \( X \) be any topological space with trivial topology, i.e. only \( X \) and \( \emptyset \) are open. Then any map \( f : Y \to X \) from any topological space \( Y \) to \( X \) is continuous.

Additionally, we can equivalently define continuity itself using closed maps.

**Theorem 4.2.2.** A map \( f : X \to Y \) is continuous if and only if for all closed sets \( C \subseteq Y \), \( f^{-1}(Y) \) is closed in \( X \).

### 4.3 Homeomorphism

Moving on, we now introduce and define homeomorphism, the property that functionally equates two spaces in the field of topology.
**Definition 4.3.1.** A *homeomorphism* is a function $f : X \rightarrow Y$ satisfying the following conditions:

1. $f$ is continuous.
2. $f$ is a bijection.
3. $f^{-1}$ is continuous.

If such a map exists between two spaces $X$ and $Y$, then we call $X$ and $Y$ *homeomorphic*, and, as topological objects, we consider them to be identical.

**Example 4.3.2.** The sigmoid activation function $f : \mathbb{R} \rightarrow (0, 1)$ given by $f(x) = \frac{e^x}{e^x + 1}$, as described earlier in the Neural Network Overview, is a homeomorphism, as

1. $f$ is continuous.
2. $f$ has an inverse $f^{-1}(y) = \ln \frac{y}{1-y}$.
3. $f^{-1}$ is continuous.

This topological equivalence between finite- and infinite-length open intervals tells us that topology does not distinguish spaces purely based on the size of any of the spaces’ features, i.e. size is not a topological property. However, one important consideration involving homeomorphisms is that they cannot "glue" points together from a space $X$ to create a space $Y$, as we'll see in the following example.

**Example 4.3.3.** The function $f : [0, 2\pi) \rightarrow S^1$ given by $f(\theta) = (\cos(\theta), \sin(\theta))$ is not a homeomorphism.

1. $f$ is continuous, and
2. $f^{-1}$ exists, but
3. $f^{-1}$ is not continuous.

We can see that Consideration 3 holds by observing that $[0, \pi/4)$ is an open set of $[0, 2\pi)$ using the subspace topology induced from $\mathbb{R}$, but $(f^{-1})^{-1}([0, \pi/4))$ is the set of points on the arc of $S^1$ from $(0, 0)$ inclusive to $(\sqrt{2}/2, \sqrt{2}/2)$ noninclusive. From Example 4.2.12 we know that this is *not* an open set of $S^1$.

On the other hand, there is a way of subtly altering such topological spaces that allows for homeomorphisms that involve such "gluing" procedures. Specifically, the notion of a *quotient space* will allow us to create a version of $[0, 2\pi)$ that is homeomorphic to $S^1$.

### 4.4 Quotient Spaces

Let’s now capture the intuition of ”gluing points together” which will form the foundation for the definition of a quotient space. Since a new topological space will be formed, we will need to define a topology for it, i.e. a special collection of subsets to be considered open. As demonstrated in the following example, we want open sets in the ”glued” space to be unions of sets that were open in the original ”pre-glued” space.

**Example 4.4.1.** Consider two hollow cubes as visualized below in Figure 4.4 that are glued together along a face to create a connected space. The two dotted half-circles on each cube represent two open sets, which form another open set (the full circle) in the glued space.

We now introduce the general definition of a *quotient space* and a *quotient topology*: note the specific action of gluing two points together is a type of equivalence relation between these two points.
Figure 4.4: an example of an open set in a space consisting of two cubes glued together along a face.

**Definition 4.4.2.** Consider a topological space $X$ with topology $\tau_X$ and an equivalence relation $\sim$ on $X$. The quotient space $Y = X/\sim$ is the set of equivalence classes of $X$ under $\sim$: we denote the equivalence class of $x \in X$ under $\sim$ as $[x]$. We endow $Y$ with the quotient topology from $X$ and $\sim$ as follows.

$$\tau_Y = \{U \subseteq Y : \{x \in X : [x] \in U\} \in \tau_X\}$$

That is, an open set in the quotient space $Y$ is a union of equivalence classes such that the union of all elements from all equivalence classes involved is an open set in $X$.

Gluing points together through the use of such equivalence relations underlies the construction of larger objects we will refer to soon, called *CW complexes*. An important type of CW complex is the simplicial complex, the main object we will be observing in the TDA practices discussed in future sections.

We will now apply this definition to the space from Example 4.3.3 and show that this allows us to create a version of the interval $[0, 2\pi]$ that is homeomorphic to the unit circle $S^1$. The process we set out below is, intuitively, gluing the endpoints of the interval together by including them in the same equivalence class.

**Proposition 4.4.1.** Define $X = [0, 2\pi]$ and $\sim$ as the equivalence relation such that $[0] = \{0, 2\pi\}$ and $[x] = \{x\}$ for any $x \in X \setminus \{0, 2\pi\}$. Then $X/\sim$ is homeomorphic to $S^1$.

A homeomorphism between these two spaces has a similar form to the non-homeomorphism from Example 4.3.3, but we’ll need to define it subtly differently as $f : X/\sim \to S^1$, where

$$f([\theta]) = \begin{cases} lr(0,0), & \text{if } x \in \{0, 2\pi\} \\ (\cos(\theta), \sin(\theta)), & \text{if } x \notin \{0, 2\pi\}. \end{cases}$$

As discussed before in Example 4.3.3, it is clear that $f$ is both continuous and bijective, and so an inverse $f^{-1}$ exists. However, our alteration of the interval $[0, 2\pi]$ to the quotient space $X/\sim$ has also
made $f^{-1}$ a continuous map, as we no longer need to consider $[0,x]$ (for any $x$ in the open interval $(0,2\pi)$) as an open set in $X$.

Alternately, we may define quotient spaces using an object called the quotient map. A special case of a quotient map is $\pi : X \to X/\sim$, which sends $x$ to its equivalence class, i.e. $x \mapsto [x]$ through $\pi$. An essential property of this map $\pi$ is that

$$U \subseteq X/\sim \text{ is open } \iff \pi^{-1}(U) \subseteq X \text{ is open},$$

implying $\pi$ is continuous. More generally,

**Definition 4.4.3.** For two topological spaces $X$ and $Y$, a quotient map is a surjective map $\pi : X \to Y$ such that

$$U \subseteq Y \text{ is open } \iff \pi^{-1}(U) \subseteq X \text{ is open}.$$

An important example of a quotient map is the simple projection from $\mathbb{R}^n + k$ to $\mathbb{R}^n$. Below we use the example of projecting the plane $\mathbb{R}^2$ vertically to the x-axis.

**Example 4.4.4.** Let $\pi : \mathbb{R}^2 \to \mathbb{R}$ be the map given by $(x,y) \mapsto x$. Then for any open interval $U = (a,b)$ in $\mathbb{R}$, $\pi^{-1}(U)$ is the region bounded by (but not including) the vertical lines $x = a$ and $x = b$.

![Figure 4.5: visualization of the pre-image of an open interval in $\mathbb{R}$ through the simple projection map $\pi$ described in Example 4.4.4.](image)

A crucial property of the quotient topology is given below, with a proof.

**Theorem 4.4.2.** Let $\pi : X \to Y$ be a quotient map. Then for any topological space $B$ and map $f : Y \to B$, $f$ is continuous if and only if $f \circ \pi : X \to B$ is continuous.

**Proof.** First, assume $f$ is continuous. Then because compositions of continuous maps are continuous, $f \circ \pi$ is continuous.

Next assume $f \circ \pi$ is continuous, and consider an open set $U \subseteq B$. Since $f \circ \pi$ is continuous, $(f \circ \pi)^{-1}(U)$ is then an open set, i.e. $\pi^{-1}(f^{-1}(U))$ is open. It follows from Definition 4.4.3 that $f^{-1}(U)$ is open, confirming that $f$ is a continuous map.

An important corollary to Theorem 4.4.2 is commonly referred to as passing to the quotient.

**Corollary 4.4.3.** For topological spaces $X$, $Y$, and $B$, let $\pi : X \to Y$ be a quotient map, then let $f : X \to B$ be a continuous function with the condition that if $\pi(x) = \pi(y)$, then $f(x) = f(y)$. Then there exists a unique continuous map $\tilde{f} : Y \to B$ such that $f = \tilde{f} \circ \pi$. 

46
We may say that \( f \) factors to \( \tilde{f} \) through the quotient map \( \pi \).

**Example 4.4.5.** Consider the map \( f : [0, 2\pi] \to [-1, 1] \) given by \( f(x) = \sin(x) \) and the quotient map \( p : [0, 2\pi] \to S^1 \) given by \( p(\theta) = (\cos(\theta), \sin(\theta)) \). Then the unique map \( \tilde{f} \) satisfying \( f = \tilde{f} \circ p \) is \( \tilde{f}(x, y) = y \), the simple projection to the \( y \)-axis.

### 4.5 CW Complexes

Let’s now move to a more complex and general structure that can be formed from such a quotient-topology approach, where we ”glue” points via equivalence relations: the *CW complex*. We’ll start from the building blocks of such complexes, which are \( n \)-dimensional disks, denoted \( D^n \).

**Definition 4.5.1.** The \( n \)-dimensional disk \( D^n \) is the set of points within a unit distance of the origin in \( \mathbb{R}^n \), i.e.

\[
D^n = \{(x_1, x_2, \ldots, x_n) : \sum_{i=1}^{n} x_i^2 \leq 1\}
\]

and the \( n-1 \)-sphere \( S^{n-1} \) is the set of points exactly a unit distance from the origin in \( \mathbb{R}^n \), i.e.

\[
S^{n-1} = \{(x_1, x_2, \ldots, x_{n-1} : \sum_{i=1}^{n-1} x_i^2 = 1)\}.
\]

By convention, \( D^0 \) is a single point.

In this sense, \( S^{n-1} \) is the boundary of \( D^n \); for example, the disk \( D^2 \) has the unit circle \( S^1 \) as its boundary. Noting this connection between \( D^n \) and \( S^{n-1} \) presents the foundation for building CW complexes, structures that are formed by gluing copies of \( D^n \) along their boundaries, which are copies of \( S^{n-1} \).

**Example 4.5.2.** Consider the space visualized below in Figure 4.6. This is an example of a 2-dimensional CW complex.

![Figure 4.6: a simple 2-dimensional CW complex.](image)

In gluing terms, we can think of the first point \( D^0 \) as the first step in constructing this space. From here, we glue two copies of the unit interval \( D^1 \) with their endpoints situated at the location of our \( D^0 \). We then glue a copy of \( D^2 \) along its boundary \( S^1 \) to the right copy of the unit interval to finish constructing the space.

Let’s now generalize this procedure to arrive at a rigorous definition of a CW complex, along with a few related terms.

47
Definition 4.5.3. A CW complex is a topological space built using the following procedure:

1. Start with a discrete set of points. This set will be called $X^0$.

2. We now follow an inductive process to construct the $n$-skeleton $X^n$ from $X^{n-1}$ by attaching copies of $D^n$ to $X^{n-1}$. Let $D_n$ refer to this collection of $D^n$ copies and $S_{n-1} \subseteq D_n$ refer to the union of their boundaries (copies of $S^{n-1}$). Define a map $\phi_n : S_{n-1} \rightarrow X^{n-1}$: this lets us know which pairs of points $D_n$ and $X^{n-1}$ will be attached.

In quotient terms, this "attachment" or "gluing" is done by starting with the disjoint union $X^{n-1} \sqcup D_n$, then defining an equivalence relation $\sim$ between $S_{n-1}$ and $X^{n-1}$, where $x \sim \phi(x)$ for $x \in S_{n-1}$.

3. This process may end at a finite step $n$, and in this case the final CW complex $X$ is the $n$-skeleton $X^n$ with the quotient topology. Recall this means that open sets in $X$ are unions of open sets from all $m$-skeletons, where $m \leq n$. If $X = X^n$ and $X \neq X^{n-1}$, then $X$ is called $n$-dimensional.

It is possible to continue this process indefinitely, but infinite-dimensional CW complexes are beyond the scope of our discussion.

Example 4.5.4. Any graph is an example of a CW complex formed by gluing copies of $D^0$ and $D^1$.

Example 4.5.5. For any $n \in \mathbb{N}$, $S^n$ is a CW complex formed from a point $x$ and a copy of $D^n$, as shown below in Figure 4.7’s examples. The map used to attach $D^n$ to $x$ is the constant map $\phi : S^{n-1} \rightarrow x$. 
Figure 4.7: Visualization of combining $D^0$ with $D^1$ or $D^2$ to form $S^1$ and $S^2$ respectively.

Higher-dimensional spheres can be formed using the same process, including forming the 3-sphere using the 2-sphere in $\mathbb{R}^3$ as the boundary to identify with the single point $x$.

**Definition 4.5.6.** A closed subset of a CW complex $X$ that is itself a CW complex is called a subcomplex of $X$.

**Definition 4.5.7.** Let $X$ be a CW complex with a subcomplex $A$. Then $(X, A)$ is called a CW pair.

**Example 4.5.8.** Referring back to Figure 4.6, we can create a CW pair $(X, A)$ by considering the entire space as $X$ and considering the left loop as $A$.

Figure 4.8: Figure 4.6 with labels to illustrate the CW pair $(X, A)$. 
A more involved example of a CW complex uses lines in $\mathbb{R}^{n+1}$ as its elements rather than points.

**Definition 4.5.9.** $\mathbb{R}P^n$ is the set of all lines through the origin in $\mathbb{R}^{n+1}$, i.e.

$$\mathbb{R}P^n = \{ \{\lambda x : \lambda \in \mathbb{R}\} : x \in \mathbb{R}^n \setminus \{0\}\}.$$

**Example 4.5.10.** $\mathbb{R}P^n$ is a CW complex. We can start to illustrate this by starting from the fact that any line in $\mathbb{R}P^n$ intersects $S^n$ in exactly two points. Figure 4.9 shows this for $\mathbb{R}P^1$ and $S^1$.

![Figure 4.9: illustration of two-point intersections between $S^1$ and elements from $\mathbb{R}P^1$.](image)

We call such pairs of points on $S^n$ **antipodal**, and we call the map relating antipodal points the **antipodal map** $x \mapsto -x$. In particular, $\mathbb{R}P^n$ is homeomorphic to $S^n/\{x \sim -x\}$, i.e. where the quotient is taken with respect to the equivalence relation identifying antipodal points.

Let’s now consider how this relates $\mathbb{R}P^2$ and $S^2$. As shown in Figure 4.10, we see again that a pair of antipodal points can be identified with a line, providing further justification for the use of $\{x \sim -x\}$ equivalence classes.
Note that points in the upper hemisphere of $S^2$ are then identified with points in the lower hemisphere through this equivalence relation, and points on the equator are identified with other points on the equator. This means that we can eliminate the remainder of the sphere (the lower hemisphere) from consideration.

We now note that the equator region of $S^2$ is simply $S^1$, but since antipodal points are identified, this equator region can be reduced to $\mathbb{R}P^1$. In this way, we can construct $\mathbb{R}P^2$ by attaching $D^2$ to $\mathbb{R}P^1$.

Extending to $\mathbb{R}P^n$ inductively, it can be shown that $\mathbb{R}P^n$ is homeomorphic to $\bigcup_{i=0}^n D^i$, where at each step $D^i$ is attached to $X^{i-1} = \mathbb{R}P^{i-1}$ by the quotient map $\pi : S^{i-1} \to \mathbb{R}P^{i-1}$ induced by the antipodal map on $S^{i-1}$.

### 4.6 Product Spaces

We will soon be introducing a weaker form of equivalence between spaces than homeomorphism, known as homotopy equivalence. Essential to the notions of homotopy and homotopy equivalence is the concept of a product space. Such a space $X \times Y$ can be considered to have a copy of $Y$ at each point in $X$, or vice versa.

**Definition 4.6.1.** For two topological spaces $X$ and $Y$, the *product space* is the Cartesian product of $X$ and $Y$ and is given a topology which can be defined by the basis

$$B = \{U \times V : U \in \tau_X, V \in \tau_Y\},$$

i.e. open sets in $X \times Y$ are Cartesian products of open sets in $X$ and $Y$, respectively.

**Example 4.6.2.** The product space $\mathbb{R} \times \mathbb{R}$ is functionally equivalent to $\mathbb{R}^2$ from many different perspectives: in particular for topology, they are homeomorphic when using the standard product topology.

If we take the set of open intervals as a basis of both copies of $\mathbb{R}$ used to construct $\mathbb{R}^2$, then, as pictured in Figure 4.11, the set of open rectangles forms a basis for $\mathbb{R}^2$. 

---

Figure 4.10: illustration of two-point intersections between $S^2$ and elements from $\mathbb{R}P^2$. 

---

51
Figure 4.11: Translation of open intervals as a basis of $\mathbb{R}$ to open rectangles as a basis of $\mathbb{R}^2$.

Figure 4.12: attaching a copy of $S^1$ to every point on an original instance of $S^1$ (e.g. the inner ring in this figure) forms the standard hollow torus in $\mathbb{R}^3$.

**Example 4.6.3.** The product space $S^1 \times S^1$ is the hollow torus in $\mathbb{R}^3$, as illustrated in Figure 4.12.

### 4.7 Topological Manifolds

In Carlsson’s research on the distribution of image patches and Carlsson and Gabrielsson’s research on the distribution of trained CNN kernels, some of the observations they made dealt with two fundamental mathematical objects: the circle and the Klein bottle. These are important examples of topological manifolds, which are defined below, along with one of their essential properties, being locally Euclidean of dimension $n$.

**Definition 4.7.1.** A topological space $X$ is locally Euclidean of dimension $n$ if for all $x \in X$, there is a neighborhood of $x$ homeomorphic to an open ball in $\mathbb{R}^n$. A $n$-dimensional topological manifold is a second countable Hausdorff space that is locally Euclidean of dimension $n$.

In this document we do not specifically refer to second countable or Hausdorff spaces, as in the work
described we do not deal with spaces that lack these properties. Second countable spaces have countable bases, and Hausdorff spaces give us the ability to separate any pair of points by a pair of disjoint neighborhoods. These are both desirable properties of a topological space for a number of reasons, and the space $\mathbb{R}^n$ itself possesses them, as well as many other spaces, including the aforementioned circle and Klein bottle, the general $n$-sphere, and the torus.

We introduce the defining properties of the Klein bottle and the torus through identification of the edges on a rectangle cut out from the 2-dimensional plane. Formally, this identification is done through a quotient map. However, from this point, we can think of this rectangle as a piece of paper. This process of edge identification is readily visualized by constructing a cylinder, then a torus, from the original rectangle.

By gluing two opposing edges of our piece of paper together, we obtain a cylinder. Further, if we also glue the other pair of opposing edges, this cylinder now becomes a torus. Gluing the edges together in this manner is an example of identifying edges while preserving their orientation. We can show this visually by marking the pairs of opposing edges using arrows that point in the same direction, as in Figure 4.13.

![Figure 4.13: visualization of the quotient map on a rectangle (i.e. $I \times I$) for defining a torus through edge identification, from [24].](image)

When considering the construction of new shapes by edge identification of a rectangle, the process of realizing the Klein bottle is, in a sense, very similar to that of the torus. However, the major difference between these two structures comes from how we identify one of the two edge pairs. When we construct the Klein bottle, instead of preserving the orientation for both edge pairs, we flip the orientation for exactly one of the two edge pairs. We can show this by drawing arrows on one of the edge pairs pointing in the same direction, while drawing arrows on the other edge pair pointing in opposite directions, as in Figure 4.14.

![Figure 4.14: visualization of a Klein bottle's definition by identifying edges, from [24].](image)

The resulting twist that is necessary to realize the Klein bottle as a manifold is the reason why it
takes a fourth dimension to truly construct it as a manifold. Since we live in a 3-dimensional world, the closest visualization we have to the Klein bottle is that shown above: a structure that intersects itself.

One particular observation of interest from Carlsson and Gabrielsson’s research involves the appearance of three circles, with intersection behavior resembling the Klein Bottle's characteristic "twist" identification. One of the major tools in TDA used in their research, persistent homology, while a powerful tool generally, has an interesting limitation when it comes to uncovering Klein bottle-like geometry. Later, when we dive into the fundamental concept of homology, we will see why persistent homology has this limitation and why further analysis was needed to connect the observed collection of data points to the Klein bottle.

4.8 Homotopy and Homotopy Equivalence

Note: for this and future sections in this chapter, assume all maps are continuous unless specified otherwise.

Homotopies and homotopy equivalences are essential tools in algebraic topology for defining a specific type of equivalence between topological spaces. As stated previously, this is a weaker form of equivalence than homeomorphism: in particular, one of its uses is to emphasize the appearance of holes in a topological space. The coffee cup and doughnut from Figure 4.1 each have one such hole, as stated before, which is the main property that suggests equivalence between them.

Before discussing homotopy equivalence of topological spaces, we start by defining homotopies of maps; this is the key to any application of homotopy to entire spaces.

For the rest of this chapter, we will use $I$ to represent the closed unit interval $[0, 1]$.

**Definition 4.8.1.** Start with two topological spaces $X$ and $Y$ and two maps $g : X \to Y$ and $h : X \to Y$. $g$ and $h$ are homotopic if, for $t \in I$ there is a family of maps $f_t : X \to Y$, where

1. $f_0 = g$,
2. $f_1 = h$, and
3. the map $F : X \times I \to Y$ defined by $F(x, t) = f_t(x)$ is continuous.

We will notate this as $g \simeq h$, and call the family $f_t$ a homotopy between $g$ and $h$.

Intuitively, we may choose think of this variable $t$ as a time measurement or an extra dimension of size 1 appended to elements of $f$’s domain $X$. Let’s illustrate this notion of homotopic maps with an example using the unit circle $S^1$ as a subspace of $\mathbb{R}^2$.

**Example 4.8.2.** Let $X$ be the "glued" version of $[0, 2\pi]$ from Proposition 4.4.1, where 0 and $2\pi$ are equivalent and $X$ is homeomorphic $S^1$. Now let $h : X \to \mathbb{R}^2$ be defined by $h(\theta) = (\cos(\theta), \sin(\theta))$, and let $g : x \to \mathbb{R}^2$ be the constant map $g(\theta) = (0, 0)$.

Define a family of functions

$$f_t(\theta) = ((1 - t)\cos(\theta), (1 - t)\sin(\theta)).$$

As visualized in Figure 4.15 below, we can pictorially represent this as a hollow cylinder with height 1 ($S^1 \times I$) mapped to smaller circles in $\mathbb{R}^2$ for larger values of the cylinder’s height value $t$.

Note that that $f_0 = h$, $f_1 = g$, and $f_t(\theta)$ is continuous in $(\theta, t)$. We conclude $g \simeq h$. 
Definition 4.8.3. A map $f$ that homotopic to a constant map is said to be null-homotopic.

We are now ready to start building toward the concept of homotopy equivalence of spaces, the fundamental equivalence relation between spaces specific to algebraic topology. As we will see, homeomorphism is a special case of homotopy equivalence.

Example 4.8.4. Let the family $f_t : I \to I$ be defined by $f_t(x) = (1-t)x$. Note that when $t = 0$, this is the identity map, while at $t = 1$, this is the zero map. Intuitively, changing $t$ amounts to "morphing" the image of $f_t$ between the unit interval $I$ and the singleton $\{0\}$.

This example illustrates a key homotopy, which can then be used to draw a homotopy equivalence between these two spaces. We now introduce this equivalence relation rigorously.

Definition 4.8.5. A map $f : X \to Y$ is a homotopy equivalence if there exists another map $g : Y \to X$ such that $g \circ f \simeq id_X$ and $f \circ g \simeq id_Y$. Under the maps $f$ and $g$, the spaces $X$ and $Y$ are said to be homotopy equivalent as well, we will notate this as $X \simeq Y$.

Let’s generalize the scenario started in Example 4.8.4, finishing our verification that the unit interval is homotopy equivalent to a point. Such spaces have an important name; we have used this term in informal contexts before, and we now introduce it rigorously.

Definition 4.8.6. A space homotopy equivalent to a point is called contractible.

Example 4.8.7. Let $x$ be any single point, and let $f : x \to I$ and $g : I \to x$ be any maps between their specified spaces. $(f \circ g)(y) = f(x)$ for all $y \in I$, and $(g \circ f)(x) = x$, and so both are constant maps. By Example 4.8.4, both are homotopic to their domains’ respective identity functions, and thus
are homotopy equivalences between $I$ and $x$.

We will conclude this general introduction to homotopy and homotopy equivalence by introducing a special type of homotopy. This homotopy aims to gradually "fold" a space $X$ toward a subspace $A$, as visualized in Figure 4.16: this is a special case of a deformation retraction, defined below.

Figure 4.16: basic intuitive demonstration of retracting or "folding" a space $X$ to a subspace $A$.

**Definition 4.8.8.** A deformation retraction of a space $X$ onto a subspace $A$ is a homotopy $f_t : X \to X$ such that

1. $f_0$ is the identity function on $X$.
2. $f_1(X) = A$
3. $f_t|_A = id_A$ for all $t \in I$, i.e. $f_t(a) = a$ for all $a \in A$.

Under this definition $f_1$ is a special case of a retraction, as defined below.

**Definition 4.8.9.** For a topological space $X$ and subspace $A$, a map $f : X \to X$ is a retraction if $r(X) = A$ and $r|_A = id_A$.

A key example relating to Carlsson and Gabrielsson’s work regarding a search for circles is shown below. Lee, Mumford, and Pedersen’s research relating to the distribution of image patches from a natural image dataset turned up an annulus of edge-type patches, but, as we’ll see below, from an algebraic topology perspective, we can identify such an annulus with the circle $S^1$.

**Example 4.8.10.** Let $X = S^1 \times [1, 2]$ and $f_t : X \to X$, note $X$ is an annulus of radius 2 whose central hole is the unit circle. Define $f_t(x, r) = (x, (1 - t)r + t)$. Then $f_t$ is a deformation retraction of $X$ onto $S^1 \times 1$, essentially onto $S^1$.

This homotopy “squishes” the image of $f_t$ from $S^1 \times [1, 2]$ toward the subspace $S^1 \times \{1\}$ while leaving all points on $S^1 \times \{1\}$ unchanged, as pictured in Figure 4.17.

An important connection between deformation retractions and homotopy equivalence is given by the following fact.

**Lemma 4.8.1.** Let $X$ be a topological space with subspaces $A, B$ such that there exist deformation retractions $f : X \to A$ and $g : X \to B$. Then $A \simeq B$. 

56
4.9 The Fundamental Group

Now that general notions involving the topology of spaces, continuity, and homotopy are now established, we now move on to describing algebraic objects that can be derived using them. These are topological invariants, and are specifically known as algebraic invariants of a space. This section will build toward an extremely important algebraic invariant: the fundamental group.

The underlying set for this group will involve loops on a space $X$: in particular, we will use the notion of homotopy to create equivalence classes for these loops via a relation $\sim$, and then divide out by this equivalence relation to obtain the group $X/\sim$. Informally, the group operation between two loops $f$ and $g$ will yield a loop $fg$, completed by traversing $f$, then $g$. Later in this section, we will show that this set equipped with this operation is indeed a group.

Let’s start by defining a path, general set of maps which includes loops. As usual for this chapter, let $I = [0, 1]$.

**Definition 4.9.1.** A path in a space $X$ is a (continuous) map $f : I \to X$. This space is path connected if for any $x, y \in X$, there exists a path $f$ such that $f(0) = x$ and $f(1) = y$. Additionally, a path component of $X$ is a maximal path-connected subspace of $X$.

**Example 4.9.2.** $\mathbb{R}^n$ is path connected, since any two points $x, y \in \mathbb{R}^n$ can be joined by the path $f(t) = (1-t)x + ty$. Note $f(I)$ in particular is the line segment with endpoints $x$ and $y$.

**Example 4.9.3.** A non-example of a path-connected space is $X = [0, 1] \cup [2, 3]$, as there does not exist a path $f$ where $f(0) = 1, f(1) = 2, \text{and } f(t) \in X$ for all $t \in I$.

An essential ingredient for defining the equivalence classes we will use in constructing the fundamental group is an application of the homotopy concept to paths. Note that a homotopy of paths will require an additional constraint compared to the weaker homotopy of maps: the endpoints of all paths in the homotopy must be conserved.
**Definition 4.9.4.** A *homotopy of paths* in a space $X$ is a family of paths $f_t : I \to X$, such that

1. For some fixed $x_0, x_1 \in X$, $f_t(0) = x_0$ and $f_t(1) = x_1$ for all $t \in I$.
2. The map $F : I \times I \to X$ defined by $F(s, t) = f_t(s)$ is continuous.

Figure 4.18 displays examples of paths from such a homotopy on a space $X$.

![Homotopy of Paths](image)

**Example 4.9.5.** For any convex set $S \subseteq \mathbb{R}^n$, any pair of paths $f_0, f_1$ with the same endpoints are homotopic.

**Example 4.9.6.** On the other hand, take the annulus $X$ shown in Figure 4.19, a non-convex set. The paths $f_0$ and $f_1$ have the same endpoints, but are not homotopic; here the hole in the middle is preventing any family of paths between $f_0$ and $f_1$ from being continuous.

We now formally define the product of two paths: this notion will be our general foundation for defining the group operation for the fundamental group.

**Definition 4.9.7.** For two paths $f$ and $g$ where $f(1) = g(0)$, the *product of paths* $f \cdot g$ is defined as follows:

$$(f \cdot g)(t) = \begin{cases} f(2t), & \text{if } 0 \leq t \leq .5 \\ g(2t - 1), & \text{if } .5 < t \leq 1. \end{cases}$$

An example is drawn in Figure 4.20.

An important property of this product definition is that it respects homotopy classes. This is what will allow us to equate homotopic paths when defining the fundamental group.

**Lemma 4.9.1.** For loops $f_0, f_1, g_0$, and $g_1$, if $f_0 \simeq f_1$ and $g_0 \simeq g_1$, then $f_0 \cdot g_0 \simeq f_1 \cdot g_1$. 

58
Figure 4.19: two paths on an annulus that are not homotopic, despite sharing a pair of endpoints.

Proof. Define the family
\[ h_t(s) = \begin{cases} f_t(2s), & \text{if } 0 \leq s < .5 \\ g_t(2s - 1), & \text{if } .5 \leq s \leq 1 \end{cases} \]

Let \([f]\) be the equivalence class of all paths homotopic to \(f\) and let \([g]\) be the equivalence class of all paths homotopic to \(g\). Then the product \([f] \cdot [g]\) is well-defined as \([h] = [f \cdot g]\).

Let’s now define the special paths which underpin the fundamental group.

**Definition 4.9.8.** A loop in a space \(X\) is a path \(f : I \to X\) such that \(f(0) = f(1)\). \(f(0)\) is the base point of the loop.

Note that \(I/\{0 \sim 1\}\) is homeomorphic to \(S^1\) (Proposition 4.4.1), so a loop is topologically equivalent to a map \(f : S^1 \to X\).

With this foundation, we are now ready to define the set that comprises the fundamental group, along with its group operation.

**Definition 4.9.9.** For a topological space \(X\), the set of all homotopy classes \([f]\) of loops \(f : I \to X\) at base point \(x_0\) is called the fundamental group and notated \(\pi_1(X, x_0)\).

It remains to be shown that \(\pi_1(X, x_0)\) is in fact a group. To start the verification process, we’ll start with a quick definition that will allow us to reconsider the ”rate” at which we traverse our loops.

**Definition 4.9.10.** Let \(f : I \to X\) be a path, and \(\phi : I \to I\) be any map where \(\phi(0) = 0\) and \(\phi(1) = 1\). Then \(f \circ \phi\) is called a reparametrization of \(f\).

However, by the next proposition, all reparametrizations of any loop \(f\) share a crucial property.

**Proposition 4.9.2.** For any path \(f : I \to X\) and reparametrization \(f \circ \phi\), \(f \circ \phi \simeq f\).

Proof. \(I\) is convex, which admits a homotopy \(\phi_t(s) = (1 - t)\phi(s) + ts\) between \(\phi\) and \(id_I\). It follows that \(f \circ \phi_t\) is also a homotopy between \(f \circ \phi\) and \(f\).
We now come to the theorem underlying the use of algebraic processes in studying homotopies of loops.

**Theorem 4.9.3.** The set $\pi_1(X, x_0)$, equipped with the operation $[f][g] = [f \cdot g]$ as defined in Definition 4.9.7, is a group.

**Proof.** We will show that under this multiplication of paths operation $\cdot$, there is an identity element, and each element of $\pi_1(X, x_0)$ has an inverse. Additionally, this operation is associative.

- **Identity:** We will show that the constant path $e : I \to X$, where $e(t) = x_0$, is an identity element under $\cdot$. Note that we can call it the identity once $\pi_1(X, x_0)$ is established as a group under path multiplication.

Let $f : I \to X$ be an arbitrary loop. Then

$$ (f \cdot e)(t) = \begin{cases} f(2t), & \text{if } 0 \leq x < 1/2 \\ x_0, & \text{if } 1/2 \leq x \leq 1, \end{cases} $$

a reparametrization of $f$ under $\phi : I \to I$, where

$$ \phi(s) = \begin{cases} 2s, & \text{if } 0 \leq s < 1/2 \\ 1, & \text{if } 1/2 \leq s \leq 1, \end{cases} $$

Since $f \cdot e$ is a reparametrization of $f$, we know that $f \cdot e$ is homotopic to $f$. This establishes $e$ as a right identity element of $\pi_1(X, x_0)$. A similar argument shows $e$ to be a left identity element as well, as

$$ (e \cdot f)(t) = \begin{cases} x_0, & \text{if } 0 \leq x < 1/2 \\ f(2t - 1), & \text{if } 1/2 \leq x \leq 1, \end{cases} $$
a reparametrization of $f$ under $\psi : I \to I$, where

$$
\psi(s) = \begin{cases} 
0, & \text{if } 0 \leq s < 1/2 \\
2s - 1, & \text{if } 1/2 \leq s \leq 1, 
\end{cases}
$$

Thus $e$ is an identity element of $\pi_1(X, x_0)$ under $\cdot$.

- Inverses: Start with an arbitrary loop $f : I \to X$, and define $\bar{f} : I \to X$ as $\bar{f}(s) = f(1 - s)$. To establish $\bar{f}$ as an inverse of $f$, we need to show that $f \cdot \bar{f}$ and $\bar{f} \cdot f$ are homotopic to $e$.

Now define the path $f_t : I \to X$ as

$$
f_t(s) = \begin{cases} 
f(s), & \text{if } 0 \leq s < 1 - t \\
f(1 - t), & \text{if } 1 - t \leq s \leq 1, 
\end{cases}
$$

and note that under the $\bar{f}$ convention defined earlier, we have

$$
\bar{f}_t(s) = \begin{cases} 
f(1 - t), & \text{if } 0 \leq s \leq t \\
f(1 - s), & \text{if } t < s \leq 1. 
\end{cases}
$$

We now let $h_t = f_t \cdot \bar{f}_t$, which implies $h_1 = e$ and $h_0 = f \cdot \bar{f}$. By defining $h_t$ in this way, we have constructed a homotopy from $f \cdot \bar{f}$ to $e$, establishing $\bar{f}$ as a right inverse of $f$ in $\pi_1(X, x_0)$. A visualization of the endpoints of this homotopy, along with an intermediate loop $h_{1/2}$, is shown in Figure 4.21.

![Figure 4.21: visualization of the homotopy $h_t$ between $f$ and $f \cdot \bar{f}$](image)

A similar argument shows $\bar{f}$ to be a left inverse of $f$ as well. Define $\bar{h}_t$ as $\bar{f}_t \cdot f_t$: $h_1 = e$ and $h_0 = \bar{f} \cdot f$, so $\bar{h}_t$ is a homotopy between $\bar{f} \cdot f$ and $e$.

Thus, $\bar{f}$ is an inverse element of $f$ in $\pi_1(X, x_0)$, i.e. $\bar{f} = f^{-1}$.

- Associativity: It remains to be shown that for arbitrary loops $f, g, h : I \to X$, $(f \cdot g) \cdot h$ is homotopic to $f \cdot (g \cdot h)$.

Note that, by our definition of $\cdot$,

$$
[(f \cdot g) \cdot h](s) = \begin{cases} 
f(s), & \text{if } 0 \leq s < 1/4 \\
g(s), & \text{if } 1/4 \leq s < 1/2 \\
f(1 - s), & \text{if } 1/2 \leq s \leq 1, 
\end{cases}
$$
Let \( \text{Definition 4.9.12.} \) of points. which relates fundamental groups based at two different points, provided a path exists between the pair the constant function at the origin. We also refer to this constant function as the trivial loop chase of base point map different fundamental groups of the same space. This is the foundation of the base point \( h(x_0) = x \) onto the point \( x \) \( \bar{c} \) by \( \bar{x} \). We see that \( \bar{c} \) \( \bar{x} \) \( \bar{c} \) \( \bar{x} \)

\[ \text{Example 4.9.11.} \ \pi_1(\mathbb{R}^n, 0) \text{ is trivial, as all loops with base point at the origin in } \mathbb{R}^n \text{ are homotopic to the constant function at the origin. We also refer to this constant function as the trivial loop.} \]

Let \( c_t : \mathbb{R}^n \to \mathbb{R}^n \) be defined by \( c_t(x) = tx \), where \( t \in I \), and let \( f : I \to \mathbb{R}^n \) be a loop based at the origin. We can see that \( c_t \circ f \) is then a homotopy of \( f \) onto the trivial path.

This can be generalized to any contractible space. If \( C \) is contractible with a deformation retraction onto the point \( x_0 \), then \( \pi_1(C, x_0) \) is the trivial group \( \{e\} \).

So far, all considerations of the fundamental group we’ve addressed have been dependent on the base point \( x_0 \). As we’ll see, there are some important situations in which this group becomes independent of the base point \( x_0 \).

Before getting into the details, we need to establish a map that we can use to relate elements of different fundamental groups of the same space. This is the foundation of the chase of base point map, which relates fundamental groups based at two different points, provided a path exists between the pair of points.

\[ \text{Definition 4.9.12.} \ \text{Let } f : I \to X \text{ be a loop based at } x_1, \text{ and let } h : I \to X \text{ be a path, where } h(0) = x_0 \text{ and } h(1) = x_1. \text{ The change of base point map is the map } \beta_h : \pi_1(X, x_0) \to \pi_1(X, x_1) \text{ defined by } \beta_h(f) = h \cdot f \cdot \bar{h}, \text{ where } \cdot \text{ is the operation from Definition 4.9.7 and } \bar{h}(t) = h(1 - t). \]

A crucial property of this map is given below.

\[ \text{Proposition 4.9.4.} \ \text{The change of base point map } \beta_h \text{ is an isomorphism between } \pi_1(X, x_0) \text{ and } \pi_n(X, x_1). \]

\[ \text{Proof.} \ \text{We'll start by showing that } \beta_h \text{ is a homomorphism, i.e. that } \beta_h(f \cdot g) = \beta_h(f) \cdot \beta_h(g). \text{ Note that } \]

\[ \beta_h(f \cdot g) = [h \cdot f \cdot g \cdot \bar{h}] = [h \cdot f \cdot h \cdot g \cdot \bar{h}] = [h \cdot f \cdot h] \cdot [h \cdot g \cdot \bar{h}] = \beta_h(f) \cdot \beta_h(g). \]

Note also that \( (\beta_h \circ \beta_h)(f) = [h \cdot \bar{h} \cdot f \cdot h \cdot \bar{h}] = [f] \), establishing \( \beta_h \) as an inverse of \( \beta_h \). Thus \( \beta_h \) is an isomorphism.
The important conclusion of these facts is that, for any path-connected space \( X \), the fundamental group of \( X \) based at any point \( x_0 \) is independent of \( x_0 \). This allows us to use the more general notation \( \pi_1(X) \) when dealing with a path-connected space \( X \).

The notion of a fundamental group also allows us to draw conclusions about maps between two different spaces \( X \) and \( Y \). We will now cover a special type of map between the fundamental groups of spaces called an \emph{induced homomorphism}, starting with some essential terms and facts.

\textbf{Definition 4.9.13.} Let \( X \) be a space containing a point \( x_0 \). We call the pair \((X, x_0)\) a \emph{pointed space}. Additionally, for two pointed spaces \((X, x_0)\) and \((Y, y_0)\), a map \( \phi : (X, x_0) \to (Y, y_0) \) with \( \phi(x_0, x_0) = (y_0, y_0) \) is called a \emph{pointed map}.

From this point forward, assume (unless stated otherwise) that all maps of the form \( \phi : (X, x_0) \to (Y, y_0) \) are pointed maps.

We can use pointed maps to relate loops in different spaces, since if \( f : I \to X \) is a loop based at \( x_0 \) and \( \phi : (X, x_0) \to (Y, y_0) \) is a pointed map, then \( \phi \circ f \) is a loop based at \( y_0 \). By considering \( \phi \circ f \) for a homotopy of paths \( f_t \), we get a homotopy of paths in \( Y \), which leads to the following well-defined map between fundamental groups of \( X \) and \( Y \).

\[ \phi_* : \pi_1(X, x_0) \to \pi_1(Y, y_0), \text{where } \phi_*([f]) = [\phi \circ f]. \]

Some other key properties of \( \phi_* \) are listed below:

1. \( \phi_* \) is a homomorphism.
2. If \( \phi : (X, x_0) \to (X, x_0) \) is the identity map, then \( \phi_* \) is the identity map on \( \pi_1(X, x_0) \).
3. If \( \phi_1 : (X, x_0) \to (Y, y_0) \) and \( \phi_2 : (Y, y_0) \to (Z, z_0) \) are pointed maps, then \( (\phi_2 \circ \phi_1)_* = \phi_2_* \circ \phi_1_* \).

We’ll now connect the concept of fundamental groups with homotopy equivalence between spaces.

\textbf{Definition 4.9.14.} Pointed spaces \((X, x_0)\) and \((Y, y_0)\) are \emph{pointed homotopy equivalent} if there exist maps \( \phi_1 : (X, x_0) \to (Y, y_0) \) and \( \phi_2 : (Y, y_0) \to (X, x_0) \) such that \( \phi_1 \circ \phi_2 \) and \( \phi_2 \circ \phi_1 \) are both homotopic to the identity function on their respective spaces. In particular, the homotopy used for \( \phi_2 \circ \phi_1 \) must fix the point \( x_0 \), and the homotopy used for \( \phi_1 \circ \phi_2 \) must fix the point \( y_0 \).

An important consequence of pointed homotopy equivalence follows.

\textbf{Proposition 4.9.5.} If two pointed homotopy equivalent spaces \((X, x_0)\) and \((Y, y_0)\), then the fundamental groups \( \pi_1(X, x_0) \) and \( \pi_1(Y, y_0) \) are isomorphic.

\textit{Proof.} Consider a loop \( f : I \to X \) and let \( \phi_1 : (X, x_0) \to (Y, y_0) \) and \( \phi_2 : (Y, y_0) \to (X, x_0) \) be inverse pointed homotopy equivalences. Note that \( \phi_2 \circ \phi_1 : I \to X \) is a loop in \( X \) based at \( x_0 \).

Since \( \phi_1 \) and \( \phi_2 \) are inverse homotopy equivalences, we have a base point fixing homotopy \( \phi_2 \circ \phi_1 \), such that \( (\phi_2 \circ \phi_1)_0 = \phi_2 \circ \phi_1 \) and \( (\phi_2 \circ \phi_1)_1 = \text{id}_X \). This establishes \( (\phi_2 \circ \phi_1)_t \circ f \) as a homotopy between \( \phi_2 \circ \phi_1 \circ f \) and \( f \).

Since, as we’ve seen, \( (\phi_2 \circ \phi_1)_* = \phi_2_* \circ \phi_1_* \) and individually, \( \phi_1_* \) and \( \phi_2_* \) are homomorphisms, we conclude that \( \phi_1_* \) and \( \phi_2_* \) are isomorphisms.

\[ \square \]

\subsection{Fundamental Group of the Circle}

In the TDA tools that we will be using to investigate CNNs, especially persistent homology, we will be focusing on the first homology of a topological space, which, as we will see, is very closely related to the fundamental group. In particular, the fundamental group and the first homology are designed
to investigate a space for holes. Homology also describes aspects of a space beyond that of so-called one-dimensional holes, but before introducing homology as a general tool, we will conclude discussion of the fundamental group with an essential case-study example: the fundamental group of the circle $S^1$, or $\pi_1(S^1)$.

We will see that the fundamental group of the circle is non-trivial, and in particular is isomorphic to $\mathbb{Z}$ with addition as its group operation. Intuitively, the integers correspond to counting repeated trips around the circle, representing trips in one direction (say counterclockwise) as positive and trips in the other direction as negative.

Below, we will state this important relationship between $\pi_1(S^1)$ and $\mathbb{Z}$ formally, along with the exact isomorphism between them.

**Theorem 4.9.6.** $\pi_1(S^1)$ is isomorphic to $\mathbb{Z}$ equipped with the addition operation, by the isomorphism $f : \mathbb{Z} \to \pi_1(S^1)$, where $f(n) = g_n$ and $g_n : I \to S^1$ is given by $g_n(t) = (\cos(2\pi nt), \sin(2\pi nt))$.

Computing this group for the circle $S^1$ directly turns out to be a surprisingly difficult task, but it has many rewarding applications, such as its use in a proof of the Fundamental Theorem of Algebra.

Our first step in proving Theorem 4.9.6 will be to introduce the notion of a covering space. For a space $X$, such a covering space will provide a way of more easily distinguishing between loops that cover the same area multiple times, for example looping around $S^1$ 4 times vs 5 times.

**Definition 4.9.15.** Let $X$ be a topological space. A covering space of $X$ is a space $\tilde{X}$ together with a map $p : \tilde{X} \to X$ satisfying the following:

1. For all $x \in X$, there is an open set $U \subseteq X$ such that $x \in U$ and $p^{-1}(U)$ is a disjoint union of open sets in $\tilde{X}$, i.e. for a sequence $\tilde{U}_i \subseteq \tilde{X}$ and index set $I$, 
   $$p^{-1}(U) = \bigsqcup_{i \in I} \tilde{U}_i.$$

2. For all $i$ in the index set, $p(\tilde{U}_i) = U$ and $p|_{\tilde{U}_i}$ is a homeomorphism.

Each such open set $U$ is called evenly covered. From this point forward, we will use the notation $p : \tilde{X} \to X$ to establish covering spaces for a general space $X$.

A crucial example of such a covering space for $S^1$ is given below.

**Example 4.9.16.** Consider drawing $\mathbb{R}$ as a helix $\tilde{S}^1$ in $\mathbb{R}^3$ via the map $s \mapsto (\cos(2\pi s), \sin(2\pi s), s)$, as visualized in Figure 4.22.

The desired projection map $p : \tilde{S}^1 \to S^1$ is then given by $p(x, y, z) = (x, y)$. Note that for any open set $U \subseteq S^1$, $p^{-1}(U)$ is a corresponding union of open sets in $S^1$, as visualized in Figure 4.23. An open arc $U$ in $S^1$ is denoted in red on the right side of the figure, and its pre-image is denoted in red on the left side.

An important companion map to the projection map from the covering space is called a lift, as defined below. This is a map into the covering space evaluated before projecting to the covered space.

**Definition 4.9.17.** Let $A, X$ be topological spaces, $p : \tilde{X} \to X$ be a covering space of $X$, and $f : A \to X$ be a map. A lift of $f$ is a map $\tilde{f} : A \to \tilde{X}$ such that $p \circ \tilde{f} = f$.

Covering spaces satisfy an essential property known as the homotopy lifting property, described in the proposition below.

**Proposition 4.9.7.** Let $X, Y$ be topological spaces, where $X$ has a covering space $p : \tilde{X} \to X$. Additionally, let $F : Y \times I \to X$ be a map and let $\tilde{F}_0 : Y \times \{0\} \to \tilde{X}$ be a lift of $F|_{Y \times \{0\}}$. Then there exists a unique map $\tilde{F} : Y \times I \to \tilde{X}$ that is both a lift of $F$ and admits $\tilde{F}|_{Y \times \{0\}} = \tilde{F}_0$. 

64
Figure 4.22: map from the real number line to a helix in $\mathbb{R}^3$.

Proof. Take a point $y_0 \in Y$, along with an open set $N \subseteq Y$ such that $y_0 \in N$: $N$ is also called a neighborhood of $y_0$. Our first step is to construct a special such neighborhood $N$, then construct a lift on $N \times I$.

Recall that we assume all maps in this chapter to be continuous unless stated otherwise: this includes $F$. Further note that $Y \times I$ has the product topology, so that open sets in $Y \times I$ are precisely products of open sets in $Y$ and open intervals in $I$. Finally, note that by Definition 4.9.15, $F(y_0, t)$ has an evenly covered open neighborhood in $X$ for all $t \in I$, and since $F$ is continuous, the pre-image of each such neighborhood is also open.

It then follows that each point of $y_0 \times I$ has a neighborhood $N_t \times (a_t, b_t)$ (where $(a_t, b_t)$ is an open interval in $I$) such that $F(N_t, (a_t, b_t))$ is evenly covered. A well-known property of the closed interval $I$ is that it is compact, i.e. for any union of open subsets $I_i$ such that $I \subseteq \bigcup_{i=1}^n I_i$, there is a finite subset $\{I_1 \ldots I_n\}$ such that $I \subseteq \bigcup_{i=1}^n I_i$. This implies that a finite union of neighborhoods of the form $F(N_{I_i}, (a_t, b_t))$ contains $y_0 \times I$.

We now intersect all of the $N_I$ components of this finite set of neighborhoods to obtain a single set in $Y$ which we call $N$. Consider a set $\{t_1 \ldots t_{m-1}\} \subset I$ where $t_i < t_{i+1}$, and let $I_0 = [0, t_2)$, $I_{m-2} = (t_{m-2}, 1]$, and $I_i = (t_{i-1}, t_{i+2})$ for $i \in \{1 \ldots m-3\}$. This admits a union

$$N \times I = \bigcup_{i=0}^{m-2} N \times I_i$$

that contains $y_0 \times I$ and where each image $F(N \times I_i)$ is evenly covered in $X$.

Next we construct the desired lift $\tilde{F}$ restricted to $N \times I_0$. To do this, first note that $p^{-1}(F(N \times I_0))$ is a disjoint union of the form $\bigcup \tilde{U}_k$, since $F(N \times I_0)$ is evenly covered. Recall that $\tilde{F}_0$, a lift on $Y \times \{0\}$, is already given in the hypotheses of the proposition, and so we define $\tilde{U}_i$ to be the component of the above union containing $\tilde{F}_0(N \times \{0\})$. This allows us to declare $\tilde{F}(N \times I_0) = p^{-1}|_{\tilde{U}_i}(F(N \times I_0))$.

Since the subintervals $I_i$ and $I_{i+1}$ of $I$ have a nonempty intersection, we may proceed inductively to construct the lift $\tilde{F}$ for all of $N \times I$. Additionally, since $y_0$ was an arbitrary point in $Y$, we have proved the existence of such a lift for the entirety of $Y$. 

65
It remains to show that the map $\tilde{F}$ we have constructed is unique given $\tilde{F}_0$. We’ll start by taking any point $y_0 \in Y$ and showing that for any map $F : \{y_0\} \times I \to X$ (we’ll state this equivalently as $F : I \to X$), any lift of $F$ is unique given a fixed $\tilde{F}(0)$.

Let $\tilde{F}$ and $\tilde{F}'$ be two lifts such that $\tilde{F}(0)$ is known and $\tilde{F}(0) = \tilde{F'}(0)$. We’ll then partition $I$ into the same neighborhoods $I_i$ as before, so that $F(I_i)$ is again evenly covered for all $i \in \{1 \ldots m - 2\}$. As we have before, consider $F(I_0)$, noting that it is evenly covered, then from its covering set in $\tilde{X} \{\tilde{U}_k\}$, select the $\tilde{U}_i$ containing $\tilde{F}(0)$. Since the projection map $p$ is then a homeomorphism between $\tilde{U}_i$ and $F(I_0)$, we conclude that $\tilde{F}(I_0)$ and $\tilde{F'}(I_0)$ are both subsets of $\tilde{U}_i$, and further, that $\tilde{F}|_{I_0} = \tilde{F'}|_{I_0}$. Thus, fixing the lift $\tilde{F}$ implies that $\tilde{F}|_{I_0}$ is unique.

This process continues inductively for all $I_i$ by again noting that $I_i$ and $I_{i+1}$ have a nonempty intersection.

To summarize, we have shown the existence of a lift homotopy $\tilde{F}$ on a neighborhood of an arbitrary point $y_0 \in Y$, then shown that $\tilde{F}$’s restriction to $\{y_0\} \times I$ is unique. These ingredients allow us to uniquely patch together the desired unique lift homotopy $\tilde{F} : Y \times I \to \tilde{X}$.

Some key corollaries follow from the homotopy lifting property.

**Corollary 4.9.8.** Let $f : I \to X$ be a path and let $x_0 = f(0)$. For all $\tilde{x}_0 \in p^{-1}(x_0)$, there exists a unique lift $\tilde{f} : I \to \tilde{X}$ such that $\tilde{f}(0) = \tilde{x}_0$.

This follows very closely from the uniqueness argument from Proposition 4.9.7.

**Corollary 4.9.9.** Start a covering space $p : \tilde{X} \to X$ and a homotopy of paths $f_t : I \to X$, where $f_t(0) = x_0$ and $f_t(1) = x_1$. Take a point $\tilde{x}_0 \in p^{-1}(x_0)$. Then there is a unique lifted homotopy of paths $\tilde{f}_t : I \to \tilde{X}$ where $\tilde{f}_t(0) = \tilde{x}_0$.

The proof of this second corollary is less trivial than the first given the arguments from Proposition 4.9.7.
Proof. Let $F : I \times I \to X$ be defined by $F(s,t) = f_t(s)$. Corollary 4.9.8 then admits a unique lift of $F|_{I \times \{0\}}$. This can be extended by Proposition 4.9.7 to a lift $\tilde{F} : I \times I \to \tilde{X}$. Then by the uniqueness of the lifted path from Corollary 4.9.8, we conclude that $\tilde{F}|_{I \times \{0\}}$ and $\tilde{F}|_{I \times \{1\}}$ must be constant paths, i.e. $\tilde{f}_t$'s endpoints are fixed for all $t \in I$, making it a true homotopy of paths.

Let's now finally return to our discussion of the fundamental group $\pi_1(S^1)$ and how this relates to the logical space very closely related to loops. Cycles will be defined rigorously as we elaborate on the algebraically-informed topological invariant of a space: homology. Instead of dealing with homotopy groups directly, topological data analysis often works with another logical space very closely related to loops. Cycles will be defined rigorously as we elaborate on the algebraically-informed topological invariant: homology.

The homology of a topological space $S$ is a sequence of homology groups

$$H_0(S), H_1(S), H_2(S) \ldots$$

Each of these homology groups $H_k(S)$ is generated by a basis of special non-trivial cycles, and is referred to as the $k^{th}$ homology group or $k^{th}$ homology. In particular, each $H_k(S)$ we will observe, both in explanatory examples and research examples, is isomorphic to

$$\mathbb{Z}^n \oplus T_1 \oplus \cdots \oplus T_m$$

for some non-negative integers $m, n$. Here $n$ is the rank of $H_k(S)$, also referred to as the $k^{th}$ Betti number $\beta_k(S)$, and the non-$\mathbb{Z}$ groups $T_i$ are finite.

4.10 Homology

Instead of dealing with homotopy groups directly, topological data analysis often works with another algebraically-informed topological invariant of a space: homology, built on cycles, structures in a topological space very closely related to loops. Cycles will be defined rigorously as we elaborate on the process of computing homology groups.

The homology of a topological space $S$ is a sequence of homology groups

$$H_0(S), H_1(S), H_2(S) \ldots$$

Each of these homology groups $H_k(S)$ is generated by a basis of special non-trivial cycles, and is referred to as the $k^{th}$ homology group or $k^{th}$ homology. In particular, each $H_k(S)$ we will observe, both in explanatory examples and research examples, is isomorphic to

$$\mathbb{Z}^n \oplus T_1 \oplus \cdots \oplus T_m$$

for some non-negative integers $m, n$. Here $n$ is the rank of $H_k(S)$, also referred to as the $k^{th}$ Betti number $\beta_k(S)$, and the non-$\mathbb{Z}$ groups $T_i$ are finite.
As we will see demonstrated in a 1-dimensional homology example, a basis of $H_k(S)$ can be interpreted as the number of unique $k$-dimensional "holes," or non-trivial cycles, in $S$. This hole-counting property is a major similarity between homology and homotopy, and in particular, we will discuss a crucial rigorous connection between the first homology $H_1(S)$ and the fundamental group $\pi_1(S)$.

We will describe two alternatives for computing the homology of a space, called simplicial homology and singular homology. In particular, simplicial homology is a special case of singular homology, but here we will be focused primarily on simplicial homology, an iterable combinatorial process that lends itself to straightforward computer implementation. Simplicial homology requires the construction of a special topological space called a simplicial complex, and our use of persistent homology, a TDA tool, makes use of these simplicial complexes.

### 4.10.1 Simplicial Homology

A simplicial complex is a structure formed from a collection of simplices. A simplex (singular of simplices) is generated by a set of $k + 1$ affinely independent points in $\mathbb{R}^n$, where $n \geq k$, by considering the convex hull of these $k + 1$ points. For a choice of $k$, this is called a $k$-simplex. Alternatively, we can use the definition from [4]:

**Definition 4.10.1.** A $k$-simplex (in standard form) is the set in $\mathbb{R}^{k+1}$ where

$$x_0 + x_1 + \cdots + x_k = 1$$

and $x_0, x_1, \ldots, x_k \geq 0$.

A subset of a $k$-simplex which is itself a simplex is called a face of the simplex.

This makes a simplex a generalization of a tetrahedron: a 3-simplex is a traditional tetrahedron, while a 2-simplex is a triangle, a 1-simplex is a line segment, a 0-simplex is a single point, and for $k > 3$, a $k$-simplex is a non-visualizable higher-dimensional structure similarly composed of triangles and tetrahedra.

**Definition 4.10.2.** A simplicial complex is a collection of simplices $K$ in $\mathbb{R}^n$ such that

1. For all $\sigma \in K$, all faces of $\sigma$ are also in $K$.
2. For any two simplices $\sigma_1, \sigma_2 \in K$, $\sigma_1 \cap \sigma_2 = \emptyset$ or $\sigma_1 \cap \sigma_2$ is a common face of $\sigma_1$ and $\sigma_2$.

Such a simplicial complex is visualized below in Figure 4.24. Note the inclusion of the endpoints of each simplex; this demonstrates the need to include each 0-simplex in the complex. Figure 4.25 contrasts two potential situations involving intersecting 2-simplices, one of which violates the definition of a simplicial complex and one of which does not.

We now build the definition for the $k^{th}$ homology group of a simplicial complex. To start, we consider the set of all $k$-simplices in such a complex. Using this set of $k$-simplices as a basis, we then construct a free Abelian group $C_k$ using the set of $k$-simplices as a basis.

**Definition 4.10.3.** $k$-chains are the elements of $C_k$, i.e. elements of the form

$$\sum_{i=1}^{m} n_i \sigma_i,$$

where $n_i \in \mathbb{Z}$ and $\sigma_i$ is an element of the simplicial complex. $C_k$ is also called the $k^{th}$ chain group of the simplicial complex.

Now that we have a way of combining $k$-simplices to form $k$-chains, let’s now consider a special property of a $k$-chain which will be immensely helpful in counting holes in our simplicial complex: the boundary. In order to define the boundary of a $k$-chain, we define the boundary map on $C_k$, notating a $k$-simplex generated by vertices $x_0, x_1 \ldots x_k$ as $[x_0, x_1 \ldots x_k]$. 

68
Figure 4.24: An example simplicial complex.

Figure 4.25: When two 2-simplices intersect in a simplicial complex, they may only intersect in a common face (as shown on the left): note that the figure on the right depicts an intersection that is not a common face.

Definition 4.10.4. The boundary map on $C_k$ is the homomorphism $\delta_k : C_k \rightarrow C_{k-1}$, defined on each $k$-simplex in $C_k$ as

$$\delta_k([x_0, x_1 \ldots x_k]) = \sum_{i=0}^{k} (-1)^i [x_0, x_1 \ldots \hat{x}_i \ldots x_k]$$  \hspace{1cm} (4.1)

where $[x_0, x_1 \ldots \hat{x}_i \ldots x_k]$ is the $k-1$-simplex generated by $x_0, x_1 \ldots x_k$ with $x_i$ removed. For any $k$-chain $x$, $\delta_k(x)$ is a $k-1$-boundary.

Note that since $\delta_k$ is a homomorphism and the set of simplices in our complex is a basis for $C_k$, we now have a definition for $\delta_k(x)$ for all $x \in C_k$.

Note further that if we define $C_k$ on the field $\mathbb{Z}/2\mathbb{Z}$, we can drop the $(-1)^i$ from this expression, as $-1 = 1$ in $\mathbb{Z}/2\mathbb{Z}$. However, if $C_k$ is defined on a different field, where $-1 \neq 1$, we can intuitively think of $-[x_0, x_1, \ldots, x_k]$ as the opposite orientation of $[x_0, x_1, \ldots, x_k]$.

We now introduce two special subgroups of $C_k$, which we will notate as $Z_k$, the kernel of $\delta_k$, and $B_k$, the image of $\delta_{k+1}$. $B_k$ is then the space of $k$-boundaries, i.e. boundaries of $k+1$-chains. On the other hand, we will refer to the elements of $Z_k$ as $k$-cycles: these are $k$-chains that have a boundary of 0. As we will show below, there is an important fact about these two subgroups that is key in defining the homology group $H_k(S)$ as a tool for counting $k$-dimensional homological features, or holes.

Theorem 4.10.1. $\delta_k \circ \delta_{k+1} = 0$. 

69
This can be shown as follows. For an arbitrary $k + 1$-simplex $[x_0, x_1, \ldots, x_{k+1}]$, we have
\[
\delta_k(\delta_{k+1}([x_0, x_1, \ldots, x_{k+1}])) = \delta_k\left(\sum_{i=0}^{k+1} (-1)^i x_0, x_1, \ldots, \hat{e}_i, \ldots, x_k \right)
\]
\[
= \sum_{i=0}^{k+1} (-1)^i \delta_k([x_0, x_1, \ldots, \hat{e}_i, \ldots, x_k])
\]
\[
= \sum_{i=1}^{k+1} \sum_{j=0}^{i-1} (-1)^{i+j}[x_0, \ldots, \hat{x}_j, \ldots, \hat{x}_i, \ldots, x_{k+1}]
\]
\[
+ \sum_{j=1}^{k+1} \sum_{i=0}^{j-1} (-1)^{i+j-1}[x_0, \ldots, \hat{x}_i, \ldots, \hat{x}_j, \ldots, x_{k+1}]
\]
\[
= 0,
\]
as $(-1)^{i+j-1} = -(-1)^{i+j}$.

As an immediate consequence,

**Corollary 4.10.2.** $B_k$ is a subgroup of $Z_k$.

In other words, every $k$-boundary is a $k$-cycle. This confirmation that $B_k \subseteq Z_k$ allows us to state a direct definition of the homology group $H_k(S)$ (letting $S$ be a simplicial complex).

**Definition 4.10.5.** The $k^{th}$ homology group $H_k(S)$ for a simplicial complex $S$ is the quotient group $Z_k/B_k$.

Through the lens of simplicial homology, all $k$-cycles that differ only by a $k$-boundary are equivalent, and all boundaries are equivalent to the identity element in $Z_k/B_k$.

Let’s look specifically at the 1-dimensional homology group $H_1(S)$, as discussed by [4], of a topological space $R$ through a topologically equivalent simplicial complex $S$, known as a triangulation of $R$. This will be particularly relevant to Carlsson and Gabrielsson’s findings, as it will shed light on the treatment of 1-dimensional holes in a simplicial complex as equivalent to circles in a general topological space. $R$ and $S$ are visualized by [4] as in Figure 4.26.

![Figure 4.26: a space R and its triangulation S, from [4]](image)

A quick inspection of $S$ (or $R$) shows that it has two holes, and this feature will be reflected in its 1st Betti number. However, to cement this notion, we need to demonstrate that $H_1(S)$ has a dimension of 2, i.e. there are two non-equivalent cycles in $Z_1/B_1$ that span the homology group. It’s helpful here to note an intuitive way of interpreting the addition of simplices in $C_k$; we can think of this addition as drawing paths along these simplices, and of cycles as paths starting and ending at the same point.
Using this intuition, Figure 4.27 shows an example of two non-equivalent cycles. The righthand cycle represents the boundary of a 2-chain (its borders are entirely filled in by points in the set), while the lefthand cycle traverses the 1-dimensional hole on the left and thus is not a boundary.

![Figure 4.27](image)

**Figure 4.27**: a visualization of two cycles: the right cycle is a boundary, while the left cycle is not, from [4].

Figure 4.28 is an example of two equivalent paths in $H_1(S)$.

![Figure 4.28](image)

**Figure 4.28**: two cycles traversing the same hole in $S$, from [4].

Since both paths negotiate the same hole in the simplicial complex $S$, they differ only by the boundary of the 2-chain shown Figure 4.29.

![Figure 4.29](image)

We can see that cycles whose borders contain only points in the set $S$ are all equivalent to the identity element in $H_1(S)$ (as they are boundaries), and so all non-zero cycles in $H_1(S)$ must in some way negotiate at least one of the two holes in the complex. Additionally, one can verify that any cycle that rounds the lefthand hole without rounding the righthand hole is homologically distinct from any cycle that rounds the righthand hole without rounding the lefthand hole. This intuitively leads to the fact that these distinct holes outlined in Figure 4.30 form a basis for $H_1(S)$, telling us that $S$ (and therefore $R$) has a 1st Betti number of 2.

This application of simplicial homology is the process carried out by the persistent homology algorithm we used when analyzing our dataset of trained CNN kernels. However, while this iterable, combinatorial process of computing homology is convenient for programming contexts, it lacks generalizability to spaces such as the torus and the Klein bottle, where finding a triangulation is a rather arduous process. In order to make homology computations for such spaces more manageable, we can use similar computational steps, but before making those computations, we need to loosen our restrictions on what constitutes an appropriate $k$-chain.
4.10.2 $\Delta$ (delta)-complexes and the Major Connection between the First Homology and Homotopy

[14] uses the more flexible concept of the $\Delta$-complex when explaining the fundamentals of simplicial homology. This generalization focuses on a general space $X$, using special maps from the $k$-simplex to $X$ as generators for $k$-chains rather than $k$-simplices themselves. One of the essential criteria for such a map to be considered part of the complex deals with the open simplex of its domain. To quote [14], and denoting the $k$-simplex as $\Delta^k$,

**Definition 4.10.6.** "The union of all the faces of $\Delta^k$ is the boundary of $\Delta^k$, or $\partial \Delta^k$. The open simplex $\overset{\circ}{\Delta}^k$ of $\Delta^k$ is $\Delta^k - \partial \Delta^k$, the interior of $\Delta^k$.

We again borrow from [14] to fully define the $\Delta$-complex.

**Definition 4.10.7.** "A $\Delta$-complex structure on a space $X$ is a collection of maps $\sigma_\alpha : \Delta^k \to X$, with $n$ depending on the index $\alpha$, such that:

1. The restriction $\sigma_\alpha|_{\overset{\circ}{\Delta}^k} : \overset{\circ}{\Delta}^k \to X$ is injective, and each point of $X$ is in the image of exactly one such restriction $\sigma_\alpha|_{\overset{\circ}{\Delta}^k}$.

2. Each restriction of $\sigma_\alpha$ to a $[k-1]$-dimensional face of $\Delta^k$ is one of the maps $\sigma_\beta : \Delta^{k-1} \to X$. Here we are identifying the face of $\Delta^k$ with $\Delta^{k-1}$ by the canonical linear homeomorphism between them that preserves the ordering of the vertices."
3. A set \( A \subseteq X \) is open if and only if \( \sigma^{-1}_\alpha(A) \) is open in \( \Delta^n \) for each \( \sigma_\alpha \).

When using the \( \Delta \) complex to compute homology groups, these maps \( \sigma_\alpha \) will be the objects used to generate the chain groups \( C_k \) rather than the \( k \)-simplices of a triangulation, as in strict simplicial homology. Additionally, we define the boundary map and our key subspaces of \( C_k \) in a similar way to how we did for simplicial homology.

**Definition 4.10.8.** In a \( \Delta \)-complex structure, the boundary map on \( C_k \) is the homomorphism \( \delta_k : C_k \to C_{k-1} \), defined on each \( \sigma_\alpha \) as

\[
\delta_k(\sigma_\alpha|_{[x_0,x_1,...x_k]}) = \sum_{i=0}^{k} (-1)^i \sigma_\alpha|_{[x_0,x_1,...\hat{x}_i,...x_k]}
\]

where again, \([x_0,x_1,...\hat{x}_i,...x_k]\) is the \( k-1 \)-simplex generated by \( x_0, x_1,...x_k \) with \( x_i \) removed.

The cycle group \( Z_k \) is defined as \( \ker(\partial_k) \) and the boundary group \( B_k \) is defined as \( \text{im}(\partial_{k+1}) \), as before, and the homology group \( H_k \) is, again, \( Z_k/B_k \).

As an aside, if we instead allow these maps \( \sigma_\alpha \) to include any continuous map from the \( k \)-simplex to \( X \), and again use these maps as the objects from which each \( C_k \) is generated, then each map is known as a singular \( k \)-simplex and the objects from 4.10.8 are defined in an analogous way. When homology computations are done using such singular \( k \)-simplices, this method is known as singular homology.

Crucially, this allows us plenty of freedom in how we are able to reconstruct a space using \( k \)-simplices: in particular, it allows us to make certain identifications between simplices that are not appropriate in a simplicial complex. This will be demonstrated in the following homology computation examples: here we use \( \Delta \)-complex computations rather than singular homology. We recommend the University of New South Wales course by N.J. Wildberger\(^4\) on algebraic topology for additional such \( \Delta \)-complex example computations.

**Example 4.10.9.** Under a \( \Delta \)-complex structure, the circle \( S^1 \)'s homology groups can be computed using a single 0-simplex and a single 1-simplex, as shown in Figure 4.31.

![Figure 4.31: \( \Delta \)-complex visualization of \( S^1 \).](https://www.youtube.com/playlist?list=PL6763F57A61FE5E8)

Note that if we were to use a simplicial complex, this would require three points, as simplicial complexes do not allow us to “glue” endpoints of a 1-simplex, i.e. identify them through a quotient map. This, as mentioned before, is a significant advantage of using the more general \( \Delta \)-complex structure. Let’s compute the chain, cycle, boundary, and homology groups of \( S^1 \), noting their isomorphisms free Abelian groups on the integers. To denote free Abelian groups generated by elements \( a_0, a_1,...a_n \), we will use the notation \( <a_0, a_1,...a_n> \).

\(^4\)https://www.youtube.com/playlist?list=PL6763F57A61FE5E8
The chain group $C_0$ is then $\langle x \rangle$, the free Abelian group generated by the points $x$; since this is only one element, $C_0 \cong \mathbb{Z}$. This makes $Z_0 = \langle x \rangle$ as well, since $\partial_0$ is the zero map, i.e. $C_0 = Z_0$ and $Z_0 \cong \mathbb{Z}$.

To compute $B_0$, we need to look at $\partial_1$. Since we only have one 1-simplex, which we’re calling $a$, we just look at $\partial_1(a)$; however, $a$’s endpoints are both $x$, so $\partial_1(a) = x - x = 0$, and so $B_0 = \langle 0 \rangle$. Thus $H_0 = \langle x \rangle / \langle 0 \rangle$, i.e. $H_0 \cong \mathbb{Z}$.

Again, we only have one 1-simplex $a$, and so $C_1 = \langle a \rangle$. Additionally, since $\partial_1 = 0$ as we know, $Z_1 = C_1$, and so $Z_1 \cong \mathbb{Z}$.

This structure on $S^1$ does not allow for any $k$-simplices with $k \geq 2$, so $B_1 = \langle 0 \rangle$. Thus $H_1 \cong \mathbb{Z}$.

This will be the fundamental fact used to investigate sets of data points when we introduce persistent homology, since our analysis will be constrained to the first homology group. We will be constructing simplicial complexes using these data points, connecting points that are close to each other; the appearance of of such non-boundary 1-dimensional cycles will be noted in the resulting analysis as “hole”- or circle-like structures. Let’s now turn to another familiar space to show the ability of homology groups to extend to higher dimensions.

**Example 4.10.10.** Under a $\Delta$-complex structure, the sphere $S^2$’s homology groups can be computed using three 0-simplices, three 1-simplices, and two 2-simplices, as shown in Figure 4.32.

![Figure 4.32: $\Delta$-complex visualization of $S^2$.](image)

As before, we compute the chain, cycle, boundary, and homology groups of $S^2$, noting their isomorphisms free Abelian groups on the integers.

The chain group $C_0$ is $\langle x, y, z \rangle$, the free Abelian group generated by the points $x, y, z$, and so $C_0 = Z_0 \cong \mathbb{Z}^3$.

To compute $B_0$, we need to look at $\partial_1$. Since we have three 1-simplices, let’s look at $\partial_1$’s output for a general element of $C_1$ and integers $\alpha, \beta, \gamma$.

$$\partial_1(\alpha a + \beta b + \gamma c) = \alpha(x - y) + \beta(z - x) + \gamma(y - z)$$

Thus $B_0 = \langle x - y, z - x, y - z \rangle$, but since $y - z = (-1)(x - y) + (-1)(z - x)$, this simplifies to $\langle x - y, z - x \rangle$, and so $B_0 \cong \mathbb{Z}^2$. Thus $H_0 \cong \mathbb{Z}$.  

74
Given our three 1-simplices $a, b, c$, we have $C_1 = \langle a, b, c \rangle$. To compute $Z_1$, we need to use our general form of $\partial_1$ again, setting the output equal to 0.

$$\partial_1(\alpha a + \beta b + \gamma c) = \alpha(x - y) + \beta(z - x) + \gamma(y - z)$$

$$= (\alpha - \beta)x + (\gamma - \alpha)y + (\beta - \gamma)z$$

Setting this last expression to 0, we get $\alpha = \beta = \gamma$, and so $Z_1 = \langle a + b + c \rangle$ and $Z_1 \cong \mathbb{Z}$.

For computing $B_1$, we turn to $\partial_2$. For an arbitrary element of $C_2$ generated from 2-simplices $A$ and $B$, we have

$$\partial_1(\alpha A + \beta B) = \alpha(a + b + c) + \beta(-a - b - c).$$

Since $A$ and $B$ share the edge connecting $y$ and $z$, note that we orient the edge $c$ such that $c$ has the opposite sign when computing $\partial(A)$ and $\partial(B)$: soon this will allow us to state $A + B$ as a cycle.

From the equation above, we conclude that $B_1 = \langle a + b + c \rangle$, and so $B_1 \cong \mathbb{Z}$. We conclude $H_1 \cong \langle 0 \rangle$, i.e. there are no nontrivial 1-dimensional cycles in $S^2$.

For computing $H_2$, we can note immediately that $B_2 = \langle 0 \rangle$, because $C_k = \langle 0 \rangle$ for $k \geq 3$. Additionally, we have

$$\partial_2(\alpha A + \beta B) = \alpha(a + b + c) + \beta(-a - b - c).$$

$$= (\alpha - \beta)(a + b + c),$$

and setting this expression equal to 0 yields $\alpha = \beta$. We conclude $H_2 = Z_2 = \langle A + B \rangle$, and so $H_2 \cong \mathbb{Z}$.

Next, we investigate the torus, which has two distinct non-trival cycles rather than just one, as in the case of the circle.

**Example 4.10.11.** Under a $\Delta$-complex structure, the torus’s homology groups can be computed using one 0-simplex, three 1-simplices, and two 2-simplices, as shown in Figure 4.33.
The chain group \( C_0 \) is \(< x >\), and so \( C_0 = \mathbb{Z}_0 \cong \mathbb{Z} \).

To compute \( B_0 \), we need to look at \( \partial_1 \). Since we have three 1-simplices, let’s look at \( \partial_1 \)’s output for a general element of \( C_1 \) and integers \( \alpha, \beta, \gamma \).

\[
\partial_1(\alpha a + \beta b + \gamma c) = \alpha(x - x) + \beta(x - x) + \gamma(x - x) = 0
\]

Thus \( B_0 = < 0 > \), and so \( H_0 = \mathbb{Z}_0 \cong \mathbb{Z} \).

Given our three 1-simplices \( a, b, c \), we have \( C_1 = < a, b, c > \). However, since we just determined \( \partial_1 = 0 \), we have \( Z_1 = C_1 = < a, b, c > \).

For computing \( B_1 \), we again turn to \( \partial_2 \). For an arbitrary element of \( C_2 \) generated from 2-simplices \( A \) and \( B \), we have

\[
\partial_1(\alpha A + \beta B) = \alpha(a + b + c) + \beta(-a - b - c).
\]

From the equation above, we conclude that \( B_1 = < a + b + c > \), and so \( B_1 \cong \mathbb{Z} \). We conclude

\[
H_1 = Z_1/B_1 = < a, b, c > / < a + b + c > = < a + b + c, b, c > / < a + b + c > \cong < b, c > \cong \mathbb{Z}^2.
\]

For computing \( H_2 \), we can note again that \( B_2 = < 0 > \), because \( C_k = < 0 > \) for \( k \geq 3 \). Again, since

\[
\partial_2(\alpha A + \beta B) = \alpha(a + b + c) + \beta(-a - b - c).
\]

and setting this expression equal to 0 yields \( \alpha = \beta \), we conclude \( H_2 = Z_2 = < A+B > \), and so \( H_2 \cong \mathbb{Z} \).

Finally, we compute the homology groups of the Klein bottle, the other manifold aside from the circle found in Carlsson and Gabrielsson’s work. Unlike the other three manifolds we have just observed, the Klein bottle’s first homology has a finite component. As we have pointed out, this makes persistent homology unable to uncover the Klein bottle’s homology entirely, as it is only able to detect copies of the infinite cyclic group \( \mathbb{Z} \) in a simplicial complex.

**Example 4.10.12.** Under a \( \Delta \)-complex structure, the Klein bottle’s homology groups can be computed using one 0-simplex, three 1-simplices, and two 2-simplices, similar to the torus, as shown in Figure 4.34.

As before, we compute the Klein bottle’s chain, cycle, boundary, and homology groups, noting their isomorphisms free Abelian groups on the integers.

The chain group \( C_0 \) is \(< x >\), and so \( C_0 = \mathbb{Z}_0 \cong \mathbb{Z} \).

To compute \( B_0 \), we need to look at \( \partial_1 \). Since we have three 1-simplices, let’s look at \( \partial_1 \)’s output for a general element of \( C_1 \) and integers \( \alpha, \beta, \gamma \).

\[
\partial_1(\alpha a + \beta b + \gamma c) = \alpha(x - x) + \beta(x - x) + \gamma(x - x) = 0
\]

Thus \( B_0 = < 0 > \), and so \( H_0 = \mathbb{Z}_0 \cong \mathbb{Z} \).

Given our three 1-simplices \( a, b, c \), we have \( C_1 = < a, b, c > \). However, since we just determined \( \partial_1 = 0 \), we have \( Z_1 = C_1 = < a, b, c > \).
Figure 4.34: ∆-complex visualization of the Klein bottle, arranged using the traditional method of identifying the edges of a square.

For computing $B_1$, we again turn to $\partial_2$. For an arbitrary element of $C_2$ generated from 2-simplices $A$ and $B$, we have

$$\partial_1(\alpha A + \beta B) = \alpha(a + b + c) + \beta(-a + b - c).$$

From the equation above, we conclude that $B_1 = \langle a + b + c, -a + b - c \rangle$. We conclude

$$H_1 = \mathbb{Z}_1 / B_1 = \langle a, b, c \rangle / \langle a + b + c, -a + b - c \rangle.$$

For computing $H_2$, we can note again that $B_2 = \langle 0 \rangle$, because $C_k = \langle 0 \rangle$ for $k \geq 3$. Since

$$\partial_2(\alpha A + \beta B) = \alpha(a + b + c) + \beta(-a + b - c),$$

$$= (\alpha - \beta)(a + c) + (\alpha + \beta)b,$$

, and setting this expression equal to 0 yields $\alpha = \beta = 0$, we conclude $H_2 = \mathbb{Z}_2 = \langle 0 \rangle$.

To connect our discussions of homotopy and homology, we introduce an often-cited result, representing an important connection between the first homology and fundamental group of a space. First we define some important algebraic term, followed by the result itself.

**Definition 4.10.13.** Given a group $G$, the *commutator subgroup* $[G, G]$ is the subgroup generated by elements of the form $yx^{-1}y^{-1}$, for all $x, y \in G$. The quotient group $G/[G, G]$ is then called the *abelianization* of $G$.

It can be shown that this commutator subgroup must be normal, and so this quotient is always applicable. Additionally, the term “abelianization” is apt, as this quotient group is in fact Abelian: note that, for any $x, y \in G$,
\[= xy(y^{-1}x^{-1}yx[G, G])\]
\[= yx[G, G]\]
\[= (y[G, G])(x[G, G]).\]

We now present the essential connection between perhaps the two important groups discussed in this chapter. Proving this property is beyond the scope of this text, and makes notable use of singular homology, as mentioned earlier in this chapter.

**Theorem 4.10.3.** For a path-connected space \(X\), the first homology \(H_1(X)\) is isomorphic to the abelianization of the fundamental group \(\pi_1(X)\).
Chapter 5

TDA Overview

We now formally introduce the two major tools we will use in our analysis of trained separable CNN kernels: persistent homology and the Mapper algorithm. We will be using these tools to point to changes in the underlying distribution of these CNN kernels, namely the appearance of non-boundary cycles. As we apply these techniques at different points during our CNNs’ training process, persistent homology will aim to detect the formation of such cycles, while the Mapper algorithm will allow us to investigate the space from a more qualitative perspective and allow us to locate what types of kernels are populating the nontrivial cycles that form. Our main reference text for building up definitions in this chapter is [10].

5.1 Persistent Homology with Point Clouds

Persistent homology is a technique applied to a sequence of spaces, increasing in size, such that each successive space contains the previous space in the sequence, i.e. a sequence $S_1 \ldots S_n$ where $S_{k+1} \subseteq S_k$ for each $k \in \{1 \ldots n\}$. The goal here is to observe the changes in homology that occur that persist in many successive spaces in this sequence, and, more specifically, to pinpoint a ”birth” and ”death” point for each non-trivial cycle in this sequence of spaces. There are many different sorts of such sequences that may be analyzed using persistent homology, such as growing sublevel sets for a function [2]. However, the specific case we will focus on here involves point clouds, or sets of points equipped with a metric. In particular, Carlsson/Gabrielsson’s work and our work both represent $3 \times 3$ CNN kernels as points in $\mathbb{R}^9$.

In persistent homology analysis on a point cloud, the special growing sequence of spaces we choose is an increasingly-connected filtration of simplicial complexes (as defined below) derived from the point cloud. A common method of defining such a filtration from the point cloud is called the Vietoris-Rips filtration, which we will define later. In the Vietoris-Rips filtration and related but less-computer-friendly (and thus less-used) Cech filtration, edges and further simplices will be added to the point cloud depending on the distance between each pair of points: close-enough pairs of points will be connected via edges and simplices.

**Definition 5.1.1.** Let $K$ be a simplicial complex, and for $\tau, \sigma \in K$, define the relation $\tau < \sigma$ if $\tau$ is a face of $\sigma$. Using this relation on $K$, let $f : K \to \mathbb{R}$ be a nondecreasing function, i.e. when $\tau$ is a face of $\sigma$ ($\tau < \sigma$), $f(\tau) \leq f(\sigma)$.

Enumerate the elements of $f(K)$ as $a_1 \ldots a_n$, and for $a \in \mathbb{R}$, let $K(a)$ be the sublevel set $f^{-1}([-\infty, a))$. Writing $K(a_i)$ as $K_i$, this admits a sequence of complexes

$$K_1 \subseteq \cdots \subseteq K_n = K.$$  

This admits a sequence of $p^{th}$ homology groups

$$H_p(K_1) \to H_p(K_2) \to \cdots \to H_p(K_n) = H_p(K)$$

connected by homomorphisms. In particular, when $i < j$, such a homomorphism takes the form $f_p^{i,j} : H_p(K_i) \to H_p(K_j)$, where each $f_p^{i,j}$ is induced by the inclusion map on $K_i \subseteq K_j$. Additionally,
for \( i < j < k \), define the composition \( f_p^{i,k} = f_p^{j,k} \circ f_p^{i,j} \). Each of these homomorphisms admits a new algebraic object, defined below.

**Definition 5.1.2.** The collection of images of the homomorphisms \( f_p^{i,j} \), notated \( H_p^{i,j} \), are the \( p^\text{th} \) **persistent homology groups** of the filtration \( K_1 \subseteq \cdots \subseteq K_n \).

Equivalently, \([34]\) states the \( p\)-persistent \( k^\text{th} \) homology group of \( K_i \) as

\[
H_p^{i,p} = Z_k^i / (B_k^{i+p} \cap Z_k^i).
\]

The first definition can be thought of as describing the birth and death points of a non-boundary cycle, while the second denotes the birth point and persistence of the same cycle. Along with these persistent homology groups comes a notion of rank specific to persistent homology.

**Definition 5.1.3.** The \( p^\text{th} \) persistent Betti number corresponding to \( H_p^{i,j} \) is \( \beta_p^{i,j} \), the rank of \( H_p^{i,j} \).

A central objective of using persistent homology on a given filtration of simplicial complexes is to identify important homological features that form and dissipate during the filtration process. This importance is determined by what is deemed the **persistence** of each homology class \( \gamma \), i.e. each non-trivial cycle, which represents, intuitively, the lifespan of \( \gamma \) in the filtration before it becomes the boundary of a \( p \)-chain in the simplicial complex \( K_j \).

**Definition 5.1.4.** Given a homology class \( \gamma \in H_p(K_i) \), we say \( \gamma \) is **born** if \( \gamma \notin H_p^{i-1,j} \), and we say \( \gamma \) **dies entering** \( H_p(K_j) \) if \( f_p^{i,j-1}(\gamma) \notin H_p^{i-1,j-1} \) but \( f_p^{i,j}(\gamma) \in H_p^{i-1,j} \). The **persistence** of a homology class \( \gamma \) is the difference \( a_j - a_i = f(K_j) - f(K_i) \).

This birth-death process is visualized for a homology class \( \gamma \) in Figure 5.1.

![Figure 5.1](image)

**Figure 5.1:** In this example, the homology class \( \gamma \) is born at \( H_p(K_i) \) and dies entering \( H_p(K_j) \). Diagram from [10].

The output of a persistence homology analysis procedure is a set of intervals \([a_i, a_j]\), each representing the birth and death of a nonzero homology class. This can be visualized via a **persistence barcode**, which shows the interval \([a_1, a_n]\), along with each interval \([a_i, a_j]\) overlaid in the form of a bar, or via a **persistence diagram**, which displays a scatterplot of points of the form \((a_i, a_j)\). Thus, in a barcode, bars of significant length represent highly persistent features, whereas in persistence diagrams, highly persistent features correspond to points far above the diagonal.

The left half of Figure 5.2 shows a pair of persistence barcodes is from Carlsson and Gabrielsson’s conclusions regarding structures found in collections of \( 3 \times 3 \) kernels taken from multiple fully trained CNNs. Note that in the dimension-0 barcode, a single bar extending the full length of observed \( d \) values is inevitable, as there always exists at least one connected component in the complex, regardless of the
On the other hand, particularly significant to Carlsson and Gabrielsson’s findings is the appearance of three significant holes in the point cloud of CNN filters. These three holes show up as three bars of significant length (compared to the others) in the dimension-1 barcode. As we will see, these three holes can be visualized using the Mapper algorithm, detailed in the following section, and the CNN filters existing at their borders constitute what Carlsson terms the primary circle and two secondary circles.

The right side of Figure 5.2 shows a persistence diagram from our work, which shows a single clear 1-dimensional homology class in its corresponding point cloud. The point cloud for this diagram was generated from a collection of trained separable CNNs, which, as we will describe later, suggests a close connection between the topological information learned by separable and standard CNNs.

![Figure 5.2](image)

Figure 5.2: (left) persistence barcodes of dimension 0 and 1. Note the three longer bars in the dimension-1 barcodes, corresponding to three distinct 1-dimensional holes. (right) a persistence diagram of dimension 0 (red) and 1 (green). There is one significant 1-dimensional homology class, given the single point seen far above the diagonal.

Finally, let’s introduce some specific examples of how we may compute persistent homology given a relatively simple simplicial complex, and how the resulting persistence intervals can change given different choices of filtration on the same simplicial complex $K$. This will make use of a matrix reduction algorithm over the field $\mathbb{Z}_2$; this algorithm is detailed and motivated in [34], and will be explained as we walk through the first example. A great resource for walking through these computations in a dynamic classroom environment is Bei Wang Phillips’s Computational Topology course at the University of Utah\(^1\).

**Example 5.1.5.** Let’s consider the simplicial complex $K$ generated by the boundary of a 2-simplex, together with an ordering of its simplices for defining our chosen filtration, as visualized below.

\(^1\)https://www.youtube.com/playlist?list=PLDZ6LA165DbIvbmgCjcCuTA7nttfXjie
We can describe the points and connecting edges in a matrix as follows, representing each simplex labeled 1-6 in the figure above with a column:

\[
\begin{pmatrix}
0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
\end{pmatrix}
\]

Our matrix is \(n \times n\), where \(n\) is the number of simplices in our final complex of the filtration, i.e. the number of steps in the filtration. Note that we have each 0-simplex to a zero column in the matrix, and for each 1-simplex-associated column, we have placed a 1 in each entry associated to a corresponding 0-simplex. For example, column 4 is associated to edge 4 in the simplicial complex, which connects the vertices labeled 1 and 2, so the first and second entries of the fourth column are ones. This process will generalize to higher order simplices, as we will place ones in the matrix entries associated to the boundary of each simplex.

We then carry out a step-by-step process to reduce this matrix to a special form, which encodes our persistence intervals:

1. Check each column \(j\) from left to right in the matrix.
2. If column \(j\) is a zero column, do nothing and continue to the next column.
3. Otherwise, let \(i\) denote the lowest one entry in column \(j\). If all entries in row \(i\), columns 1, 2, \ldots \(j-1\) are zero, do nothing and continue to the next column.
4. Otherwise, replace column \(j\) with a linear combination of columns 1, 2, \ldots \(j\) over \(\mathbb{Z}_2\) such that the lowest one entry in column \(j\) has no ones directly to the left of it.

Applied to the matrix above, this process results in the matrix

\[
\begin{pmatrix}
0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
\end{pmatrix}
\]

as column 6 is added to both columns 4 and 5.

To convert this matrix representation into a set of persistence intervals, we first investigate the nonzero columns, looking for the lowest one in each. The coordinates of this lowest one will correspond to the endpoints of a persistence interval: e.g. the ones in this matrix correspond to 0-dimensional features with persistence intervals \([2, 4)\) and \([3, 5)\). We know these are 0-dimensional cycles because their
death points correspond to the introduction of 1-simplices.

Additionally, each empty column corresponds to the birth of a nonzero homology class; we have accounted for vertices 2 and 3 and edges 4 and 5, but we see that there are no lowest-entry ones in rows 1 or 6 and no lowest-entry ones in columns 1 or 6. Thus we have nonzero homology classes born with vertex 1 and edge 6 that persist indefinitely in this filtration; i.e. their persistence intervals are $[1, \infty)$ and $[6, \infty)$. Since simplex 1 is a vertex and simplex 6 is an edge, these features are a 0-dimensional cycle and a 1-dimensional cycle, respectively.

The following example will demonstrate how 2-simplices fit into the picture when observing the 1-dimensional persistent homology of a filtration.

**Example 5.1.6.** Consider the following pair of 2-simplices with nonempty intersection.

![Figure 5.4: Two 2-simplices sharing an edge.](image)

The matrix corresponding to this filtration is

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

Note that here, the 2-simplices 10 and 11 are encoded in the matrix by their boundary edges 5,7,8 and 6,8,9 respectively. Carrying out the same reduction operation from the previous example results in the matrix

$$
\begin{bmatrix}
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
$$
implying that we have nonzero homology classes with the following persistence intervals:

- dimension 0: \([1, \infty), [2, 5), [3, 6), \text{ and } [4, 7)\)
- dimension 1: \([8, 10)\) and \([9, 11)\).

**Example 5.1.7.** Consider the simplicial complex from 5.1.6 with a different filtration.

![Figure 5.5: Two 2-simplices sharing an edge, as in Figure 5.4 with a relabeling of the edges.](image)

The matrix corresponding to this filtration is

\[
\begin{pmatrix}
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix},
\]

Note that here, the 2-simplices 10 and 11 are encoded in the matrix by their boundary edges 5,7,8 and 6,8,9 respectively. Carrying out the same reduction operation from the previous example results in the matrix
implying that we have nonzero homology classes with the following persistence intervals:

- **dimension 0**: 
  
  - \([1, \infty)\)
  
  - \([2, 5)\)
  
  - \([3, 8)\)
  
  - \([4, 6)\)

- **dimension 1**: 
  
  - \([7, 10)\)
  
  - \([9, 11)\)

Note the difference in the persistence of each nonzero homology class in this filtration when compared to 5.1.6, despite them being derived from the same simplicial complex.

### 5.1.1 Example Filtrations: Cech and Vietoris-Rips

Let’s now walk through the construction of a given complex from a point cloud when using the Cech and Vietoris-Rips filtrations. Define \(d \in [0, M/2]\), where \(M = \max_{i,j} ||x_i - x_j||\), the maximum distance between any two points \(x_i\) and \(x_j\) in the cloud, given whichever distance function we choose. We can then draw a ball of radius \(d\) around each point in \(X\). By first setting \(d = 0\) then increasing \(d\), we can start from simply the set of points itself, then gradually grow each ball by a uniform amount until \(d\) reaches \(M/2\), which guarantees that for every pair of points \(x_i, x_j \in X\), their corresponding balls \(B(x_i, d)\) and \(B(x_j, d)\) have a nonempty intersection.

For values of \(d\) strictly between 0 and \(M/2\), we observe the intersection \(B(x_i, d) \cap B(x_j, d)\) for each pair of points \(x_i, x_j \in X\). If this intersection is nonempty, we connect \(x_i\) and \(x_j\). If the intersection is empty, we leave them disconnected. This is the process we use for building the edges of a simplicial complex on the point cloud \(X\), and the final simplicial complex will be the space we use in the sequence for a chosen value of \(d\).

We will finish the construction of our simplicial complex by introducing a rule for dealing with cycles formed by these connecting edges between points. The main question we have once one of these cycles is formed is “Should we fill this cycle in to form a simplex or leave it as an open (non-trivial) cycle?” Two answers to this question are provided respectively by the Cech and Vietoris-Rips complex constructions. In the Cech construction, we only form a simplex if, for points \(x_1 \ldots x_k\), the intersection between their corresponding balls \(\cap_{i=1}^k B(x_i, d)\) is nonempty. However, in practice, this can be difficult to compute for large high-dimensional datasets, so often the simpler-to-compute Vietoris-Rips complex is used instead. For the Vietoris-Rips construction, for points \(x_1 \ldots x_k\), a simplex is filled in when \(||x_i - x_j|| \leq 2d\) for all pairs of points in \(\{x_1 \ldots x_k\}\): i.e. the intersection of each pair of balls in \(\{B(x_1, d) \ldots B(x_k, d)\}\) is nonempty.

Figure 5.6 shows an example of one such Vietoris-Rips complex on a point cloud for a particular ball radius \(d\). There are a few important features to note about this complex. If you look closely at the single filled-in triangle at the top of the complex, you will see a gap in the balls surrounding its three corner points. This shows that we are not looking at a Cech complex, otherwise, the triangle would not be filled in as a simplex, and instead, we would merely see three connected line segments with a hollow middle. In contrast, the tetrahedron in the lower left of the image would still be filled in if this were a Cech complex. This is because its corresponding four balls, collectively, have a common region of intersection.

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Figure 5.6: Vietoris-Rips complex with a ball of radius $d$ centered at each point, from [31].

Now that we have a simplicial complex constructed for each value of $d$ in $[0, M/2]$, we now check each complex for 0- and 1-dimensional homology. We can think of 0-dimensional homology as counting connected components and 1-dimensional homology as counting holes in the complex, i.e. cycles not filled in to complete a simplex. In some cases (particularly in lower dimensions, where such calculations are feasible with existing technology) we may check for 2-dimensional homology as well, which counts hollow components that can "hold volume", e.g. a torus. However, we will primarily be focusing on 1-dimensional homology in this document, as this is the perspective most relevant to Carlsson’s conclusions and our work.

Thus, for $d = 0$, each point is a component of the space’s 0-dimensional homology, and we find nothing when looking at the space’s 1-dimensional homology. As we increase $d$, growing the size of the ball around each point, we may start to find Vietoris-Rips complexes with holes. When applying persistent homology, we consider each hole we find in our sequence of complexes, then find the interval of $d$ values across which this hole remains present, or persists. The resulting intervals for both 0- and 1-dimensional homology are the intervals used when visualizing the output of the analysis via persistence barcode or persistence diagram.

The Vietoris-Rips structure, representing a single step in its corresponding Vietoris-Rips filtration in persistent homology, can alternately be described as a special type of complex derived from a graph induced by the point cloud, as defined in [3].

**Definition 5.1.8.** A $k$-clique of an undirected graph $G = (V, E)$ is a complete subgraph with $k$ vertices. The clique complex of $G$ is the simplicial complex $\text{Cl}(G)$ with $G$’s vertex set $V$ as its set of 0-simplices and a $k$-simplex corresponding precisely to each $k$-clique.

When viewed through the lens of clique complexes, setting up a Vietoris-Rips complex requires us to, starting with our point cloud $X$ with distance $d$ between each pair of points in $X$, construct a weighted graph $G = (V, E)$ with $X$’s points as its vertex set $V$. $G$’s edge set $E$ will then be weighted, and each weight will be set to the value of the distance between each edge’s endpoints.

We then define $G_\delta = (V_\delta, E_\delta)$ as the subgraph of $G$ with $V_\delta = V$ and $E_\delta \subseteq E$ where $E_\delta$ includes only edges with weights less than or equal to $\delta$. Given these initial steps, the Vietoris-Rips complex with distance threshold $\delta$ can be defined as the clique complex $\text{Cl}(G_\delta)$.

### 5.2 Mapper Algorithm

The Mapper algorithm is another tool used to interpret a set of points from a topological perspective, developed by Gurjeet Singh, Facundo Mémoli, and Gunnar Carlsson and described in [29]. Like persistent homology, it uses simplicial complexes to help make inferences about the arrangement of data points in a point cloud $X$. However, the way that it uses simplicial complexes is completely distinct from
that of persistent homology, as its main goal is qualitative visualization rather than direct computation. An example of its output is shown in Figure 5.9.

The first major difference to point out between these two methods of analysis is the format of the output. Whereas persistent homology outputs a barcode or persistence diagram representing the "life" of each 0-, 1-, or 2-dimensional cycle in the dataset in a sequence of consecutive simplicial complexes, the Mapper algorithm’s output is itself a single simplicial complex: in particular, the example displayed in Figure 5.9 is a graph. This single picture attempts to recreate the "shape" of the given point cloud \( X \) in a more easily visualizable space. Often, as in our study of CNNs, this amounts to projecting points in a high-dimensional space to a lower-dimensional space, often \( \mathbb{R}^2 \).

We start with some background details and definitions.

**Definition 5.2.1.** Let \( X \) be any set and \( A \) be an indexing set. A cover \( U \) of \( X \) is a collection of sets \( U_\alpha \), with \( \alpha \in A \), such that \( X \subseteq \bigcup_{\alpha \in A} U_\alpha \).

Note that in certain topological contexts, such as when discussing compactness of spaces, we may insist that \( X = \bigcup_{\alpha \in A} U_\alpha \), but it is to our advantage to remain more general in this definition when describing the Mapper algorithm.

Additionally, we need to define an operation on such a covering that results in a summary simplicial complex.

**Definition 5.2.2.** Let \( X \) be a set with covering \( U \). The nerve of \( U \) \( N(U) \) is an abstract simplicial complex with vertex set \( A \) with \( k \)-simplices spanned by vertex sets \( \{\alpha_0, \alpha_1 \ldots \alpha_k\} \) precisely when \( U_{\alpha_0} \cap U_{\alpha_1} \cap \ldots \cap U_{\alpha_k} \) is nonempty.

For clarification, an abstract simplicial complex is a generalization of a simplicial complex, defined below as quoted from [5].

**Definition 5.2.3.** "An abstract simplicial complex is a pair \((V, \Sigma)\), where \( V \) is a finite (vertex) set, and \( \Sigma \) is a family of non-empty subsets of \( V \) such that \( \sigma \in \Sigma \) and \( \tau \subseteq \sigma \) implies that \( \tau \in \Sigma \).

This construction will provide some useful geometric information about the original space, and a key insight into the precision of this information is given by what is known as the nerve theorem, stated here without proof. As a warning, this theorem makes use of a numerability criterion for a covering, which is beyond the scope of our work but is defined in [26].

**Theorem 5.2.1.** Given a space \( X \) with covering \( U \), suppose that all elements of \( U \) are open sets of \( X \), and \( U \) is numerable. Additionally, for the full vertex set \( A \) generating the nerve \( N(U) \), suppose that for all nonempty \( S \subseteq A \), \( \bigcap_{\sigma \in S} U_\sigma \) is either contractible or empty. Then \( N(U) \) is homotopy equivalent to \( X \).

**Example 5.2.4.** In [5], Carlsson introduces an example of such a covering that can be used to produce a simplicial complex approximating the shape of the unit circle \( S^1 \), visualized in Figure 5.7.

In particular, for \( x, y \in \mathbb{R} \), \( A = \{(x, y)|y < 0\} \), \( B = \{(x, y)|y > 0\} \), and \( C = \{(x, y)|y \notin \{-1, 1\}\} \). Carlsson presents two methods for representing \( S^1 \) as a graph given this cover, the results of which are shown below in Figure 5.8.

The graph on the right in Figure 5.8 is the result of including a vertex for each element of the covering (i.e. \( A, B, \) or \( C \)), then connecting vertices whose sets have a nonempty intersection. In other words, this graph is the nerve of \( \{A, B, C\} \).

The graph on the left is also the result of taking a nerve of a covering, however it is clear from the image that this covering provides more sensitive information about \( S^1 \). In particular, this covering
Figure 5.7: Example of a cover of the unit circle $S^1$ from [5]

Figure 5.8: Two graphs generated by the cover $\{A, B, C\}$ of $S^1$, from [5]

is $\{A, B, C_1, C_2\}$, where $C_1$ and $C_2$ are the two path components of $C$. This provides us with more information from which to build the resulting graph, and it turns out that this graph is homeomorphic to $S^1$.

A very closely related object to such approximating simplicial complexes is the Reeb graph/space, defined below. While the Reeb graph is derived from a generic topological space, we will see that the Mapper algorithm carries out a very similar process, taking a point cloud as input. In particular, when constructing Mapper complexes from point clouds, we often look to approximate a Reeb space, i.e. a simplification of the underlying topological space.

Definition 5.2.5. For a topological space $X$, let $f : X \to \mathbb{R}$ be a continuous map, and let $\sim$ be the equivalence relation on $X$ such that $x \sim y$ if $f(x) = f(y)$ and there exists a path in the subspace $f^{-1}(f(x))$ connecting $x$ and $y$. Then the Reeb graph $\mathcal{R}(X, f)$ is the quotient space $X/\sim$. If we let $f$ map more $\mathbb{R}^n$, the result is the more general Reeb space.

In accordance with the above principles, let’s now expand on the process of constructing a Mapper diagram.

We start by projecting our point cloud into the lower-dimensional target space $Z$ described above using a filter function $f : X \to Z$. For this discussion, we can let $Z$ be $\mathbb{R}^2$, since this is a common choice of target space for Mapper analysis; in particular, it is the target space chosen for our analysis of CNN kernels. Additionally, it is common to let $f$ be the function returning the first two principal components of a point from $X$.

From here, we divide the target space $Z$ into a collection of overlapping sets, i.e. establish a covering $\{Z_1 \ldots Z_k\}$ consisting of subsets of $Z$ such that each $Z_i$ has a nonempty intersection with at least one other set $Z_j$. Mapper implementations generally accept a number of cubes argument to control the number of open sets, as well as an overlap percentage argument that allows us to control the extent to which these sets overlap.

Once we have these open subsets $Z_1 \ldots Z_k$ constructed, we take the pre-image $f^{-1}(Z_i)$ of each subset: this returns the points of $X$ mapped to each subset through $f$. Since $f$ is a map from the entire point cloud to $Z$, these pre-images now divide our original point cloud $X$ into overlapping bins: note that these bins will likely not be mutually disjoint, as the subsets $Z_1 \ldots Z_k$ of $Z$ are overlapping.
We now consider each of these bins of $X$ separately and apply any clustering algorithm of choice to each bin. Each of the resulting clusters will be represented by a vertex in the final Mapper graph output, roughly corresponding to path components of a topological space derived from $X$. Finally, it remains to decide when to connect two vertices in the Mapper output. The Mapper algorithm specifies that two vertices will be connected by an edge if and only if their corresponding clusters in the point cloud have a nonempty intersection. This is why we establish $Z_1 \ldots Z_k$ as overlapping sets; if they were distinct, then there would be no edges in the Mapper output under this criterion. The effect of forming such vertices and connections is to take the nerve of our chosen covering of the point cloud.

Figure 5.9 displays a Mapper diagram generated using the Kepler Mapper library in Python. When applying this library, there is some room for customizing certain aspects of the point cloud analysis, namely the clustering method and the type of cover selected. The clustering method used is the DBSCAN algorithm, whose major parameters to be chosen are epsilon, which gives a maximum pairwise distance between data points for inclusion in a single cluster (Mapper node), and min_samples, which gives the minimum number of data points for the creation of a node in the Mapper diagram. When selecting the type of cover to use, i.e. when dividing the space of possible data points, there are two additional parameters:

- the percent overlap between clusters, and
- the number of cubes (overlapping sets) to use,

and these come standard with Kepler Mapper, the package we used for our Mapper analysis.

In particular, this Mapper diagram represents a set of $3 \times 3$ kernels taken from several hundred neural networks trained using the Keras library, using a methodology similar to what Carlsson and Gabrielson used in their research. Although the image below shows what appears to be a figure-eight shape, it does not truly intersect itself, i.e. the object is functionally a singular loop rather than two separate loops.

Figure 5.9: an example Mapper graph.
Chapter 6

Applying TDA Tools to a Trained CNN, Prior Related Research

Understanding the training process of CNNs and what types of patterns are reflected in their weights is a topic of great interest in deep learning research. Direct inspection of the kernels in trained CNNs and application of traditional statistical techniques have yielded some interesting qualitative results. However, it is clear that there is more information to be gleaned from applying other techniques, possibly techniques from other disciplines. This is because the weight data produced from a trained neural network is high-dimensional, and the fact that a change in weights in an early layer of a CNN cascades through the rest of the network.

Traditional dimension-reduction techniques, such as using principal components, have proven to lead to significant information loss. On the other hand, the TDA methods described in the previous chapter have proven to provide a more conclusive interpretation of the general progression of weights in a CNN both during and post-training when using classic CNN research datasets MNIST and CIFAR-10. In this section, we describe some major pieces of prior research that have helped form the basis for the analysis of both image-patch distributions and image-classification CNNs using topological methods.

We first refer to Lee, Mumford, and Pedersen [23], who explored a natural image dataset to mine information specifically pertaining to observed collections of $3 \times 3$-pixel image patches. Their work is a major source we point to as inspiration for the practice of flattening such image patches to vectors in $\mathbb{R}^9$, a fundamental step in the analysis we conducted on $3 \times 3$ CNN kernels. [23]'s findings for $\mathbb{R}^9$ points generated by natural image patches suggested an interesting underlying distribution related to what they called a "highly nonlinear" surface, in particular, a 2-dimensional manifold.

Carlsson [7] later studied $3 \times 3$ patches in same dataset previously explored by Lee, Mumford, and Pedersen, this time approaching the resulting $\mathbb{R}^9$ vectors from the standpoint of TDA. As we will describe in this chapter, Carlsson’s application of persistent homology and the Mapper algorithm (developed in part by Carlsson himself!) resulted in findings that pointed to a circular subspace of edge and corner patches (which he deemed the primary circle, as well as two other nontrivial circles intersecting this primary circle, termed the secondary circles. The particular intersections of these three circles led Carlsson to conclude that the 2-dimensional manifold underlying these patches was a Klein bottle.

From these findings, Carlsson and Gabrielsson, in [6] and [12], used these techniques to explore the space of $3 \times 3$ kernels generated by training a collection of image-classification CNNs. These CNNs were trained on the MNIST and CIFAR-10 datasets described in previous chapters. Note that in comparisons between our work and Carlsson and Gabrielsson’s, we can most directly point to their MNIST-related findings, as in both cases, the trained CNNs contained prominent patterns specifically related to the primary circle. However, also in both cases, the CNN kernels bared no significant resemblance to Carlsson’s secondary circles. Carlsson and Gabrielsson further found more complex CNN-kernel patterns in their CIFAR-10 analysis, some of which were more closely related to the secondary circles.

Our work contributes further to this exploration by applying these persistent homology and Mapper
techniques in additional CNN-related contexts. Most notably, we investigated CNNs constructed primarily using \textit{depthwise separable convolution} layers, as detailed in Chapter 3. Our application of these TDA algorithms focused on the first of the two major steps in each separable layer, i.e. the \textit{depthwise convolution}, and yields findings that point to a kernel space with a nontrivial first homology group. Specifically, the major cycle we encountered in the depthwise convolutions of all four layers of our CNNs was precisely the edge- and corner-kernel primary circle observed in Carlsson’s image-patch work and Carlsson and Gabrielsson’s CNN work. Additionally, the separable CNNs we trained were on a different set of data from the MNIST and CIFAR-10 datasets seen in the work introduced above and detailed later in this chapter: a collection of chest X-rays labeled with a 1 or 0 depending on whether the patient was suffering from pneumonia.

Our findings provide evidence for the ability of separable CNN layers to learn similar topological information to standard CNN layers with significantly fewer trained parameters, and that, when decoupling the convolution process into depthwise and cross-channel steps, this topological learning occurs specifically in the depthwise step. Our 5-layer CNN architecture, steps for separating the parameters from their kernels and flattening into $\mathbb{R}^9$ points, and results layer-by-layer are all detailed in the subsequent methods and results chapters.

### 6.1 Topology and Natural Image Patch Distributions

We can trace the use of topology with image data to an earlier work by Lee, Mumford, and Pedersen [23], which investigates the distribution of high-contrast $3 \times 3$-pixel patches within a natural image dataset. They found that high-density collections of these patches pointed to an interesting nonlinear geometry: in particular, their findings indicated a structure that “greatly reduce[d] the dimensionality of the problem.”

Lee, Mumford, and Pedersen represented $3 \times 3$ image patches from their natural image dataset as points in $\mathbb{R}^9$ and standardized them to points on the high-dimensional sphere $S^7$ by mean-centering them, then dividing each point by its Euclidean 2-norm. This is a practice that will be repeated both in Carlsson and Gabrielsson’s and our work regarding trained CNN kernels and applying earlier-defined TDA techniques. In Lee, Mumford, and Pedersen’s work, it was observed that viewing high-density sets of patches yielded a “$C^1$ 2-dimensional manifold, $M^2$, embedded in the 7-dimensional sphere.”

Mumford and Carlsson applied persistent homology, as described previously, to this specific dataset in [7], resulting in the discovery of some particularly interesting circular patterns. In particular, observing 1-dimensional persistent homology yielded multiple significant 1-dimensional homology classes, which implied the existence of important distinct 1-dimensional cycles in the dataset.

When viewed as normalized points on $S^7$, Mumford and Carlsson fit these high-variance $3 \times 3$ patches to what they termed a three-circle space, as pictured in Figure 6.1.

As a brief note, besides normalization to $S^7$, one other crucial step was taken when pre-processing the points: \textit{density filtration}, not to be confused with the filtration used in persistent homology. This is a process that aims to reduce the size of the dataset to only the points in the dataset’s densest regions. Perhaps more importantly, it aims to reduce the noise inherent in large datasets, as the addition a single unusual “noisy” point can result in wildly different outputs in persistent homology analysis. In particular, we choose a proportion $p$ of our data points deemed to have the largest density value: in later works, Carlsson and Gabrielsson refer to this subset of points to keep as a “core subset.” [12]

Mumford and Carlsson’s method for determining this density measure was to start with some $k \in \mathbb{N}$, as well as the proportion $p$ of points to keep, giving his implementation of density filtration two parameters. Carlsson continued to use this same strategy when working with CNNs, and we use this same process when applying density filtrations to our own points.

As usual, the standard Euclidean distance is used as a metric for the point cloud, and is then used to derive a density estimate $D(x_i)$ for each point $x_i$. $D(x_i)$ is then the reciprocal of the $k^{\text{th}}$ nearest
neighbor’s distance from $x_i$; i.e., a large distance from $x_i$ to its $k^{th}$ nearest neighbor suggests that the point cloud is not very dense near $x_i$. We then delete all points not in the top 100p% for density in $X$, the remaining 100p% constitutes the desired core subset.

This three-circle space was supported by three distinct significant 1-dimensional homology classes when viewing the data with a density filtration with $k = 100, p = .1$. There was also a single significant 1-dimensional nontrivial homology class observed when using a density filtration with $k = 300, p = .3$, suggesting a single, more fundamental 1-dimensional cycle occurring within the data.

Figure 6.2: Natural Image data as viewed in [7] using $k = 100, p = .1$.

Figure 6.3: Natural Image data as viewed in [7] using $k = 300, p = .3$.

This concept has been described multiple times throughout Carlsson’s related work through the idea
of Primary and Secondary Circles, using the diagram pictured in Figure 6.4.

![Figure 6.4: the primary and secondary circles of $3 \times 3$ image patches, from [6].](image)

Both of these secondary circles intersect the primary circle at two points, i.e. they include some particular edge- and corner-like patterns. However, the intersection is only at these two specific points: the rest of the patches included in the secondary circles consist of more complex patterning, sometimes involving the center pixel of the 3x3 patch.

Further analysis of these three circles showed that it was possible to map them onto an important structure defined earlier: the Klein bottle, as described in previous sections. Recall that this is an example of a 2-dimensional manifold, a structure that is "locally 2-dimensional." In colloquial terms, if you were to stand at any point on a 2-dimensional manifold, you would be able to walk around on it as if it were a 2-dimensional flat plane. Classic examples include the sphere, the torus, and the Mobius strip. The Klein bottle is a special type of 2-dimensional manifold that, in its realized form, is impossible for us to visualize, as it can only exist in a space of dimension 4 or higher.

The three circles found in the natural image patches were shown to exhibit this exact edge-identification behavior, as shown in Figure 6.5. The primary circle is in red, the secondary circles in green and black, and the identified edges with blue.

![Figure 6.5: arrangement of image-patch primary and secondary circles using identified points on a rectangle in the form of a Klein bottle, from [7].](image)
6.2 Carlsson and Gabrielsson’s Methods and Findings

In their neural network research, Carlsson and Gabrielsson applied not only persistent homology, but also the Mapper algorithm to kernels in the convolutional layers of a simple CNN trained on two image datasets: MNIST and CIFAR-10. In this context, the Mapper technique is important for visual verification of the shapes observed in the data, and gave Carlsson and Gabrielsson a quick way to check and visualize exactly which kernels appear in the observed primary and secondary circles.

Their networks used $3 \times 3$ kernels in their two convolutional layers, and the parameters in each of these kernels were flattened into 9-dimensional points (points in $\mathbb{R}^9$), similar to the image patches used in [7] and [23]. Carlsson and Gabrielsson also used similar procedure to [7] involving normalization to $S^7$ and density filtration.

After training multiple networks with separate random initializations, the first layer’s kernels were collected from each network and arranged into a single point cloud, and the same was done for the second layer. The architecture of Carlsson and Gabrielsson’s network was as shown in Figure 6.6.

![Figure 6.6: Carlsson and Gabrielsson’s CNN Architecture, from [12].](image)

The focus, when applying TDA, was on the convolutional layers, as this is where the interpretation of network parameter sets as points in $\mathbb{R}^9$ was readily apparent. For this discussion’s purposes, the following dense layers can be ignored, but we have left them here to explain how the network finishes classifying images past its convolutional layers.

When analyzing the convolutional kernels of the trained networks, the most conclusive results were found in the first layer. This seems to be an intuitive result, as the first layer’s convolutional kernels are applied directly to the intensity values of the images themselves.

CNNs trained on the MNIST dataset were of the form $M(X, Y, Z)$ from Figure 6.6. This dataset consists of grayscale hand-drawn digits, 0 through 9, so these networks finish with softmax activation with 10 inputs and outputs. Note that since these are grayscale images, they have one color channel, and so first-layer kernels have a depth dimension of only 1, which means that flattening them to 9-dimensional points is a relatively simple process.

Figure 6.7 shows a clear cyclic shape formed by the kernels of these 100 CNNs, which were each trained for 40,000 gradient descent iterations. Investigating the nodes around this circle showed that this geometry corresponded to edge- and corner- type patterns in the CNNs’ kernels. This single-cycle structure is corroborated by the persistence barcode in Figure 6.8.

The $3 \times 3$ patches shown in Figure 6.7 were calculated by observing regions of nodes in the Mapper, then computing the mean of the points in $\mathbb{R}^9$ contained in each region of nodes: recall that individual nodes in a Mapper algorithm typically correspond to many data points.
Given the result of a single Mapper diagram on the MNIST dataset being a single cyclical structure, this first layer can be seen, topologically, as a simple ”default” CNN test case. As we will see later, a similar result is seen when observing CNNs trained on our grayscale X-Ray dataset, even when using the separable CNN format.

However, interestingly, a similar easily-definable pattern did not appear when investigating the kernels from the second layer of these networks, as can be seen in Figure 6.9. As will be discussed later with the color CIFAR-10 networks, the $\mathbb{R}^9$ points used here were derived by separating each kernel on a channel-by-channel basis. Specifically, since layer 2 has depth dimension $X$ (with $X = 64$ in [6]), each kernel in layer 2 will produce 64 points in $\mathbb{R}^9$ when constructing this Mapper diagram.

CNNs trained on the CIFAR-10 dataset were of the form $C(X, Y, Z)$ from Figure 6.6. This is another 10-class image dataset, but unlike the MNIST dataset, each image is in RGB color, and so each has three channels to investigate in the first convolutional layer of each CNN.

This required Carlsson and Gabrielsson to take additional steps when breaking down each kernel to find appropriate $\mathbb{R}^9$ points. Their strategy was to break each kernel down channel-by-channel: for example, from a single 3-channel kernel, Carlsson and Gabrielsson obtained 3 $\mathbb{R}^9$ points, one for each channel.

Carlsson and Gabrielsson additionally observed two different versions of the CIFAR-10 dataset: one in full color (original dataset) and one in grayscale (using the formula

\[
\text{intensity} = .2989R + .5870G + .1140B
\]
to represent the intensity value of each single-channel grayscale pixel in the new altered dataset).
Interestingly, when looking at fully-trained CNNs on grayscale CIFAR-10 images (70,000 batch iterations, 77% accuracy), creating a Mapper of the first layers’ kernels did not generate a clear primary circle, but did show five notable clusters of kernel patterns (Figure 6.10).

Even potentially more interesting: a primary circle pattern showed up more clearly in the second layer of these networks. Note that the dimension-1 homology associated with the second-layer kernels of these networks agrees with this notion of a single prominent circle. Additionally, there was another cluster in these filters that corresponded to a "bullseye" pattern, shown in the center of the Mapper (Figure 6.11).

Carlsson and Gabrielsson also created Mapper diagrams for mid-training snapshots of the CNNs they trained on this grayscale CIFAR-10 dataset. Separating the training into stages based on the current number of batch iterations (number of applications of gradient descent), they constructed a picture of how their CNNs’ kernels changed during the training process, for both the first and the second convolutional layers.

In Figure 6.12, more concentrated areas of kernels (nodes representing more kernels) are shown in red, and less concentrated areas in blue. We see that, as expected, a primary circle does form in the kernels, with four particularly concentrated areas, particularly in the 200-batch-iteration image.

However, as training continues, it seems that this singular edge- and corner- detecting circle is not sufficient for optimizing the network. At 500 iterations, we see the circle break more clearly into these
Carlsson and Gabrielsson note two additional observations about the final two training stages in Figure 6.12.

1. We can still see the four concentrated edge clusters as far as 2000 batch iterations into training, so these are clearly key to training a CNN on this dataset.

2. A circle pattern starts to emerge in the second layer as it wanes in the first layer, possibly suggesting that this second layer is, in a sense, compensating for the loss of the circle pattern in the first layer.

Carlsson and Gabrielsson’s study of CNNs also extended to those trained on the original color CIFAR-10 dataset. Thus, in these networks, the first and second layers contained many more parameters than the grayscale-trained networks. They still opted to represent kernels using $3 \times 3$ patches, but did so by dividing each $3 \times 3 \times 3$ filter in the first layer into $3 \times 3$, separating by color/depth channel.

As we will see, training on the color dataset is where they started to obtain results more specifically consistent with the three-circle model previously proposed. When taking a restrictive ($p = .14$) core subset of kernels in the first layer for these CNNs, a very clear single-circle pattern was uncovered through the point cloud’s Mapper diagram (Figure 6.13), very similar to that of the MNIST-trained networks mentioned earlier.

For the color CIFAR-10 dataset, this circle continued to show up when Carlsson and Gabrielsson allowed more points from the cloud ($p = .32$) into the Mapper analysis (Figure 6.14).

However, another intriguing pattern started to reveal itself here: two additional circles appeared, a fact which is most readily apparent in the persistent homology barcode. The barcode very clearly shows...
three long intervals when considering 1-dimensional homology. This finding falls in line with the secondary circles, which, in conjunction to the primary circle, are consistent with the Klein bottle structure proposed in earlier analyses of $3 \times 3$ natural image patches.

For these color-image-trained networks, Carlsson and Gabrielsson observed networks in the earlier training stages and found a similar structure to the corresponding Mapper diagrams for the first and second layers, as shown in Figure 6.15. The main contrast between the early learning process of these networks and the grayscale-image-trained networks is the seeming reappearance of the edge-detecting primary circle in the first layer at 1500 batch iterations.
In sum, the color-trained CIFAR-10 CNNs were most promising in demonstrating the appearance of primary and secondary circles in the fully-trained stage (100,000 batch iterations). This is the stage from which the three-circle Mapper was extracted. By contrast, the grayscale-normalized versions of the CIFAR-10 images produced filters that were harder to interpret, especially in the first layer of the fully-trained networks, despite the fact that a first-layer primary circle did emerge in the early stages of training.
Chapter 7

Our Methods

As mentioned before, our work’s main focus was, using the persistent homology and Mapper algorithms, to observe mid-training topological changes occurring in the layers of a CNN with multiple separable convolution layers. In particular, we looked to find points during these networks’ training where patterns emerged pointing to the formation of nontrivial homology in the weight space generated by a collection of independently trained CNNs. In fact, as we will describe in the next chapter, our results not only pointed to the formation of such patterns during training, but also showed a sudden increase in validation accuracy during a prominent turning point in the formation of these patterns.

These TDA algorithms were applied to a collection of 518 binary classification CNNs trained on a dataset of chest X-Ray images [20]. These were sorted into two categories: patients with pneumonia and patients without pneumonia. This dataset is pre-sorted into training, validation, and testing partitions. However, in training CNNs on them, we found that the models performed better in terms of validation and testing accuracy if all the images in the dataset were combined and randomly shuffled into training, validation, and testing partitions (using the original proportions) rather than using the prepackaged partitions. As we will describe, these CNNs and our training method were informed by a process detailed in Amy Jang’s article [19].

The data we used consisted of 5856 images, split into a "train" folder of 5216 images, a "val" folder of 16 images, and a "test" folder of 624 images. Each subset was markedly biased toward pneumonia images, and overall the set contained 4273 pneumonia images and 1583 non-pneumonia images. A sample of the images is given in Figure 7.1. We used a random reshuffling of the images into training, validation, and test sets of the same size as the original folders.

7.1 CNN Architecture for Our Work

The convolutional layers in our networks, standard and separable, are detailed in Table 7.1.

<table>
<thead>
<tr>
<th>Layer 1</th>
<th>Layer 2</th>
<th>Layer 3</th>
<th>Layer 4</th>
<th>Layer 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard</td>
<td>Separable</td>
<td>Separable</td>
<td>Separable</td>
<td>Separable</td>
</tr>
<tr>
<td>$3 \times 3$ convolution, 16 kernels</td>
<td>$3 \times 3$ convolution, 32 kernels</td>
<td>$3 \times 3$ convolution, 64 kernels</td>
<td>$3 \times 3$ convolution, 128 kernels</td>
<td>$3 \times 3$ convolution, 256 kernels</td>
</tr>
</tbody>
</table>

Table 7.1: Convolutional layers used in the CNN for this paper. All successive layers in this CNN consist of the standard dense classifier format used for the end of a CNN.

Each of these 5 convolutional layers is followed by max pooling and batch normalization. The network follows the sequence of convolution layers with a dense classifier, standard for classifier CNNs, and concludes using the Adam gradient descent algorithm with a learning rate that starts at .01 then
Figure 7.1: sample of x-ray images from the dataset with their corresponding labels: NORMAL and PNEUMONIA.

decreases exponentially. The exponential decrease function is \(0.01(1 + e^{-x})\), where \(x\) is the current epoch of training.

By looking at a summary of the architecture of the actual model we used (Figure 7.2) and contrasting it with a version using standard (rather than separable) convolutions (Figure 7.3), we can confirm that, indeed, there are significantly fewer trained parameters in the CNN using separable convolutions.

Before, during, and after training each network for 50 epochs, we have applied techniques from Topological Data Analysis to the trainable parameters (weights) in the collection of networks. In particular, we applied these techniques after every 5 epochs of training and used the a testing set to observe changes in testing accuracy. This was done by starting with the \(3 \times 3 \times C\) filters from each layer, separating each into \(C\ 3 \times 3\) blocks along the depth dimension, then flattening each of these \(3 \times 3\) blocks into points in \(\mathbb{R}^9\). These resulting points are the data used for analyzing the weights from a topological perspective.

As noted earlier, there is a significant imbalance in the X-ray data due to the presence of many more pneumonia images in the dataset than normal images. By altering the loss function used for updating the CNNs, we accounted for this imbalance by weighting the loss value assigned to each normal image with more importance than the loss assigned to each pneumonia image. In particular, with \(n\) being the size of the training dataset, \(n_0\) being the number of non-pneumonia training images, \(n_1\) being the number of pneumonia training images, and \(j_i^{(c)}(\theta)\) being the loss associated with the \(i^{th}\) observation labeled with class \(c\), we set \(J(\theta) = \alpha_0 \sum_{i=1}^{n_0} j_i^{(0)}(\theta) + \alpha_1 \sum_{i=1}^{n_1} j_i^{(1)}(\theta)\), where \(\alpha_0\).
7.2 Training Process and Extracting the Needed $\mathbb{R}^0$ Points

The 518 CNNs were each trained each for a total of 50 epochs and were saved in .h5 form every 5 epochs to obtain a snapshot of their training process: a batch size of 128 was used, which converts to 33 gradient descent iterations over the course of a single epoch, and thus 1650 over the entire training process for one CNN. This allowed us to evaluate each model on our testing set, then extract the weights from each model to be separated and flattened into normalized mean-centered $\mathbb{R}^0$ vectors.

<table>
<thead>
<tr>
<th>Layer (type)</th>
<th>Output Shape</th>
<th>Param #</th>
</tr>
</thead>
<tbody>
<tr>
<td>sequential_63</td>
<td>(None, 90, 90, 16)</td>
<td>155</td>
</tr>
<tr>
<td>sequential_64</td>
<td>(None, 45, 45, 32)</td>
<td>810</td>
</tr>
<tr>
<td>sequential_65</td>
<td>(None, 22, 22, 64)</td>
<td>2656</td>
</tr>
<tr>
<td>sequential_66</td>
<td>(None, 11, 11, 128)</td>
<td>9400</td>
</tr>
<tr>
<td>dropout_35</td>
<td>(None, 11, 11, 128)</td>
<td>0</td>
</tr>
<tr>
<td>sequential_67</td>
<td>(None, 5, 5, 256)</td>
<td>35200</td>
</tr>
<tr>
<td>dropout_36</td>
<td>(None, 5, 5, 256)</td>
<td>0</td>
</tr>
<tr>
<td>flatten_7</td>
<td>(None, 6400)</td>
<td>0</td>
</tr>
<tr>
<td>sequential_68</td>
<td>(None, 512)</td>
<td>3279360</td>
</tr>
<tr>
<td>sequential_69</td>
<td>(None, 128)</td>
<td>66176</td>
</tr>
<tr>
<td>sequential_70</td>
<td>(None, 64)</td>
<td>8512</td>
</tr>
<tr>
<td>dense_31</td>
<td>(None, 1)</td>
<td>65</td>
</tr>
</tbody>
</table>

Total params: 3,402,348
Trainable params: 3,399,048
Non-trainable params: 2,400

Figure 7.2: listing of parameter amounts in each layer of the network described in Table 7.1.

<table>
<thead>
<tr>
<th>Layer (type)</th>
<th>Output Shape</th>
<th>Param #</th>
</tr>
</thead>
<tbody>
<tr>
<td>sequential</td>
<td>(None, 90, 90, 16)</td>
<td>512</td>
</tr>
<tr>
<td>sequential_1</td>
<td>(None, 45, 45, 32)</td>
<td>4768</td>
</tr>
<tr>
<td>sequential_2</td>
<td>(None, 22, 22, 64)</td>
<td>18752</td>
</tr>
<tr>
<td>sequential_3</td>
<td>(None, 11, 11, 128)</td>
<td>74308</td>
</tr>
<tr>
<td>dropout</td>
<td>(None, 11, 11, 128)</td>
<td>0</td>
</tr>
<tr>
<td>sequential_4</td>
<td>(None, 5, 5, 256)</td>
<td>296192</td>
</tr>
<tr>
<td>dropout_1</td>
<td>(None, 5, 5, 256)</td>
<td>0</td>
</tr>
<tr>
<td>flatten</td>
<td>(None, 6400)</td>
<td>0</td>
</tr>
<tr>
<td>sequential_5</td>
<td>(None, 512)</td>
<td>3279360</td>
</tr>
<tr>
<td>sequential_6</td>
<td>(None, 128)</td>
<td>66176</td>
</tr>
<tr>
<td>sequential_7</td>
<td>(None, 64)</td>
<td>8512</td>
</tr>
<tr>
<td>dense_3</td>
<td>(None, 1)</td>
<td>65</td>
</tr>
</tbody>
</table>

Total params: 3,748,705
Trainable params: 3,746,305
Non-trainable params: 2,400

Figure 7.3: listing of parameter amounts in each layer of the network described in Table 7.1, but with standard convolutions in place of separable convolutions.
The percent accuracy from evaluation on the testing set and all convolutional kernels from each of the networks were saved at each 5-epoch checkpoint from each of our CNNs. The average observed testing accuracy for all 518 networks is plotted in Figure 7.5. From looking at several example validation/training accuracy plots, we saw a similar trend in the individual CNNs, an example being shown in Figure 7.4. We see a stable testing accuracy for approximately the first 15 epochs, then a sudden sharp increase until 25 epochs, where the accuracy stabilizes again. In the next chapter, we will see that this increase in accuracy coincides with significant changes in the homology of the spaces generated by several layers of our CNNs.

![Figure 7.4: Example validation accuracy plot, taken at the end of each epoch, of one of our CNNs trained on the X-Ray dataset.](image)

![Figure 7.5: Average testing accuracies, taken every 5 epochs, from our set of 518 CNNs.](image)

Each of the data points used to construct these spaces was obtained using the a separation and flattening process similar to the steps described in the previous chapter. For the first layer (the standard convolution), the weights were stored as $3 \times 3 \times 3 \times 16$ arrays, representing 16 kernels each with $3 \times 3$ channels. Each of these channels were reshaped to 1-dimensional (in array terms) arrays of 9 entries, corresponding to 24864 points in $\mathbb{R}^9$ across the 518 CNNs. These points were mean-centered and normalized to $S^7$.

The code used to reshape the kernels into $\mathbb{R}^9$ points is displayed in Figure ??, and the number of points collected in each layer is shown in Table 7.2.
Table 7.2: The each number of $\mathbb{R}^9$ data points collected from each layer across our 518 CNNs.

<table>
<thead>
<tr>
<th>Layer 1</th>
<th>Layer 2</th>
<th>Layer 3</th>
<th>Layer 4</th>
<th>Layer 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>(3)(16)(518) = 24864</td>
<td>(16)(518) = 8288</td>
<td>(32)(518) = 16576</td>
<td>(64)(518) = 33152</td>
<td>(128)(518) = 66304</td>
</tr>
</tbody>
</table>

For the remaining 4 separable convolutions, we followed a similar procedure, again breaking our kernels apart into $3 \times 3$ channels for reshaping into $\mathbb{R}^9$ points. However, this procedure applied only to the first step of separable convolution, i.e. the depthwise or channelwise convolution, and so our applications of persistent homology and the Mapper algorithm did not include all parameters in the separable layers of these CNNs.

Note that the second step of separable convolution, the $1 \times 1 \times C$ convolution step, does not have the same connection to image patches as the first step. The first depthwise convolution step is where parameter channels of size $3 \times 3$ are applied directly to the output nodes of the previous layer, which have the same grid formation as the image data. Recall that the second step’s major function is to finish the full application of convolution to an intermediate layer of nodes: thus, the distribution of its parameters is not expected to pertain to the image patches described in [23].

In addition to observing our collection of 518 CNNs at multiple different stages of training, we also observed the datasets generated from their kernels at multiple different density filtrations, as detailed in the following section. The proportion of these filtrations varied from 1 to 14% for each layer: see the next section for discussion of trends observed in each layer’s persistent homology and Mapper diagrams over the course of training at each filtration level.

### 7.3 Preprocessing and Specifications for TDA Procedures

Before applying any TDA procedures to our $\mathbb{R}^9$ points, we applied a density filtration and a normalizing to the unit sphere $S^7$ centered at the origin.

This was done in a standard way, by mean-centering and scaling each mean-centered point to have Euclidean norm 1. That is, for each $x = (x_1, x_2, \ldots, x_9)$ in our point cloud $X \subset \mathbb{R}^9$, letting

$$\bar{x} = \frac{\sum_{i=1}^{9} x_i}{9}$$

and

$$\bar{x} = (\bar{x}, \bar{x}, \ldots, \bar{x}),$$

we defined

$$x_{norm} = \frac{x - \bar{x}}{||x - \bar{x}||_2}$$

. From here

$$X_{norm} = \{x_{norm} | x \in X\}$$

was the set of points used from this point forward, including in the subsequent preprocessing step of density filtration.

Determining a density measure for each $\mathbb{R}^9$ point was done by observing a $k$-nearest neighbor measure. This $k$-nearest neighbor density measure, as described in [1], is defined below. A summary is available in Yen-Chi Chen’s University of Washington lecture\(^1\) at the link below.

\(^1\)http://faculty.washington.edu/yenchic/18W_425/Lec7_knn_basis.pdf
Definition 7.3.1. Given $x \in X \subseteq \mathbb{R}^d$, the $k$-nearest neighbor (kNN) density estimator is

$$\hat{p}_{\text{knn}}(x) = \frac{k}{n} \cdot \frac{1}{V_d \cdot (R_k(x))^d}$$

where $V_d = \frac{\pi^{d/2}}{\Gamma(d/2 + 1)}$ is the volume of a $d$-dimensional ball of radius 1 and $R_k(x)$ is the distance from $x$ to its $k^{th}$ nearest neighbor in $X$.

The motivation behind using such an estimator is also given in [1]. Let

$$B(x, R_k(x)) = \{y : ||y - x|| \leq R_k(x)\}$$

be the closed ball in $\mathbb{R}^d$ with center $x \in X$ and radius $R_k(x)$. Then by definition of $R_k(x)$, we can conclude that, writing the random variables generating each point of $X$ as $X_1, X_2, \ldots X_n$,

$$\frac{k}{n} = \sum_{i=1}^{n} I(X_i \in B(x, R_k(x))),$$

i.e. the proportion of points of $X$ inside of $R_k(x)$ must be $\frac{k}{n}$.

For a large $n$, we can then use $\frac{k}{n}$ as an estimate for the true probability that $X_i$ is in $B(x, R_k(x))$, i.e.

$$\Pr(X_i \in B(x, R_k(x)) = \frac{1}{B(x, R_k(x))} p(y) dy$$

and additionally, we choose $k$ to be a small quantity relative to $n$, so we may expect only minor changes in density inside $R_k(x)$, i.e. $p(y) \approx p(x)$. Thus, replacing $p(y)$ with $p(x)$ in the equation above, we get

$$\Pr(X_i \in B(x, R_k(x)) \approx \frac{1}{B(x, R_k(x))} p(x) dy$$

which, since we used $\frac{k}{n}$ to estimate $\Pr(X_i \in B(x, R_k(x)))$, yields

$$\frac{k}{n} \approx p(x) \cdot V_d \cdot R_k(x),$$

i.e.

$$p(x) \approx \frac{k}{n} \cdot \frac{1}{V_d \cdot R_k(x)}.$$  

[1] also describes a method for choosing $k$ as a function of $n$ for optimizing the mean-squared error of $\hat{p}_{\text{knn}}(x)$ as an estimator of $p(x)$ in the special case where $d = 1$, i.e. points are selected from the real line. In this special case, this estimator’s mean-square error is of rate $O((\frac{k}{n})^{\frac{1}{2} + \frac{1}{4}})$, leading us to choose $k = c_0 n^{\frac{1}{10}ac45}$ for a constant $c_0$ for an optimal convergence rate of $O(N^{1\frac{45}{ac45}})$. However, since we are dealing with the higher dimensional sample space $\mathbb{R}^d$, finding an optimal convergence rate in MSE is a more complicated process, and our work yielded promising exploratory results using a constant $k = 200$ in our density filtrations across all layers of our CNNs. In future work, we may move on to explore more complex CNNs trained on more complex datasets using these TDA techniques, and in these cases we may look to more precisely tune the $k$ parameter, observing the differences in the resulting analyses based on the $k$ used in density filtration.

Note that for our purposes, this density estimator is used only to rank points in $X$ by their values of $\hat{p}_{\text{knn}}(x)$, and choose the $(100p)\%$ of points from $X$ with the largest values of $\hat{p}_{\text{knn}}(x)$ to keep when performing persistent homology and Mapper analyses. Since we are not looking to compute exact estimates for density values for each point, the only quantity relevant for ranking points by $\hat{p}_{\text{knn}}(x)$ is the
distance $R_k(x)$, which we will refer to here as a dispersion measure. Clearly this dispersion measure is inversely proportional to the density estimate, and so we can choose high-density points by equivalently choosing low-dispersion points.

For each layer and each training stage, in our work, the $\mathbb{R}^3$ points derived from the trained kernels were processed to form a $n \times n$ distance matrix, containing the Euclidean distance between each pair of points in the dataset. Each of these columns was then sorted from smallest to largest so that for each point $x_i \in X$, each of the other points in $X$ were ranked by proximity to $x_i$. The $(k+1)^{th}$ row of this column-sorted matrix was then selected as our dispersion measure for each of the points in $X$. This row was then sorted from smallest to largest, where smaller values indicated higher-density points, and the leftmost $100p\%$ of the entries in this row were kept for use in the persistent homology and Mapper analyses.

**Example 7.3.2.** For an example of applying this density filtration procedure to a very small set of points, consider $x_1 = (.5, .1, .0)$, $x_2 = (.6, .4, .3)$, $x_3 = (.8, .6, .5)$, and $x_4 = (.2, .0, .5)$. Their distance matrix, rounded to two decimal places, would then be as follows.

\[
\begin{pmatrix}
0 & .44 & .77 & .59 \\
.44 & 0 & .35 & .6 \\
.77 & .35 & 0 & .85 \\
.59 & .6 & .85 & 0 \\
\end{pmatrix}
\]

Sorting each of our columns yields the matrix for dispersion measure selection.

\[
\begin{pmatrix}
0 & 0 & 0 & 0 \\
.44 & .35 & .35 & .59 \\
.59 & .44 & .77 & .6 \\
.77 & .6 & .85 & .85 \\
\end{pmatrix}
\]

Note that, especially when observing this very small sample of points, the dispersion/density measure is highly dependent on the $k$ value we select. In particular, if we select $k = 1$, $x_2$ and $x_3$ are assigned the lowest two dispersion values (both .35), as they are the two closest points to one another in the set. However, if we select $k = 2$ or $k = 3$, then $x_3$ is assigned the largest dispersion value (.77), as $x_3$ is much further away from $x_1$ (distance .77) and $x_4$ (distance .55), compared to its distance from $x_2$ (.35, as noted above).

We used a $k$ value of 200 for each filtration in all layers of or CNNs, so the 201st row was selected as the dispersion measure for each point, i.e. the reciprocal of our density measure. The density filtration process then concluded by removing all points in the top $100(1-p)\%$ of the dataset, in terms of dispersion measure. In cases where $p\%$ of the sample size was not an integer, the floor function was used to determine the number of data points to be kept.

For each density filtration of each set of points, we produced a persistence diagram using a Vietoris-Rips filtration and 15 Mapper diagrams. Both these analyses used a Variance Normalized Euclidean distance measure, which equates to scaling each column of the dataset to set its variance equal to 1. The Python code for this process is shown in Figure ??, and the code for the persistence diagram is shown in Figure ??.

Each of these 15 Mapper diagrams was constructed using the first two principal components of each point as the 2-dimensional projection/filter function. Additionally, each used a DBSCAN clustering algorithm.

Each individual Mapper diagram represented a unique pairing of the Mapper’s foundational parameters: the number of cubes and the percent overlap. The number-of-cubes argument was selected from the list $[10,20,30]$ and the percent overlap argument was selected from the list $[.5,.6,.7,.8,.9]$. The loop for constructing one of our Mapper complexes given a number of cubes and a percent overlap is shown in Figure ??.
An example persistence diagram and corresponding Mapper complex for the final layer of our collection of CNNs is displayed in Figure 7.6.

![Persistence Diagram and Mapper Complex](image)

**Figure 7.6:** Mapper diagram constructed from kernels in the fifth and final convolutional layer of our CNNs, along with the accompanying persistence diagram.

### 7.4 Technical Breakdown

The Python code for processing the X-Ray images, constructing the CNNs, and creating the persistence diagrams and Mapper images, the X-Ray image data, and our trials’ output (the persistence diagrams and interactive Kepler Mapper complexes) can be found at [https://github.com/ecourt93umsl/XRayCNNTDA](https://github.com/ecourt93umsl/XRayCNNTDA). We make several references to this code in this section, which you can view at the link above.

Each image was transformed into an appropriate input type for the CNN by using Jang’s `decode_img()` function, which uses the `decode_jpeg()` function from TensorFlow’s `io` module. `decode_jpeg()` takes in an image and extracts its pixel values as integers from 0 to 255, inclusive. To decrease the number of values needed for input to the CNN, we used the resize method from tensorflow’s `image` module, decreasing the resolution to $180 \times 180$ using its default method of bilinear interpolation.

**Remark 7.4.1.** In the definition below, the function $f$ will represent the intensity value for a pixel in one of the channels of an input image.

**Definition 7.4.2** (bilinear interpolation). For $x_1 < x_2, y_1 < y_2 \in \mathbb{R}$, a bilinear interpolation function $f : (x_1, x_2) \times (y_1, y_2) \to \mathbb{R}$ has arbitrary defined values $f(x_1, y_1), f(x_2, y_1), f(x_1, y_2),$ and $f(x_2, y_2)$ and is defined elsewhere by

$$
    f(x, y) = \frac{1}{(x_2 - x_1)(y_2 - y_1)} \left( x_2 - x \right) \left( y_2 - y \right) \begin{pmatrix} f(x_1, y_1) & f(x_1, y_2) \\ f(x_2, y_1) & f(x_2, y_2) \end{pmatrix} \begin{pmatrix} y_2 - y \\ x_2 - x \end{pmatrix}
$$

The CNNs we used were generated using the Keras package, a subpackage of TensorFlow, and were trained using GPU from Google Colab. The preliminary standard convolutional layer for each network was formed using Keras’s `Conv2D` objects, and separable convolutional layers were formed using `SeparableConv2D` objects. As detailed in Table 7.1, all convolutions were of $3 \times 3$ form.

We used some code functions, adapted from [19], to implement the CNN architecture itself. As mentioned before, our code is viewable at the GitHub link at the beginning of this section. The `conv_block()` and `dense_block()` functions, establish our general setup for implementing, respectively, a separable convolutional block and the dense (fully-connected) block to be included at the end of the network for binary classification. Again, for the convolutional block, the sequence is a separable convolution (standard convolution for the first block), followed by a max pooling layer, followed by a batch normalization, whereas for the dense block, we use a standard linear transformation with ReLU activation as described in Chapter 2 followed by a batch normalization. The implementation of the CNN architecture itself is then defined, via calls to the `conv_block()` and `dense_block()` functions, in a function called `build_model()`. The `build_model()` function as seen on the GitHub page produces a CNN with the specifications shown.
in Figure 7.2.

The weighted loss function, as described in 7.1, was implemented by applying the function pictured in Figure ??, to be used with the class_weight parameter of the fit() function in Keras. Note that class 0 applies to normal images, class 1 to pneumonia images.

The 9-dimensional points obtained from this process were the points observed using persistent homology, via the ripser package, and Mapper analysis, via the Kepler Mapper package, from each layer and 5-epoch snapshot during training. For the final layer, this process demanded more RAM than was provided in Google Colab’s High-RAM runtime (with a Colab Pro+ account), so we sampled a random subset of 57000 points from the original set of layer-5 kernels from which to select density-filtered subsets for analysis.
Chapter 8

Our Results

Recall: to observe the patterns present in CNNs trained on the X-Ray dataset, we trained 518 CNNs of the architecture from Chapter 7 each for 50 epochs, then extracted the weights from the beginning convolutional layer as well as each of the depthwise convolutional kernels, as outlined in the previous chapter.

**Notation 8.0.1.** Recall that our CNNs each consisted of a standard convolution, followed by four separable convolution. We will be referring to these layers as, from beginning to end, $C_1$, $SC_1$, $SC_2$, $SC_3$, and $SC_4$.

To reiterate, each $3 \times 3$ channel was flattened to a point in $\mathbb{R}^9$, mean-centered at zero, and normalized to the unit sphere. Python was used for implementing the TDA techniques described in Chapter 5: for persistent homology analysis, we used the ripser package, and for drawing Mapper diagrams, we used the Kepler Mapper package. For each of the images presented here, we used a $k$ parameter of 200, and unless otherwise mentioned, the Mapper diagrams will use 20 cubes and 70 percent overlap.

For CNNs trained on these X-Ray images, the results in every layer across our collection of CNNs looked markedly similar to the circle observed in Carlsson and Gabrielsson’s work with the MNIST dataset. This single-loop pattern can be observed from looking at both persistence and Mapper diagrams for each layer throughout the network: this includes both the standard convolutional layer at the beginning of the network and each successive separable convolutional layer.

Further, investigating nodes in four different regions of the Mapper diagrams and taking the average of the collection of points within each revealed that these kernels contain (as expected by the primary circle) edge- and corner-type patterns, as shown in Figure 8.1, along with the corresponding Mapper and persistence diagrams in Figure 8.2.

Additionally, recall we saved copies of each CNN at every 5 epochs of training and applied the same TDA techniques to these intermediate models at each layer. This allowed us to track testing accuracy throughout training and to track observable patterns in our CNN collection at intermediate training steps, and to distinguish between the layers for an understanding of what topological information each layer is learning.

Finally, note that we will frequently reference the average test accuracy trend of the 518 CNNs as shown in Figure 7.5. Looking at training plots for several of the 518 CNNs, we see that consistently, accuracy started near 75% before sharply increasing starting around 15-20 epochs, then stabilizing near 95% around 25 epochs, where it remained until finishing training at 50 epochs. This pattern can be seen in the example CNN accuracy plots shown in Figures 7.4 and 7.5, taken from, respectively, the results of one of the 518 networks and the average accuracy across all 518 networks.
Figure 8.1: visualizations of four edge-detecting kernels constructed from taking the means of all 9-dimensional data points in four regions of the Mapper shown in Figure 8.2 (left), taken from the fourth separable layer of our 518 CNNs with a $p = .11$ density filtration.

Figure 8.2: Mapper diagram used to derive the kernels in Figure 8.1, along with the persistence diagram to verify the clear appearance of a single circle in the point cloud.

Interestingly, we found that the primary circle appeared earliest and most clearly in the last two separable convolution layers. For both of these final convolutional layers $SC_3$ and $SC_4$, the primary circle was immediately visible from the 5-epoch stage onward, both in the Mapper and persistence diagrams, although as we will see, the penultimate layer did show minimal breakdown of the primary circle during training.

8.1 Summary of Observations by Layer

8.1.1 $SC_4$

Starting with $SC_4$, we can observe that every persistence and Mapper diagram we generated (with the aforementioned 20 cubes and 70% overlap), using density filtrations at each integer percentages from $p = .01$ to .14, clearly displays the single nontrivial cycle at all stages of training. Figure 8.3 shows the persistence and Mapper diagrams for this final separable convolution layer at 5 epochs of training and a $p = .14$ density filtration.

To show the stability of this final layer, the Mapper diagram shown in Figure 8.2 and persistence diagram in Figure 8.2 are taken from $SC_4$ at 50 epochs of training and a $p = .11$ density filtration. Thus, this final layer in its final stage of training also produced the mean patches shown in Figure 8.1.

From this point, we will work backward through the network: generally, the layers near the beginning of the network tended to be more dynamic in their kernel distributions than the later layers.
8.1.2 \textit{SC}_3

As just mentioned, there was a remarkable stability seen in the kernels of the second-to-last layer \textit{SC}_3, not only in \textit{SC}_4. The 5-epoch and 50-epoch endpoints for this layer are shown in Figures 8.4 and 8.5 respectively.

Figure 8.4: Persistence and Mapper diagrams for \textit{SC}_3 at 5 epochs of training and density filtration $p = .14$.

Figure 8.5: Persistence and Mapper diagrams for \textit{SC}_3 at 50 epochs of training and density filtration $p = .11$. 
One very slight deviation specific to this layer happened during the second half of the training process, and is only significantly noticeable when using a $p = .01$ density filtration. When looking at this density filtration, there is a breakdown in the single-cycle pattern from the perspective of both the persistence and Mapper diagram (even at a 90% overlap percentage). This breakdown is shown in Figure 8.6. It is also worth noting that a similar breakdown in the circle at this density filtration is visible as early as 20 epochs.

![Figure 8.6: Persistence and Mapper diagram taken from $SC_3$ at 50 epochs of training and density filtration $p = .01$.](image)

### 8.1.3 $SC_2$

Moving backward once more, we’ll now look at some results from $SC_2$. This layer generated substantially fewer points than the two that follow it, so we may suspect that viewing it at a higher density filtration will be necessary to find significant topological information. This does turn out to be the case, as density filtrations of $p = .01$ and .02 turn up no significant visible cycles in the corresponding $SC_2$ persistence or Mapper diagrams, regardless of the training stage. We can see this lack of a cycle both at 5 and 50 epochs at density filtration $p = .02$ in Figure 8.7 and Figure 8.8, respectively.

![Figure 8.7: Persistence and Mapper diagrams for $SC_2$ at 5 epochs of training and density filtration $p = .02$.](image)
Figure 8.8: Persistence and Mapper diagrams for $SC_2$ at 50 epochs of training and density filtration $p = .02$.

By contrast, when looking at larger density filtrations, we can find much more stable single-cycle patterns in the filters. For example, we can see that at $p = .08$, a significant cycle is clearly visible in both the persistence and Mapper diagrams at every stage of training. Figures 8.9 and 8.10 show this for the 5- and 50-epoch endpoints for $SC_2$ respectively, at density filtration $p = .08$.

Figure 8.9: Persistence and Mapper diagrams for $SC_2$ at 5 epochs of training, density filtration $p = .08$. 
Figure 8.10: Persistence and Mapper diagrams (10 cubes) for $SC_2$ at 50 epochs of training, density filtration $p = .08$.

### 8.1.4 $SC_1$

Moving back once again to our first separable layer $SC_1$, we find the latest layer in the network that, specifically at early stages in training, has no easily discernible single-cycle pattern at any density filtration we observed from $p = .01$ to $p = .14$. At 5 epochs, the only faint trace of a single cycle in the kernel distribution is visible in the persistence diagrams at density filtrations $p = .06$ and $p = .07$: we show the $SC_1$ persistence and Mapper diagrams at density filtration $p = .07$ in Figure 8.11, as well as at density filtration $p = .14$ in Figure 8.12 to show the contrast in the persistence diagram.

Figure 8.11: Persistence and Mapper diagrams for $SC_1$ at 5 epochs of training, density filtration $p = .07$. 
We start to see a clearer single cycle appear when we look at the 20-epoch training stage, which coincides with the spike in average testing accuracy trend seen earlier in Figure 7.5. This lends credence to the idea that not only do standard convolutional layers capture topological information when learning to properly categorize image data: this concept carries over to separable convolutional layers.

Figure 8.13 shows the corresponding $SC_1$ persistence and Mapper diagrams at a density filtration of $p = .07$. On the other hand, as we see in Figure 8.14, at density filtration $p = .14$, this single-cycle pattern is not as clear in the persistence diagram, and we can observe a similarly inconclusive Mapper diagram as shown beside it. In fact, above density filtration $p = .12$, we cannot see this single-cycle pattern clearly at all in the persistence diagram.
Figure 8.14: Persistence and Mapper diagrams for $SC_1$ at 20 epochs of training, density filtration $p = .14$.

At later stages of training (where the average testing accuracy stabilizes in Figure 7.5), the range of density filtrations at which the single cycle is clearly displayed in persistence diagrams expands significantly. Figures 8.15 and 8.16 show the endpoints $p = .06$ and $p = .1$ of this range: all density filtrations we observed in between these endpoints have a single clear cycle.

Figure 8.15: Persistence and Mapper (10 cubes) diagrams for $SC_1$ at 50 epochs of training, density filtration $p = .06$.

Figure 8.16: Persistence and Mapper (10 cubes) diagrams for $SC_1$ at 50 epochs of training, density filtration $p = .1$. 
8.1.5 $C_1$

Finally, we discuss the output of our TDA procedures on the beginning of the network: the standard convolutional layer $C_1$. The $C_1$ persistent homology and Mapper results are a bit unique, as the patterns observed over the training process depend heavily on the density filtration used. We’ll observe these patterns by looking at each training process using density filtrations of $p = .05$ and $p = .1$. While the $p = .1$ persistence diagrams follow a similar trajectory to the $SC_1$ layer, the $p = .05$ diagrams’ pattern is less unidirectional.

When using a $p = .05$ density filtration, there are, once again, some significant single-cycle patterns that emerge. In fact, throughout all of our training snapshots, from 5 to 50 epochs, the $p = .05$ density filtration shows a singular dominant cycle, but there is a bit of general instability, as there are some points during the training process at which some other significant cycles arise. Figure 8.17 displays the earliest snapshot taken of this $C_1$ layer at 5 epochs of training, and Figure 8.18 shows this layer at 40 epochs; in both instances, the persistence and Mapper diagrams show only a single extremely clear cycle.

On the other hand, there are some points, where, with kernels taken from this same layer and viewed with the same $p = .05$ density filtration, we have a second significant cycle, which, while its persistence value (represented by distance from the diagonal) is clearly smaller, is clearly more persistent than the remaining 1-cycles. These points are at 20 epochs, represented by Figure 8.19) and 50 epochs (where we see even a third significant, although less so than the two main cycles), represented by Figure 8.20).
Observing this layer with a density filtration of $p = .1$, we see a pattern somewhat more similar to the progression of the separable layers as seen earlier. From this perspective, we see this first layer start with a rather noisy persistence diagram and Mapper complex resembling a solid disc with no discernible "holes."

This pattern mostly persists from 5 to 35 epochs of training, with a gradual decrease in the amount of "noise" cycles and the emerging of a single dominant cycle. The endpoints of this process are displayed in Figures 8.21 and 8.22. Finally, at 40 epochs, this single cycle becomes much more clear, as we can see its corresponding point in the persistence diagram appear much further from the diagonal than in previous training stages, as in Figure 8.23.
8.2 Global Summary of Findings

To sum up our findings, we see that consistently, across every layer of these 518 CNNs, we can relate the formation of a single nontrivial cycle in our weight space (similar to what Carlsson deemed the primary circle) to an increase in testing and validation accuracy on the X-Ray dataset. In particular, referring back to Figure 7.4 for validation accuracy and Figure 7.5 for testing accuracy, despite the stable increase in training accuracy often observed in neural networks, there was a steep increase in accuracy on non-training data between 15 and 25 epochs of training. This suggests a sudden shift from a model close to the random initialization point (73% average testing accuracy at 5 epochs to 77.3% average testing accuracy at 15 epochs) towards a more generalizable model (92.7% average testing accuracy at 25 epochs).

Our layer-by-layer TDA observations, as described above, agree with this general trend, and per these observations, the layers separated themselves into two general categories. The separable convolution layers’ kernel sets later in the network contained a clear circle from the earliest observed training stage (5 epochs) that maintained itself relatively stably over the course of training (to 50 epochs). On the other hand, the earlier separable convolution and single standard convolution layers’ kernel sets started with noisier and harder-to-interpret persistent homology and Mapper outputs but showed a marked shift toward a clear single-cycle pattern at the 15- to 25- epoch stage, simultaneous with the sharp observed increase in validation and testing accuracy.

For additional confirmation that the cycles observed in other layers did in fact consist of the corner and edge kernels shown in $SC_4$ in Figure 8.1, we display a similar collection of mean filters in Figure 8.24.
Figure 8.23: Persistence and Mapper (30 cubes) diagrams for $C_1$ at 40 epochs of training, density filtration $p = .1$.

Figure 8.24: visualizations of four edge-detecting kernels constructed from taking the means of all 9-dimensional data points in four regions of the Mapper shown in Figure 8.20 (right), taken from the standard convolution layer of our 518 CNNs with a $p = .05$ density filtration.
Conclusion and Further Applications

In this dissertation, we have presented our contribution to a growing body of work combining the fields of Topological Data Analysis and machine learning. Our focus has been on the space of parameters learned by a special type of CNN model: one which uses depthwise separable convolutions. As we have seen, when compared to a standard convolution, this technique for constructing convolutional layers conserves on the number of multiplication and addition operations required for producing a prediction, as well as the number of model parameters to be trained. In particular, in training these CNNs on a set of X-ray images, we have observed that the kernels of such CNNs, when flattened to $\mathbb{R}^9$ points, tend to densely populate about a circle of edge and corner patterns. This suggests that, with fewer parameters than the standard-convolution CNNs studied by Carlsson and Gabrielsson, these special CNNs have still managed to learn similar topological (in particular homology-related) information about the image dataset’s $3 \times 3$ pixel-patch space.

In the future, we plan to apply similar analyses to separable CNNs trained on more complex datasets, such as CIFAR-10 or other natural image datasets, to see whether we can extend our findings related to Carlsson’s primary circle to the secondary circles (and resulting Klein bottle shape) as well. Additionally, we hope to apply TDA techniques to other CNN designs, such as the Inception and Xception approaches seen in Chapter 3. The use of topology in studying CNNs is a relatively young practice, and we are excited to see what patterns can be observed in the future, potentially in even more computing-intensive contexts. For example, CNNs have been applied to data types more complex than images, including video data. Additionally, we may be able to gain more insight into the learning process of CNNs by applying persistent homology in two dimensions or more, both to the data input to the CNNs and the CNNs themselves.
Bibliography


[20] Daniel Kermany. Labeled optical coherence tomography (oct) and chest x-ray images for classification, 2018.


