TOPOLOGICAL PRUNER

A NEURAL NETWORK PRUNER USING TOPOLOGICAL DATA ANALYSIS

W.M.M.J.U. Perera

199481D

Dissertation submitted in partial fulfillment of the requirements for the degree Master of Science

Department of Computational Mathematics

University of Moratuwa,

Sri Lanka

March 2022

Declaration

I declare that this is my own work and this thesis/dissertation does not incorporate without acknowledgement any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to University of Moratuwa the non-exclusive right to reproduce and distribute my thesis/dissertation, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

W. M. M. J. U. Perera

Signature of Student

Date:03/03/2022

Supervised by

Dr. Subha Fernando

Dr. Ashwini Amarasinghe

Signature of Supervisor(s)

Date:27/06/2021

Dedication

I dedicate my dissertation to my family and many friends.
A special feeling of gratitude to my loving parents, Mr. Susantha Perera and Mrs. Manel Irangani whose words of encouragement and push for tenacity ring in my ears
and also to my wife Sherine Weerasinghe who was the best source of inspiration.

Acknowledgement

First and foremost, I am grateful to my supervisors, Dr. Subha Fernando and Dr. Ashwini Amarasinghe for their valuable advices, continuous support, and patience during my research. Their immense knowledge and ample experience encouraged me all the time during my academic research and daily life. I would also like to extend my thankfulness to Prof. Asoka Karunananda for his guidance to prepare thesis materials and directing in the correct path. My sincere gratitude goes to all other lecturers and non-academic staff members who helped me to make this project a success.

I would also like to thank all my teachers from the kindergarten to the Master's level. Specially, Prof. U.N.B. Dissanayeke and Dr. Shelton Perera for strengthening my foundation in Mathematics. I would not be able to complete this research without the passion and curiosity in Mathematics, which they planted within myself.

Finally, I would like to express my gratitude to my parents, my wife and my friends. Without their tremendous understanding and encouragement in the past few years, it would not be impossible for me to complete my study programme.

Abstract

Architectural damage due to neural network pruning has been a research problem. To recover the accuracy loss, after pruning, pruned neural network needed to be trained further for a certain time period to gain the accuracy back. If the damage done by the pruning process is severe, some layers can collapse and at worse, the entire model may become untrainable. Therefore, pruning process needs to be done carefully to prevent any significant damage to the neural network. Although some existing approaches have been used to overcome this issue by identifying the salience of a neuron with respect to the overall architecture, it is not computationally efficient. Further, the exiting solutions do not count the topological meaning of the neural network architecture during the pruning process. We believe that identifying the salience of neuron with respect to the layer is sufficient to avoid severe damages to the overall architecture.

Topology, the champion of mathematical shapes, has been introduced to solve the aforesaid problem. We introduce 'Topological Pruner', a novel pruner that uses a genetic algorithm powered by a topological fitness function to identify removable neurons of each layer of a pre trained neural network. After pruning is done, the model is retrained so that the parameters of the remaining neuron can be readjusted to recover the model. As per to our knowledge this is the first ever attempt to use persistence homology, a topological tool for pruning.

Number of parameters, FLOPs and recovery time of the new pruner is evaluated on CIFAR10 dataset on VGG-16 architecture against L1Filter Pruner, L2Filter Pruner and FPGM Pruner. Evaluation results show that the new pruner competes well with the existing pruners. We conclude that, topological data analysis can be used to explain the recoverability and mitigate damage cause by neural network pruning.

Table of Content

List of Figures

List of Tables

List of Abbreviations

CBIS : Component based software engineering

FLOPS: Floating-point Operations per Second

GA : Genetic Algorithms

NAS : Neural Architecture Search

NNP : Neural Network Pruning

Chapter 1

1. Introduction

1.1 Prolegomena

This chapter gives an introduction to the research. In doing so, we have structured the introduction under several sub headings, namely, aims and objectives, background and motivation, problem definition, proposed solution and resource requirements. This chapter also provides an outline for the structure of the thesis.

1.2 Aims and Objectives

The aim of this research is to develop a solution for architectural damage due to neural network pruning with the use of topological data analysis.

In order to reach this aim, we have defined the following objectives.

- Critical review of neural network pruning research
- In depth study of technologies used for neural network pruning
- Design and develop a topology based neural network pruning solution
- Evaluate the topology based neural network pruning solution

1.3 Background and Motivation

Most of the recent research on machine learning is based on deep neural networks [1]. Many of the applications of deep neural networks, especially those having high accuracy needs a significantly large network structures and substantial amount of memory and computational power. One way of facing these challenges is by introducing algorithms that can support parallel processing like GPipe [2]. Another approach is reducing these resource

requirements by systematically pruning the neural network [1]. Latter approach produces a smaller network with similar accuracy.

Neural network pruning is not a novel subject area. One of the earliest research under neural network pruning is found as early as late 1980s [3]. Major universities like Stanford [4], Carnegie Mellon [5] have conducted research on neural network pruning.

Despite the interest shown in neural network pruning research, there has been limited research in architectural damage due to neural network pruning. Therefore, mitigating architectural damage due to neural network pruning has been a research challenge.

Further, our interest in topological data analysis realizes us that it is one of the best candidate tools to utilize in this research.

1.4 Problem Definition

Issues of neural network pruning can be classified into many categories including tradeoff between quality and accuracy, outperformance of random pruning in large scales, necessity for continues fine-tuning after each pruning step, immature pruning, architectural damage, etc. Based on this study we define research problem as architectural damage due to neural network pruning.

1.5 Proposed Solution

The hypothesis of our research is that the architectural damage caused by neural network pruning can be explained and mitigated by Topological Data Analysis. We have come up with a solution as shown in Figure 1.1.

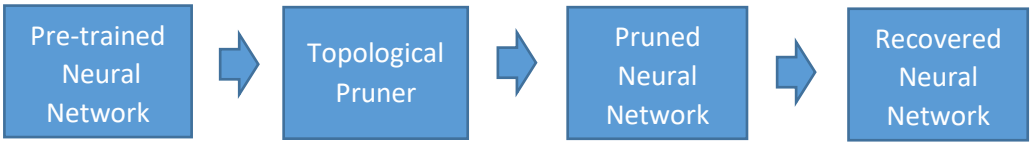


Figure 1.1: Top Level Architecture

Input: We have proposed this novel pruning technique, which uses the weight matrix of a partially trained network as an input. Convergence of the network is assumed.

Output: The output of the system is a pruned neural network. Note that it is not the same neural network with different weights. Speed up post processing produces this new network and its weight from the initial network (see recovered NN in Figure 1.1).

Preprocessing: In order to perform topological data analysis on the weight matrix, we first need to convert each layer to a topological space. Each neuron within that layer is mapped to an element in the topological space. Finally, a set of topological properties is identified for each layer.

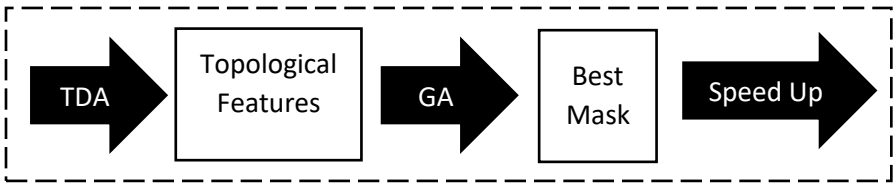


Figure 1.2: Topological Pruner Abstract Model

Process: In the novel approach, we use a genetic algorithm to generate a population of masks (each mask representing a possible prune). Then, a mask is selected for the pruning using a topological fitness function (see Figure 1.2).

1.6 Resource requirements

Since this is a theoretical research, no special equipment is required. However, for the evaluation, a significant amount of computational power and memory is required, since we need to generate a large population of neural networks using an evolutionary algorithm. Nevertheless, topological features need to be calculated for each individual in the population. Thus, following minimum resources are identified as required:

- A computer with 32 GB RAM and 3GHz or more processing power
- Windows 10 operating system
- Internet connectivity
- Google Colab account with GPU access

All these resources are currently available.

1.7 Structure of the Thesis

The rest of the thesis is organized as follows. Chapter 2 presents the background theory related to this research. It includes background knowledge of Neural Architecture Search (NAS). Chapter 3 gives a detailed review on Neural Network Pruning (NNP). The literature review identifies key researches on NNP, technologies used in NNP and related issues. In Chapter 4, two technologies are discussed which we use to come up with a solution, namely Topological Data Analysis (TDA) and Genetic Algorithms (GA). Chapter 5 presents the approach. Chapter 6 and Chapter 7 is dedicated to explain the design and implementation of the novel pruner respectively. An evaluation is provided in Chapter 8. The thesis is concluded in Chapter 9, by identifying outcomes, improvements, limitations and future improvements.

1.8 Summary

This chapter presented an introduction to the research that identified achievements in the field and the research challenges. The aims and objectives, background and motivation, problem definition, solution and resource requirements were discussed in detailed. The following chapter will give a detailed discussion on the background theory.

Chapter 2

2. Background Theory – Model Compression

2.1 Introduction

In Chapter 1, an overall picture of the research was provided and this chapter explains background theory required for the research. In doing so, we have structured the background theory under several sub headings, namely, neural architecture search, model compression techniques and neural network pruning. This chapter also provides a taxonomy for model compression.

2.2 Neural Architecture Search

Finding the most suitable neural architecture has been an open problem since the birth of feed forward networks and much research have been carried out under the umbrella of neural architecture search. Neural architecture search start by identifying a set of candidate models where one model is picked at a time from the model space using a search strategy and evaluate them. This process is repeated for all candidates to find the best fitting model [6].

2.3 Model Compression Techniques

Although it is preferred to have smaller networks, training a smaller network from scratch has several problems including under fitting and less efficiency due to overhead of avoiding local minimums [7]. Mode compression, a sub problem under neural architecture search tries to find a much simpler alternative model to an architecture, which is known to have a desired accuracy. There are many ways in the literature to perform model compression (see Figure 2.1) including pruning and quantization, low-rank factorization, transferred/compact

convolutional and knowledge distillation [8]. However, our focus in this thesis is on neural network pruning.

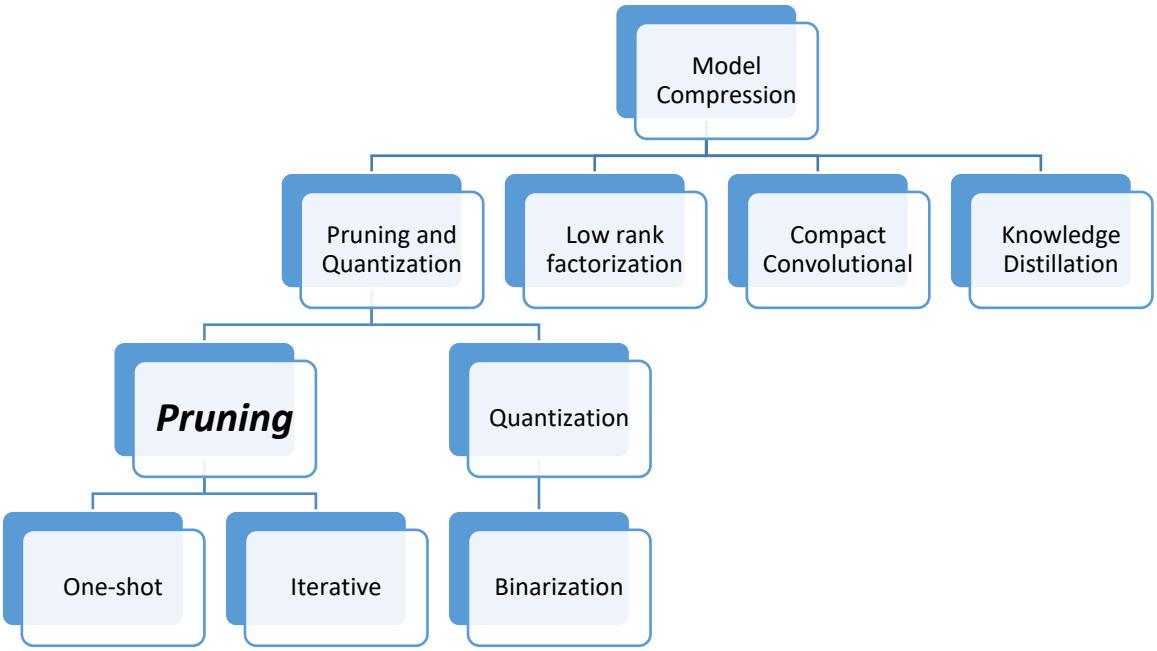


Figure 2.1: Model Compression Taxonomy

2.4 Neural Network Pruning

Neural network pruning can be done in different granularities using different techniques. Granularity levels can be connection wise, node wise, layer wise, channel wise, etc. Pruning can be iterative or one-shot. Pruning criterion can be based on absolute values of the weights in the simplest case. However, much complex metrics such as norms, cosine distance, entropy, correlation matrix can also be used as the pruning criterion [9].

2.5 Summary

This chapter presented the background theory required for this research. Neural network architecture search, model compression and neural network pruning theory is briefly discussed and the next chapter will give a detailed discussion on literature review.

Chapter 3

3. Literature Review – Neural Network Pruning

3.1 Introduction

In Chapter 2, a background theory required for the research was presented and this chapter gives a critical review of literature on neural network pruning. Accordingly, we have structured the literature review under several sub headings, namely, early development, latest development and challenges in neural network pruning. This chapter also summarizes challenges in neural network pruning and defines the research problem that we are interested in.

3.2 Gestation in neural network pruning

Neural network pruning is not a new subject area. One of the earliest research under neural network pruning is found as early as late 1980s [3].

Most of the recent research on machine learning is based on the deep neural networks. Many of the application of deep neural networks, especially those having high accuracy needs a significantly large networks structures and significant amount of memory and computational power.

One way of facing these challenges is by introducing algorithms that can support parallel processing like GPipe [2]. Another approach is reducing these resource requirements by systematically pruning the neural network [1]. Latter approach produces a smaller network with similar accuracy.

3.3 Breakthrough in Neural network pruning

Major universities like Stanford [4] and Carnegie Mellon[5] has conducted research on neural network pruning. Most of the pruning techniques for neural networks are based on the initial work of Dally and Han from Stanford with two other persons from NVIDIA [10].

This method can be briefly explained as follows. It starts with an arbitrary neural network structure and initializes its weight randomly. Then the neural network is trained until some evidence of convergence is found. Once found it starts pruning. Then weights are fine-tuned to recover any loss to the accuracy. Pruning and fine-tuning is repeated iteratively until the size of the neuron network is satisfactory [1].

Main issue of this approach and any of its variants are artificial recovery of damage to the accuracy by manipulating weights. More importantly some of the pruning techniques simply fails, resulting unrecoverable damages to the further learning [1].

A recent paper published in May 2021 by Seul-Ki Yeom et al, claim to introduce the first ever connection between neural network pruning and explainable AI [9]. It describes how one can prune a neural network by explaining its network structures. In this paper, they use Layer-wise relevance propagation (LRP).

3.4 Challenges in neural network pruning

Recoverability in neural network pruning has been a research problem. Pruning process needed to be done in small steps to prevent any significant damage to the neural network. To recover the accuracy loss, after each pruning step, pruned neural network needed to be trained further for a certain time to gain the accuracy back. This makes the pruning an iterative process, which is the cause for the performance loss. At worst, a wrong choice of pruning can end up the neural network irrecoverable.

One of the earliest attempts trying to overcome this is the Optimal Brain Damage[11]. It assumes that the diagonals of the Hessian Matrix is dominant which is not true in general [12]. A seminal paper, Optimal Brain Surgeon [12] proposed another alternative approach which does not require that assumption. However, to be more mathematical, it sacrifices the generality of the neural network [13]. Therefore, tradeoff between generality and neuron discrimination (selecting one neuron over other for pruning) is a challenge. Table 3.1 gives some issues and opportunities in recent attempts to overcome issues in neural network pruning and topological data analysis.

.

Table 3.1: Issues and Challenges in Current Technologies

Technology	pros	cons
TDA [14]	Neglect unnecessary local variations of data	Apply on initial data set
Gabrielsson and Carlsson Method [15]	Use topology on weight matrix	No consideration on pruning
Runtime Pruning [16]	Performance is better	Iterative
Pruning using Similar Functions [17]	Accuracy is better	Restrict to layer wise pruning
LRP [18]	Take meaning of structure of the neural network into consideration	No usage of topology
Sensitivity Analysis [11], [13], [19], [20]	Use second order derivatives to measure impact on error	Poor Generalization/ High sensitivity to overfitting

3.5 Problem Definition

Issues of neural network pruning can be classified into many categories including tradeoff between quality and accuracy, outperformance of random pruning in large scales, necessity for continues fine-tuning after each pruning step, immature pruning, architectural damage, etc.

Among the dozens of solutions available, one common problem is the expandability. Although many advance methods are available, mechanism is rather a black box. Why some neurons and connections are pruned is not clear [8]. Attempts try to address this issues end up having poor generality. Every method favors on some criterion, which is not proven to be directly related to recoverability allowing a neuron discrimination.

Based on this study we define research problem as how to explain recoverability and mitigate architectural damage due to neural network pruning.

3.6 Summary

This chapter presented a critical review of literature in neural network pruning, and identify achievements in the field and the research challenges. Here the research problem has been defined as how to explain recoverability and mitigate architectural damage due to neural network pruning. Next chapter, we give a detailed discussion on technology adopted for implementing neural network pruning.

Chapter 4

4. Technologies Adopted – Topological Data Analysis and Genetic Algorithms

4.1 Introduction

In Chapter 3, we provided a detailed literature review about neural network pruning. This chapter gives details about the technologies we adopted. In doing so, we have structured the technology adapted chapter under several sub headings, namely, algebraic topology, topological data analysis and persistence homology.

4.2 Topology as a Branch of Mathematics

Historically Algebra, Analysis and Topology were identified as the main three branches of pure Mathematics[21]. As shown in Figure 4.1, these branches were neither disjoint nor complete.

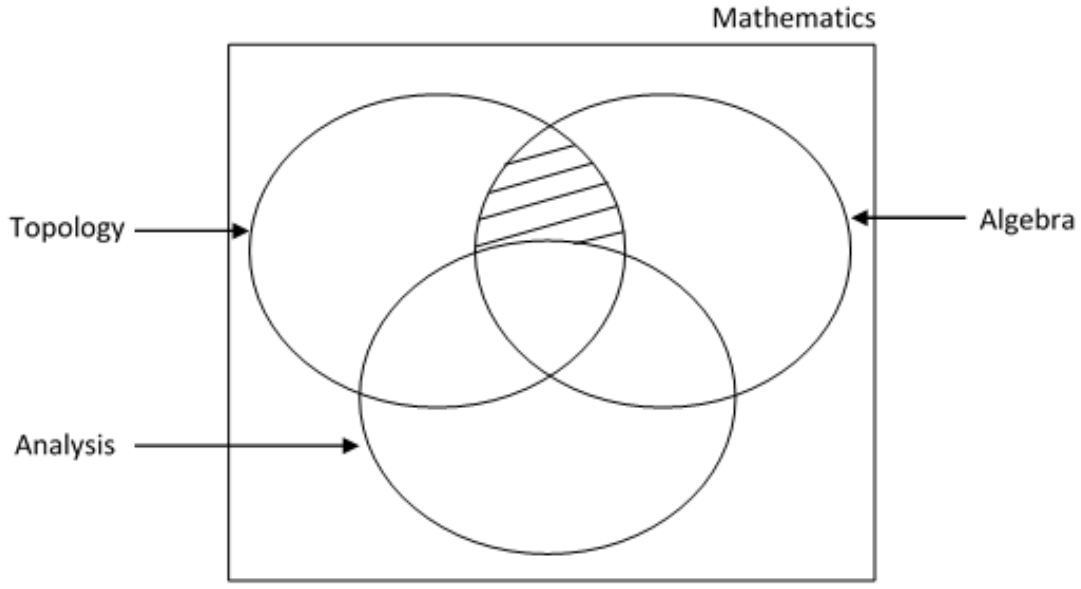


Figure 4.1: Branches of Mathematics (Algebraic topology is in the shaded area)

Algebra is the branch of Mathematics that deals with discrete structures. It provides abstract methods, tools and techniques required to study interesting pattern in a collection of discrete objects. Major areas under algebra includes Abstract algebra, Group Theory, Linear algebra, Algebraic geometry, Combinatorics, etc.

Analysis is the branch of Mathematics that deals with approximations of mathematical objects. It models complex mathematical objects with a sequence of much simpler, easy to study objects with the use of limits, the main tool of an analyst. Major areas under analysis includes real analysis, complex analysis, measure theory, functional analysis, differential equations, etc.

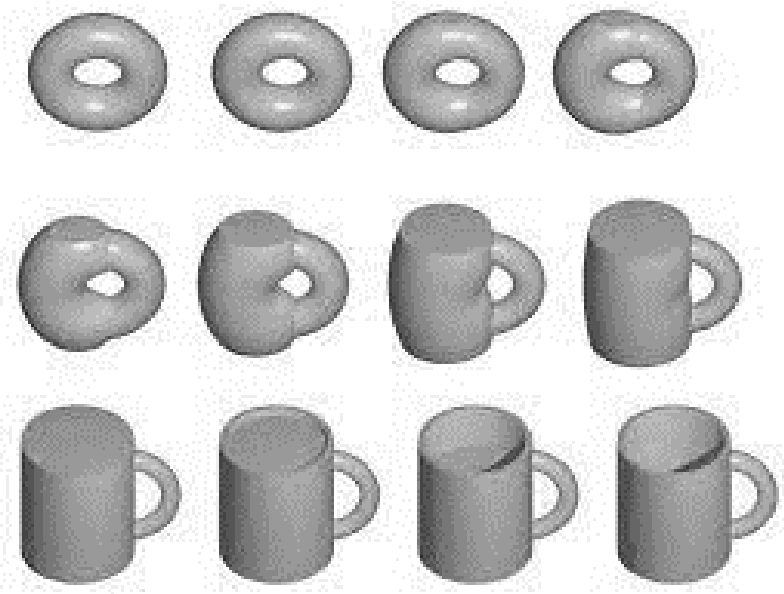


Figure 4.2: Topological Similarities

Topology is the branch of Mathematics that deals with continuous structures. In its most abstract levels, it studies about properties, which are invariant under continuous deformations. For a topologist, a doughnut and a coffee mug is indistinguishable (See Figure 4.2) as any of them can be transformed to the other with continues deformations such as stretching, twisting, crumpling, and bending. Major area under Topology includes General Topology, Algebraic Topology, Differential Topology, Geometry, etc.

Although topology is younger than the other two branches, its contribution to the modern Mathematics is commendable. Some problems that were earlier solved with a considerable effort using Algebra and Analysis, has become trivial now with the use of topology [22].

4.2.1 Definition of a Topological Space

There are few alternative definitions for a topological space. One provided here using open sets being the most common. Let X be a non-empty set and $\tau \subseteq \mathcal{P}(X)$. Then, τ is said to be a topology on the set X if it satisfies the following three axioms:

I. $\emptyset, X \in \tau$,
II. τ is closed under arbitrary union,
III. τ is closed under finite intersection.

Further, the ordered pair (X, τ) is said to be a topological space and we call the members of τ as open [23]. For example, let $X = \{a, b, c\}$ and $\tau = \{\emptyset, \{a\}, \{a, b\}, X\}$. It is easy to prove that (X, τ) is a topological space.

Some of the well-known topological spaces are 'Real Line', 'Complex Plain', 'Euclidian Spaces' (2D, 3D coordinate systems and their high dimensional analogies), 'Manifolds', Metric Spaces, etc.

4.2.2 The World through the eyes of a Topologist

One can be easily misled by thinking that the distance is the key to the idea of nearness. However, distance is sufficient but not necessary. For an example, when we say that John is a close to Tom, we are not actually referring to the exact distance between John and Tom. Instead, what we want to emphasize is the nearness. Existence of a topology τ on a set X is necessary to discuss the notion of nearness (aka neighborhood) of the elements of X. As shown in Figure , we call $U \in \tau$ as a neighborhood of $x \in X$ if $x \in U$ [24].

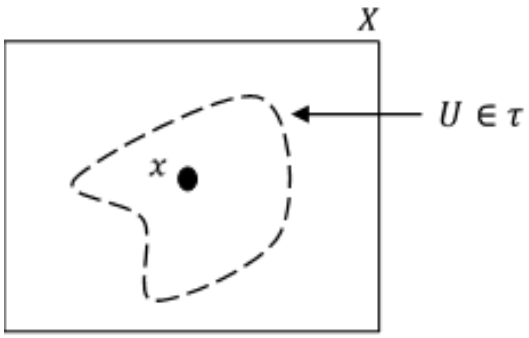


Figure 4.3: Neighborhood of x

Once the idea of nearness is available, we can easily make use of it to discuss the connectedness of the set X. Let (X, τ) be a topological space. We say that X is separated whenever the following three conditions are met simultaneously.

I. $U, V \in \tau$

II. $U \cap V = \emptyset$

III. $U \cup V = X$

A topological space is said to be connected, if it is not separated [25]. Said differently, X is said to be connected, if it cannot be expressed as a disjoint union of open sets (see Figure 4.4)

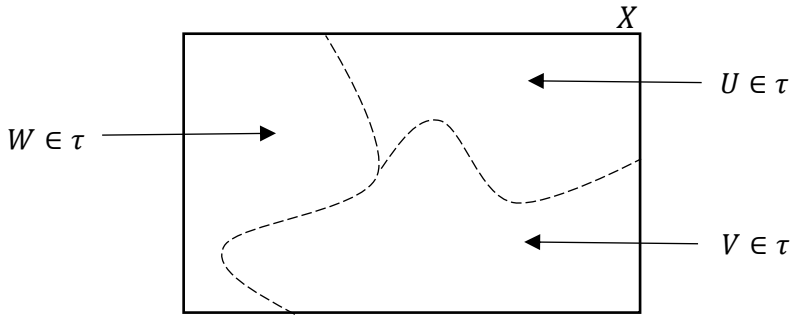


Figure 4.4: A Separated Space

Given the additional condition that the open sets U, V and W in Figure 4.4 are connected (by applying the definition of connectedness recursively), we can identify U, V and W as connected components of X. In topology, connected components can be set apart from each other as desired. However, separating a connected component (aka tearing) is not allowed. Any operation satisfying these two conditions is known as a continuous deformation (see Figure 4.5). Stretching, twisting, crumpling, and bending are some examples of continues deformations.

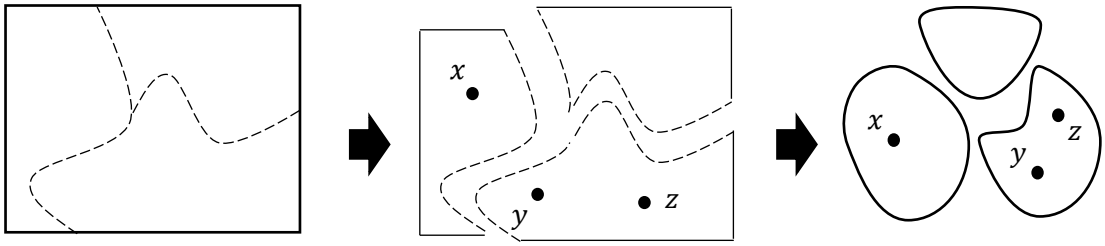


Figure 4.5: Continuous Deformations of a Topological Space

A topologist consider any two elements in a connected component (such as y and z in Figure 4.5) as near point and any two elements in different connected components (such as x and y in Figure 4.5) as far points regardless of the distance between them. This attitude gives topologists the ability to distinguish a disk from a doughnut (as it has a hole/handle) while keeping a disk equal to a coffee mug (see Figure 4.6).

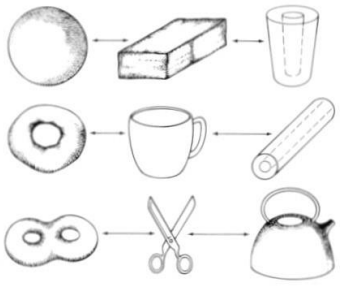


Figure 4.6: Equivalence in Topology

Thus, Topology becomes the ideal tool for any application where we want to focus not on properties of individuals but on the global realization of local constrains. This idea of global realization of local constrains is not a new thing. Even though we approached it with the aid of a heavy machinery 'Topology', it is none other than the very familiar idea 'shape'. Importance of this approach is that it gives us the capability of discussing about the shape (a global attribute of a space) without referring to the distances (a local attribute) of its members.

4.3 Algebraic Topology

In 4.2, we identified topology as one of the main branch of Mathematics. However, by 21th century, these divisions are no longer valid and meaningless with the seamless unification of Mathematics. Boundaries are getting unclear day by day as new knowledge is discovered. One such unification is through Algebraic topology (see Figure 4.1).

Besides all its powers, General Topology lacks tools and techniques needed to investigate interesting patterns among the elements of a Topological space, which are resulting from the topological features. Therefore, it needs to barrow it from Algebra. Algebraic Topology uses tools and techniques from abstract algebra to study patterns in topological spaces. Next, we may see some of the tools and techniques of algebraic topology.

4.3.1 Simplexes, polyhedrons and Simplicial Complexes

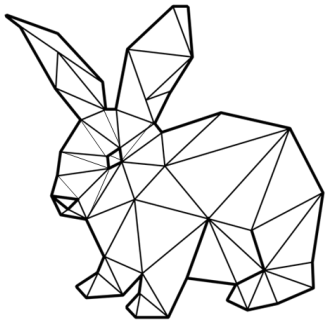


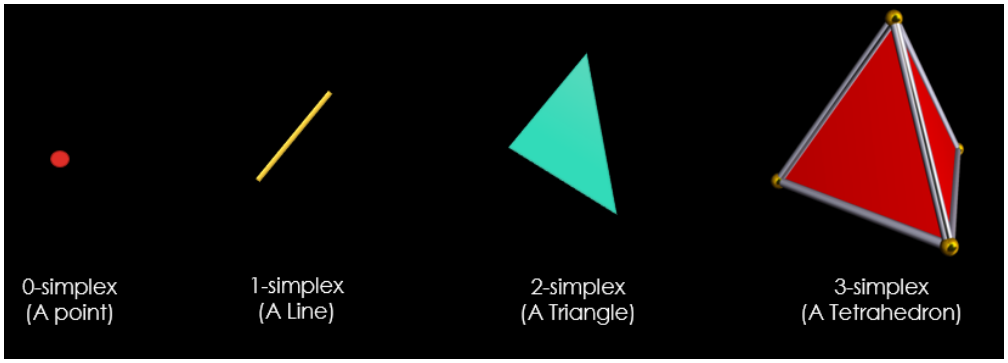
Figure 4.7: Polygon with Triangles

Triangles can be considered as the basic building blocks of any polygon (see Figure 4.7). Simplexes are the generalization of this idea to arbitrary dimensions. Table 4.1 and Figure 4.8 shows the first four simplicial complexes and their faces. Note that faces and edges of a n-simplex are also simplexes having (n-1) vertex points and (n-2) vertex point respectively [15].

Table 4.1: Faces of Simplexes

Name	Realization	Faces	Edges
0-Simplex	A point	-	-
1-Simplex	A Line	Points	-
2-Simplex	A Triangle	Lines	Points
3-Simplex	A Tetrahedral	Triangles	Lines

Although, there is a non-recursive definition for a n-simplex is available [26], following recursive definition is more intuitive to a person with less mathematical background. A point is a 0-simplex. Given the $(n-1)$-simplex and a point, which is not overlapping with the $(n-1)$-simplex, we can construct the n-simplex by connecting all the elements in $(n-1)$-simplex to the point. For an example, a tetrahedron can be constructed by connecting each point of a triangle to an external point.



A polyhedron is created by gluing simplexes together along their faces [26]. The operation ‘gluing together’ makes topologically connected components. Figure 4.9 shows some examples for polyhedrons.

Figure 4.8: First four simplexes

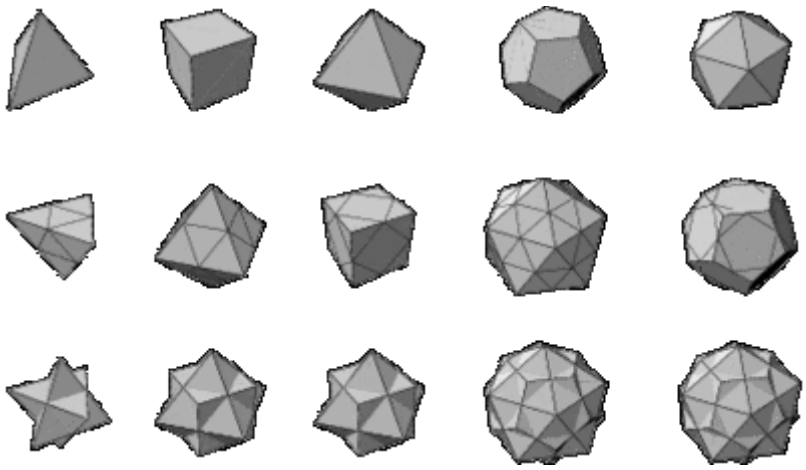


Figure 4.9: Examples for Polyhedrons

A Simplicial complex includes all possible decompositions a polyhedron into its building blocks. For an example, a triangle can be decomposed into a triangle, three edges and three vertices (see Figure 4.10).

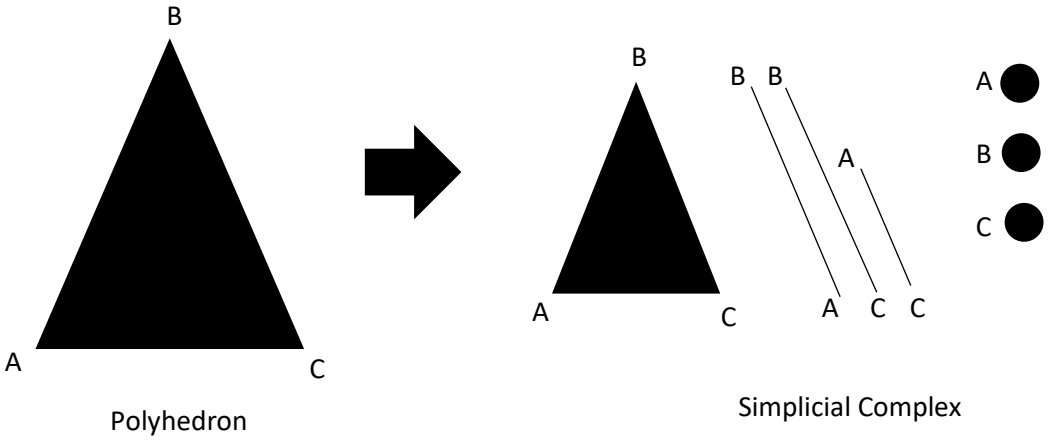


Figure 4.10: Decomposing a Polyhedral

A formal definition for a simplicial complex can be formulated as follows. A collection of simplexes K is said to be a simplicial complex if it satisfy the following two conditions [27].

I. Every face of a simplex in K is in K.

II. The intersection of any two simplexes of K is a face of each of them.

4.3.2 Persistence Homology, Bar codes and Persistence Diagrams

Once a simplicial complex is available, we need to extract topological features from it. Here we discuss some of the tools that can be used for the above task. Given a set of points, Persistence homology (PH) can identify topological features like clusters, holes, and voids within it [15]. To define persistence homology we need to have a filtration on a simplicial complex. Let $K_0, K_1, \dots, K_n$ be simplicial complexes such that $\emptyset = K_0 \subseteq K_1 \subseteq \dots \subseteq K_n = K$. Then $(K_0, K_1, \dots, K_n)$ is a filtration of K.

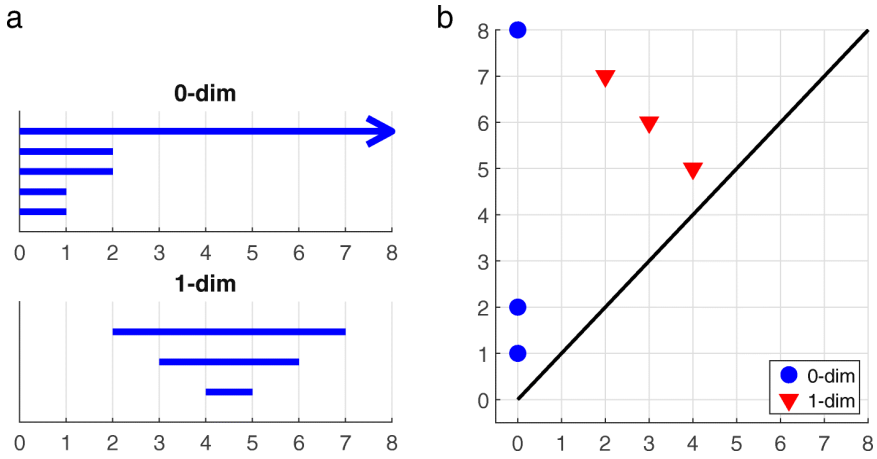


Figure 4.11: Bar codes and Persistence Diagrams

Given a point cloud, constructing this filtration is straightforward. As shown in Figure 4.12, a simplex is created when the neighborhoods of a point overlaps with another. For readers

who are interested in calculating it, a much computationally efficient dynamic programming algorithm in given in [28].

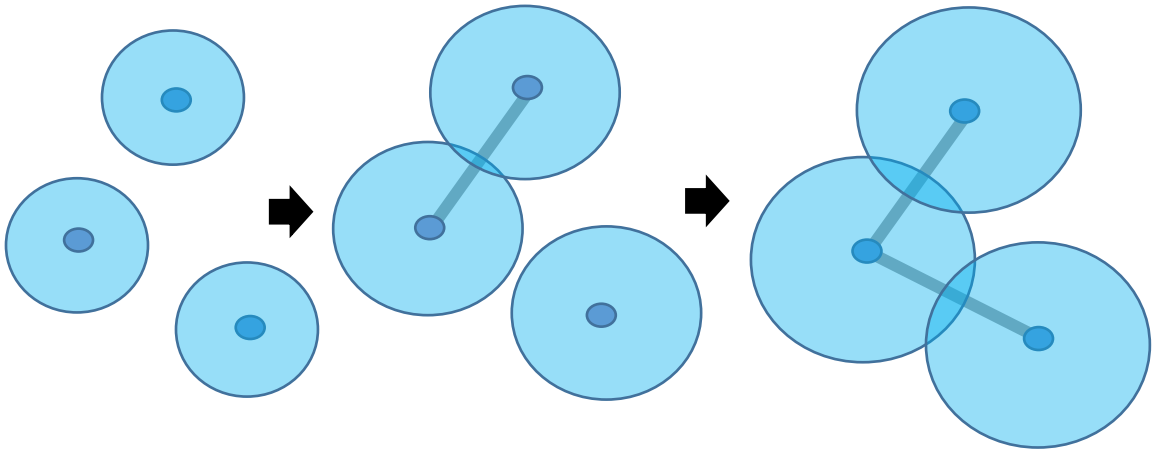


Figure 4.12: Generating simplicial complexes

Note that K_{i+1} has few simplexes which were not present in K_i. We identify $t = i + 1$ as the birth of such simplexes. On the other hand, if two components are connected at $t = j$, we keep the older component and identify at $t = j$ as the death of the younger component [28].

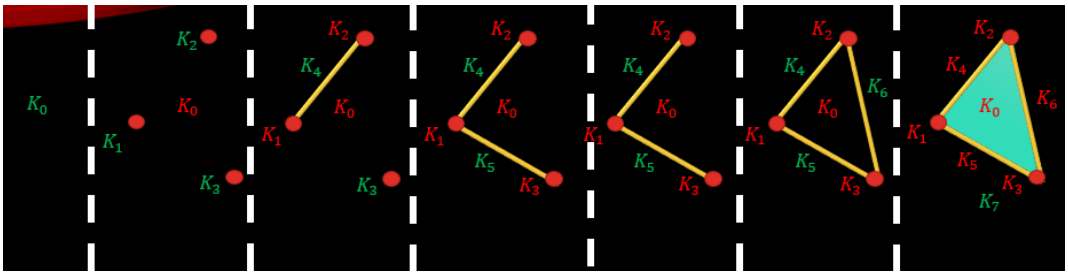


Figure 4.13: Filtration of the simplicial complexes

We use bar codes (Figure 4.11 part a) to represent the life span (aka persistence) of each component. Components with more life span becomes topological features while

components with less life span are neglected as noise. An alternative representation is through persistence diagrams (Figure 4.11 part b). It plots death of each component against the birth. Thus, as shown in Figure 4.14, a boundary is drawn to distinguish features from noise. Thus, noise is found near the diagonal.

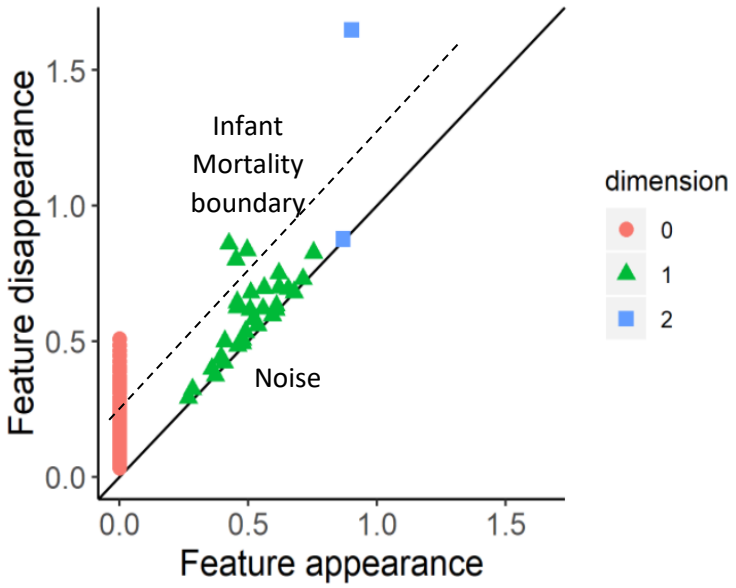


Figure 4.14: Features vs Noise

4.3.3 Bottleneck Distance

Although persistence diagrams can identify topological features in a simplicial complex, they cannot be directly used to compare the topological features of two simplicial complexes. Bottleneck distance is used to overcome this issue. Given two persistence diagrams, this distance calculates how similar they are.

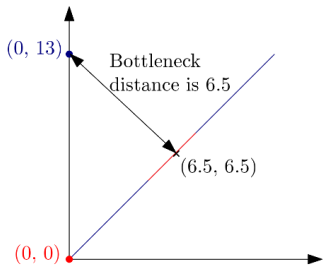


Figure 4.15: Pairing a point with its Projection

Minimal cost of matching over all possible matchings between the two persistence diagrams is known as the bottleneck distance [29]. A matching between two persistence diagrams can be done as follows. Each point of persistence diagrams is paired either with a point from other persistence diagram or (as shown in Figure 4.15) with the projection of it on the diagonal. Then the maximum distance between the paired points is called the cost of matching.

For two distinct point clouds, if the bottleneck distance is close to zero, then their shape is almost equal. Three point clouds shown in Figure 4.16 have similar topological properties. Bottleneck distance between any two diagrams would be close to zero.

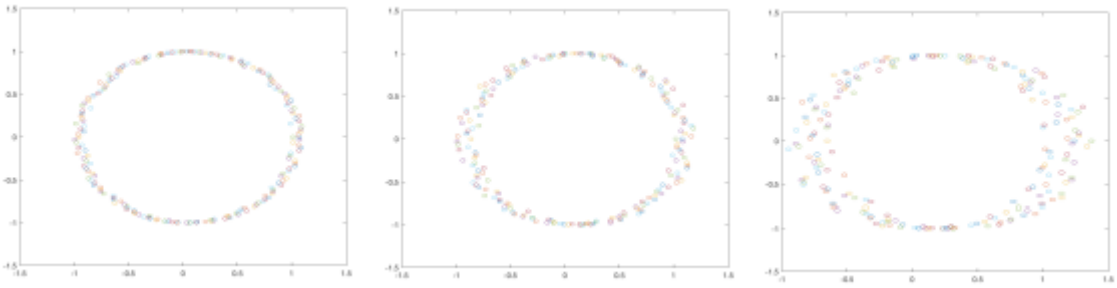


Figure 4.16: Three datasets with similar topological properties

4.4 Topological Data Analysis

As we have discussed in 4.2, Topology is the branch in mathematics, which study about the properties that are preserved through continuous deformations of objects. Data analysis is the branch of data science that study about the process of cleaning, transforming, and modeling data to discover useful information for decision-making. Topological Data Analysis (TDA), as its name suggests brings the two concepts together. It is used as a tool to uncover patterns in data sets. As illustrated in Figure 4.17, TDA is an interdisciplinary subject area which unities topology in data science.

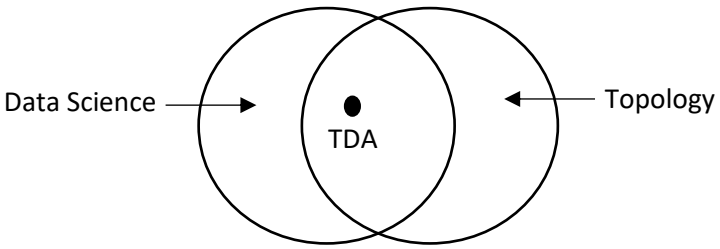


Figure 4.17: Topological Data Analysis

TDA promotes the idea that data has shape, and that shape has meaning that can be used for decision-making. Persistence homology is the basic tool use in topological data analysis [30]. Feature extraction was the main application of TDA. If the number of features in a data set is large, learning from it is difficult. For an example, pixel based image classification is hard. In such a cases, Convolutional neural networks (CNN) are used. CNNs extract features automatically. However, results generated by CNNs are less explainable.

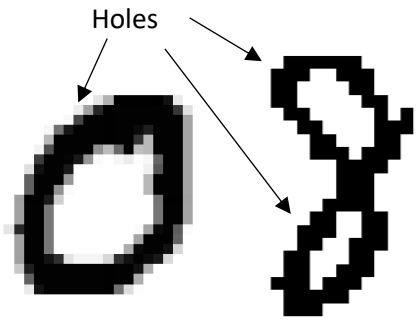


Figure 4.18: Distinguish Zero from Eight

As discussed in 4.2.2, topologists can distinguish '8' from '0' easily from a single topological feature. In fact, natural identification of Zero has nothing to do with pixels. As shown in Figure 4.18, Zero is not an Eight as it has two holes. TDA can find such interesting pattern in datasets. One such tool in TDA is 'Mapper' (see Figure 4.19).

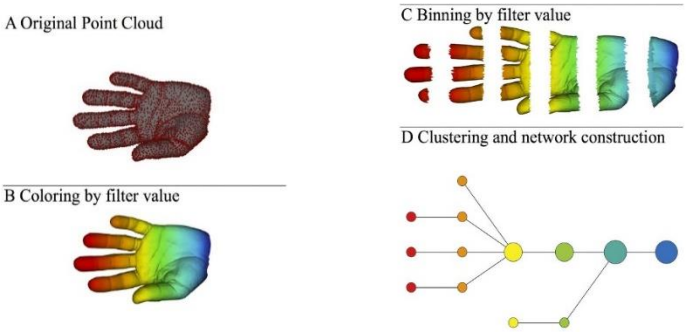


Figure 4.19: Mapper representation of a Hand shaped point could

In 2018, Gabrielsson and Carlsson from Stanford University, published a paper about a novel approach in TDA where weight matrix is used instead of the training data set [14]. In this paper, they could identify the contribution of each layer to the feature extraction process over time (see Figure 4.20). Since then various researches has used varieties of this approach.

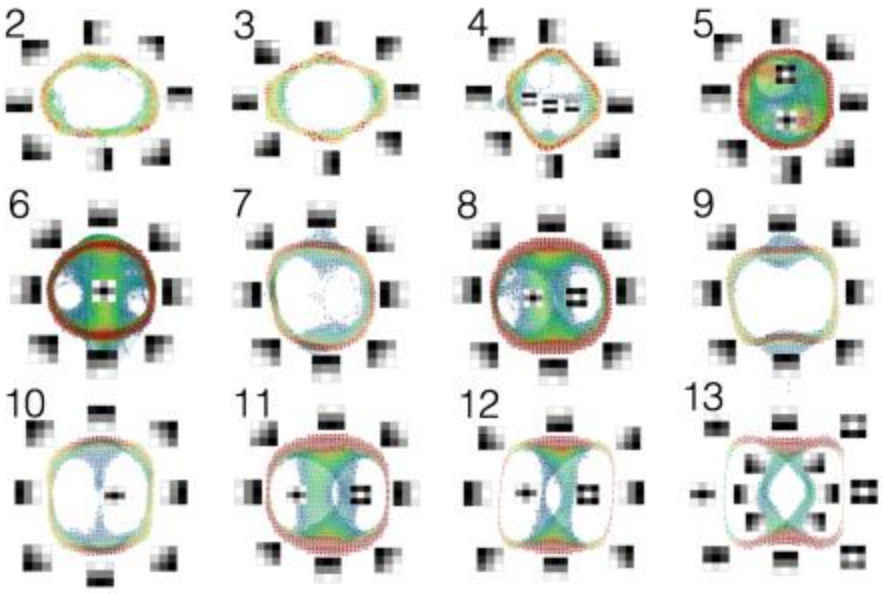


Figure 4.20: Responsibilities of each layer of VGG-16

4.5 Genetic Algorithms

Optimization problems, in its most general forms can be mathematically formulated as follows. Let S be the set of all feasible solutions to a problem. Define fitness (aka objective) function $f: S \to \mathbb{R}$. Aim of an optimization algorithm is to find $s^* \in S$ such that $\forall s \in S, f(s) \leq f(s^*)$. Such s^* is known as the global optimum [31].

Searching for $s^* \in S$ always bears a cost associated with it. This cost depends heavily on the algorithm being used. This cost is very high for most of the real world problems. Therefore, instead of finding the global optimum, we would like to satisfy by finding $s' \in S$ such that $f(s^*) - f(s') < \varepsilon$ where $\varepsilon > 0$. These methods are called approximate optimization algorithms. If some additional data (aka heuristics) is available, the searching process can be accelerated by such data. Heuristics can be either problem specific or algorithm specific. When they are algorithm specific, we call them metaheuristics. There are two favours of Metaheuristics.

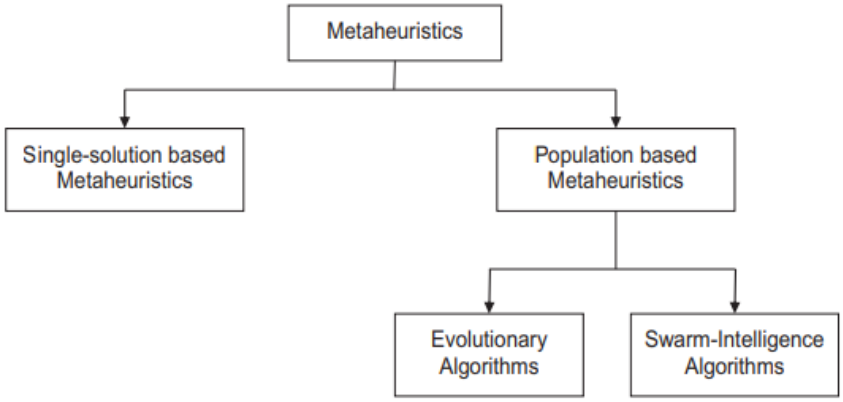


Figure 4.21: Taxonomy of Metaheuristics

Simulated Annealing [32] is a well know metaheuristic algorithm, which attempts to escape from local optimums by accepting non-improving neighbors. This class of algorithms is known as Single-solution based Metaheuristics. Another class of algorithms is population based Metaheuristics (see Figure 4.21). Genetic Algorithm (GA) falls under this category. GAs are optimization algorithms inspired by biological evolution, which utilizes the concept of survival of fittest [33].

General algorithm for GA is explained in Figure 4.22. GA starts by initializing a population of solutions. Then, fitness of each individual is calculated. Individuals with high fitness is allowed to be in the mating pool. Only the individuals in the mating pool are allowed to generate off springs for the next population. This process is continued until the population contains an individual with the desired fitness [34].

Input:
Population Size, n
Maximum number of iterations, MAX
Output:
Global best solution, Y_{bt}

begin
Generate initial population of n chromosomes $Y_i \ (i = 1, 2, ..., n)$
Set iteration counter $t = 0$
Compute the fitness value of each chromosomes
while $(t < MAX)$
Select a pair of chromosomes from initial population based on fitness
Apply crossover operation on selected pair with crossover probability
Apply mutation on the offspring with mutation probability
Replace old population with newly generated population
Increment the current iteration t by 1.
end while
return the best solution, Y_{bt}
end

Figure 4.22: General Algorithm for GA (Source: [34])

4.5.1 Encoding schemes

Encoding scheme decides as to how a real world candidate solution is converted to a chromosome [34]. Proper conversion makes various operation on chromosomes efficient and significantly reduces the encoding and decoding (process of interpreting the best fit to a real world representation) overhead. Figure 4.23 includes some of the most commonly used encoding schemes. Binary, permutation and value-based are some well-known encoding schemas [34].

Binary encoding is the most commonly used encoding scheme. In binary encoding, each individual in the solution space is represented using a binary bit string. This makes most of the chromosome operation straight forward. However, not all real world problems can be encoded using binary encoding.

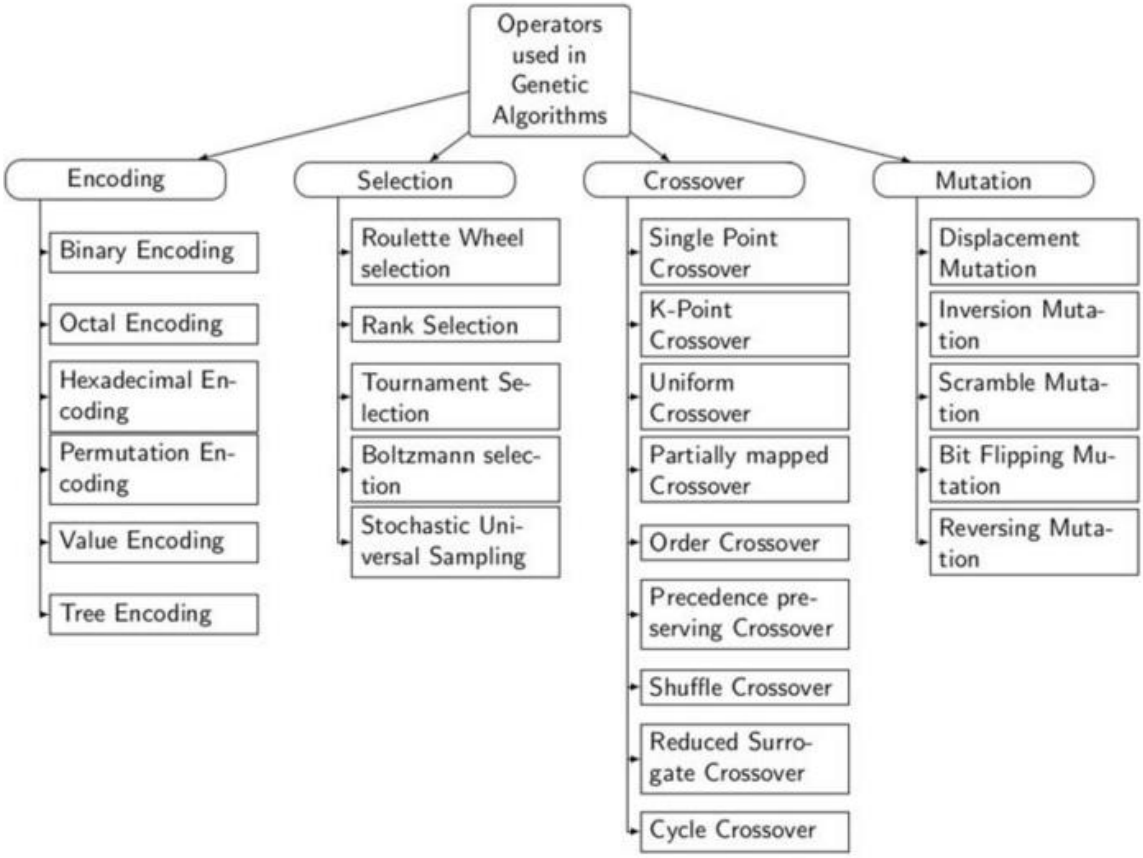


Figure 4.23: Operations use in GA (source [34])

4.5.2 Selection techniques

Selection technique determines whether an individual is allowed to participate in the reproduction process (aka mating pool). The convergence of the population is guaranteed through the selection technique. Therefore, a proper balance of exploitation and exploration is required. If we only allow best individuals to reproduce, there is a high possibility that

the algorithm is stuck in a local optimum. Hence, limited other individuals are also allowed to enter the mating pool to sustain exploration, while giving priority to the best-fitted individuals. Some of the selection techniques are shown in Figure 4.23. Roulette wheel, rank, and tournament are the most common among them [34].

4.5.3 Offspring Generation

Offspring generation is carried out through three operations, namely cloning, crossover and mutation. In cloning, an exact copy of the parent chromosome is created and this promotes elitism. Therefore, best-fitted individuals are allowed to enter the next generation as we can ensure that the good features are passed down to the younger generations.

Crossover operators (see Figure 4.23) are used to create offspring by mixing the features of two parents. The well-known crossover operators are single point, two-point, uniform, partially matched, order and cycle. The simplest form of crossover is single point crossover, which starts by selecting a random crossover point. The features of two parents which is beyond that point will be swapped [33]. Individuals in the mating pool with crossover probability are allowed to perform crossover operation. Crossover operations ensure that the younger individuals are different from their parents while keeping a mixture of features of their parents.

Mutation operation ensures that the diversity is maintained among generations. In contrast with crossover operation, mutation operation does not involve two parents. Individuals in the mating pool with mutation probability is selected to create an offspring, which is slightly different from its parent [34]. Some of the frequently used mutation operations are listed in Figure 4.22.

Irrespective of the operation used to generate offspring, we must ensure that the individuals generated are feasible solutions. One way of doing this is by carefully designing crossover and mutation operations so that the resulting chromosome is feasible at all times. However,

designing new operations are not trivial. Another approach is to add a penalty to the fitness function. This penalty reduces the fitness of infeasible solutions so that their participation in the parent pool is discouraged [34].

Genetic algorithms have shown a significant success in many of the problem domains. Specially, it is used when we need to find out a better solution to a problem to which a satisfactory solution is already available. However, a drawback of genetic algorithms is that it suffers from high resource requirements sometimes.

4.6 Summary

This chapter presented a detailed description about the technology adapted for the research and it discussed about the usage of topology, particularly on topological data analysis and persistence homology. Brief description of genetic algorithms was also depicted at the end of the chapter. The following chapter will give a detailed discussion approach used in this research.

Chapter 5

5. Approach – Neural Network Pruning using Topological Data Analysis

5.1 Introduction

In Chapter 4, we presented an overall picture of the technology adapted in the thesis. This chapter describes the approach that is used to solve the research problem. Accordingly, the approach is structured under several sub headings, namely, hypothesis, input, output and process. This chapter also indicates features of the novel approach and the possible users being benefited of this new algorithm.

5.2 Hypothesis

The hypothesis of the research is that the architectural damage caused by neural network pruning can be explained and mitigated by Topological Data Analysis.

5.3 Input

We have proposed a novel pruning technique, which uses the weight matrix of a partially trained network as an input. Convergence of the network is assumed. This input is extracted directly from the model parameters. Since the ‘Topological Pruner’ is agnostic on the type of the layer (whether convolutional, dense or etc.), any layer related arrangements need to be discarded before processing. Thus, some preprocessing is required.

5.3.1 Preprocessing

In order to perform topological data analysis on the weight matrix, each layer needs to be converted to a topological space. Each neuron within that layer is mapped to an element in the topological space. Finally, a set of topological properties is identified for each layer.

5.4 Output

The output of the system is a pruned neural network. It is not the same neural network with different weights. Speed up post processing produces this new network and its weight from the initial network.

5.5 Process

Weight matrix of the pre-trained neural network is first converted into a point cloud. This point cloud is used in the process of determining which topological properties need to be persisted. Best mask is determined then, using a genetic algorithm. This mask is applied to the original network thereafter, to produce the pruned neural network. Finally, the pruned neural network is retrained to gain accuracy.

5.6 Users

Anyone who is interested in training a neural network can use this approach. However, to start this algorithm they need to ensure that their neural network is trained up to a level, which shows signs of convergence.

Another set of potential users are who are willing to reduce the complexity of an already trained neural network without affecting its accuracy.

5.7 Features

One important feature of the novel algorithm is that it preserves recoverability. A neuron is selected for pruning if a closely similar neuron is found only. Therefore, extensive fine-tuning is not required. Another important feature is that this pruner has no neuron discrimination. A neuron is deleted not because of any of its local properties. It considers the redundancy of neurons in a particular layer and select redundant neurons to be deleted. Therefore, it improves the generality (reduces overfitting) of the neuron network.

5.8 Summary

This chapter presented a detailed description about the approach, which was used to solve the research problem. Hypothesis, input, output and process were explained in detail in this chapter. Features and potential users were also identified. In the next chapter, a detailed discussion will be presented about the design of the solution.

Chapter 6

6. Design – A topology based GA powered pruner

6.1 Introduction

This chapter designs the system processes explained in the chapter 5 as modules. For better understanding, design diagrams will be used. This system contains six modules namely, compressor, data collector, metric calculator, evolutionary, sparsity allocator and speed up. Input, output and process of each module will be identified and explained.

6.2 Compressor Module

Compressor Module acts as a placeholder to the neural network under pruning. As shown in Figure 6.1, it takes two inputs namely the pre-trained neural network and a list of configurations and output a mask that is latter used to produce the pruned neural network. Compressor module is initialized with a pruner mechanism attached to it. During the pruning process, it is the duty of the compressor module to determine what neurons to be pruned.

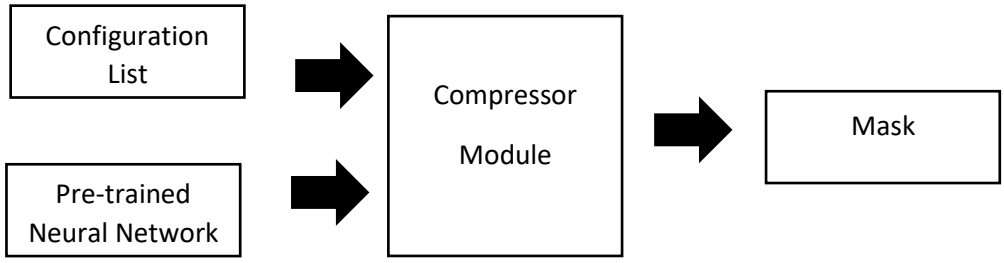


Figure 6.1: Abstract composer module

6.2.1 Configuration List

Configuration module is designed with the ‘convention over configuration’ concept in mind. Therefore, this list is optional. If the user is not satisfied with the default configuration, they can override the current configuration of the compressor by using the configuration list. Currently, it supports only three keys, namely sparsity, op_types and max_dimension. Sparsity level determines how many neurons can be removed. For an example, 90% sparsity indicates that the 90% of all neurons can be deleted. Higher the sparsity level, damage to the neural network is high. Op_types determines what layers need to be pruned. For an example, we can restrict only the compressor to prune convolutional layer leaving other types of layers such as dense layer intact.

6.2.2 Pre-trained neural network

Compressor assumes that the neural network provided to it is well trained. At least the neural network should show evidence of convergence. This is important that if the neural network is not trained well or its weights are random, compressor may identify wrong neurons as redundant. Convergence should be measured with training accuracy. Validation accuracy or the testing accuracy need not to be considered. In other terms, there is no harm by providing an over fitted neural network.

6.2.3 Mask

Shape of the mask generated by the compressor should match with the shape of the weight matrix. This mask is a binary mask. Zero indicates that a connection is marked for deletion. It is latter used for the actual pruning process

6.3 Data Collector Module

Data Collector Module is responsible for collecting weights from the neural network, and arrange them in point clouds. As shown in Figure 6.2, data collector module takes weights of the neural network and convert them to point clouds. This conversion is done by ignoring additional dimensions in weight arrangements by arranging all weights related to a particular neuron in a single dimension.

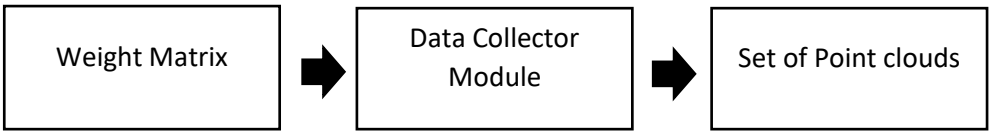


Figure 6.2: Abstract Data Collector Module

6.3.1 Weight Matrix

This is a list of weight matrices arranged in layers. Weights can be arranged in additional dimensions. For an example, weights of a 2D convolutional layer is having separate dimensions for channels and kernel data too. As there is no restrictions on the type of layers allowed, data collector module should be implemented to process any number of dimensions available.

6.3.2 Point Cloud

A point cloud is a set of data points in an arbitrary dimensional space. Data collector module outputs a point cloud for each layer. Dimension of the space is determined by the number of weights associated to a neuron in that particular layer.

6.4 Metric Calculator Module

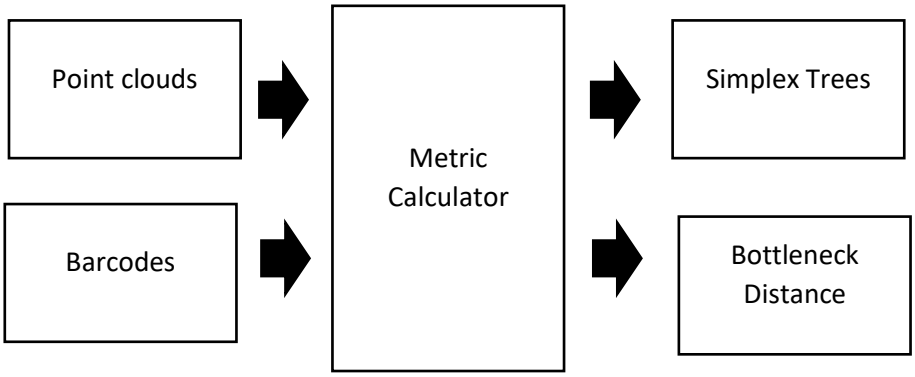


Figure 6.3: Abstract Metric Calculator Module

As shown in Figure 6.3, Metric calculator module has two responsibilities. Firstly, it takes a point cloud as an input and output the corresponding simplex tree. Secondly, it determines the bottleneck distance, once the barcodes are given. Barcodes, simplex trees and bottleneck distance follow their standard meanings described under 4.3.

6.5 Evolutionary Module

Evolutionary module communicates with the metric calculator module to determine the best mask based on topological features. Evolutionary module is powered with a topological fitness function. It determines the recoverability of mask being identified. As shown in Figure 6.4, evolutionary module takes a point cloud as an input and determines the best mask to be used for the pruning process using a standard genetic algorithm.

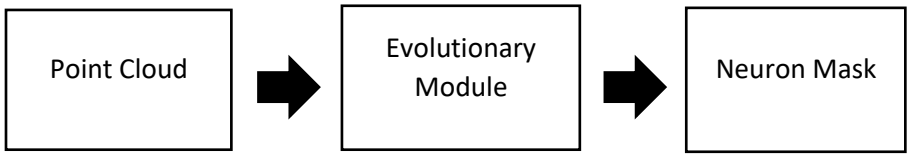


Figure 6.4: Abstract Evolutionary Module

6.5.1 Neuron Mask

This neuron mask represents the neurons selected to be pruned. It is also a binary mask. Its dimension should tally with the number of neurons in each layer.

6.6 Sparsity Allocator Module

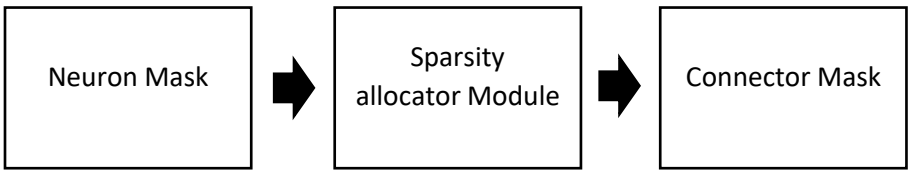


Figure 6.5: Abstract Sparsity Allocator Module

Sparsity allocator module (see Figure 6.5) converts the neuron mask to a connector mask. If a neuron is marked for deletion, this module should identify that the connection with the neuron should also be removed. Therefore, weights of such connections are made zero.

6.6.1 Connector Mask

Connector mask indicates what connections need to be pruned. Once this mask is applied on top of the weight matrix of the pre-trained neural network, weight matrix of the pruned neural network can be generated.

6.7 Speedup Module

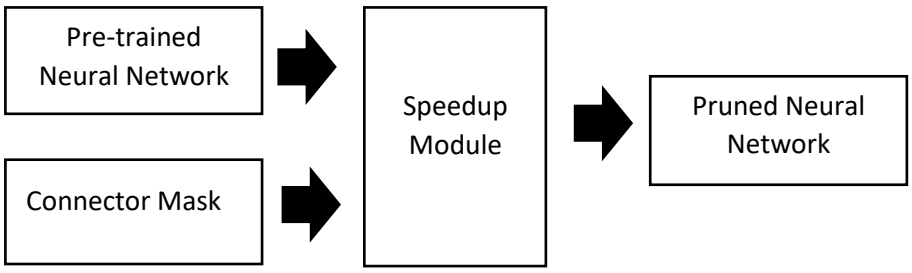


Figure 6.6: Abstract Speedup Module

As shown in Figure 6.6, speedup module takes two inputs namely the pre-trained neural network and the connector mask generated by the evolutionary module. The function of the speedup module is to generate the pruned neural network. Speedup module accomplishes this task by removing connections with zero weights from the architecture. Further, if all the connections are lost, that neuron need to be removed too.

6.7.1 Pruned Neural Network

Pruned neural network generated by 'Topological Pruner' is also having architectural damage. However, it can be recovered in a shorter period. This is done by retraining the pruned neural network for few epochs.

6.8 Summary

This chapter presented the design level details of each module. Six modules namely, compressor, data collector, metric calculator, evolutionary, sparsity allocator and speed up were explained with abstract diagrams. Under each module, input, output and process were explained. In next chapter, implementation of each module will be discussed.

Chapter 7

7. Implementation – Topological Pruner

7.1 Introduction

The design described in the previous chapter is realized in this chapter. This chapter starts by introducing the programing languages and frameworks used in this system. Software, hardware, algorithms and important code segments are listed and explained for each module. Further, data flow of the module is described using flowcharts wherever applicable.

7.2 Frameworks used

There are hundreds of libraries and frameworks written in Python. Especially when it comes to artificial intelligence, it has libraries covering almost all the knowledge areas. Therefore, Python was selected as the programing language for this research. As this research involves implementing, training and validating neural networks, ‘PyTorch’, an open source machine-learning framework was used. Its documentation can be found in [35].

7.2.1 Neural Network Intelligence

Neural Network Intelligence is a Microsoft powered lightweight toolkit that can be used for neural architecture search, model compression, feature engineering, etc. In this research, it is used as the framework for model compression. It also implements some of the commonly used pruners, which we have used for evaluating the novel pruner.

Topological Pruner, which we introduces in this research, inherits from the Basic Pruner defined in this framework. We have reused the compressor model and speedup model as their functionality matches with the design explained in 6.2 and 6.7. Model compression related documentation of NNI is available at [36].

7.2.2 GUDHI

The GUDHI library is a generic open source C++ library used for Topological Data Analysis. It has a python interface which we are using to implement concepts in topology like simplex trees, persistence homologies, barcodes, persistence diagrams and bottleneck distance. It also provides tools required for understanding and visualizing higher dimensional geometry. In this research, metric calculator module makes use of the functionalies available in this library. GUDHI documentation can be found here [37].

7.2.3 Pyeasyga

Pyeasyga provides a simple implementation of Genetic Algorithms in Python. As it follows the principle convention over configuration, only the fitness function needed to be defined. However, it provides configurations for changing encoding scheme, selection technique, crossover operation, mutation operation as well. In this research, functionalities of this library is used by the evolutionary module discussed in 6.5. Documentation of pyeasyga can be found here [38].

7.3 Special hardware, software and infrastructure used

In this research, Google Colaboratory was used, as training image classification neural networks require considerable computational power and memory availability. All the tensor calculations were carried on top of GPU with high-RAM. Further, Topological feature analysis in high dimensions also expects high memory and evolutionary module requires GPU computational power. Therefore, almost all the execution were done in this remote platform.

Apart from the remote infrastructure, no special software or hardware was used in this research.

7.4 Data collector implementation

Given a partially trained neural network, we start by extracting the values of trainable parameters. These parameters usually come in the form of tensors (see Table 7.1) organized by layers.

Table 7.1: Trainable parameters arranged by layers

Index	Name	Type	Weight Shape	FLOPs	#Params
0	feature.0	Conv2d	(64, 3, 3, 3)	1769472	1728
1	feature.3	Conv2d	(64, 64, 3, 3)	37748736	36864
2	feature.7	Conv2d	(128, 64, 3, 3)	18874368	73728
3	feature.10	Conv2d	(128, 128, 3, 3)	37748736	147456
4	feature.14	Conv2d	(256, 128, 3, 3)	18874368	294912
5	feature.17	Conv2d	(256, 256, 3, 3)	37748736	589824
6	feature.20	Conv2d	(256, 256, 3, 3)	37748736	589824
7	feature.24	Conv2d	(512, 256, 3, 3)	18874368	1179648
8	feature.27	Conv2d	(512, 512, 3, 3)	37748736	2359296
9	feature.30	Conv2d	(512, 512, 3, 3)	37748736	2359296
10	feature.34	Conv2d	(512, 512, 3, 3)	9437184	2359296
11	feature.37	Conv2d	(512, 512, 3, 3)	9437184	2359296
12	feature.40	Conv2d	(512, 512, 3, 3)	9437184	2359296
13	classifier.0	Linear	(512, 512)	262144	262656
14	classifier.3	Linear	(10, 512)	5120	5130

As shown in Table 7.1, each of these tensors are four-dimensional for Conv2d layers and two-dimensional for linear layers. In either case, first dimension corresponds to the number of neurons (aka filters when it comes to convolution layers) in each layer and rest of the dimensions corresponds to weights ending at a particular node. For an example $(64, 3,3,3)$ corresponds to a convolution layer with 64 filters each having $27\ (= 3 \times 3 \times 3)$ connections.

Next, we convert first layer tensor to a point cloud. Dimension of the point cloud is determined by the number of weights ending at each neuron of the layer. As shown in Figure 7.1, each neuron is mapped to a point in this cloud. Therefore, i[th] neuron of the layer can now be represented by a vector $\overline{x^{(i)}} = \left({x_0}^{(i)}, {x_1}^{(i)}, \dots, {x_j}^{(i)}, \dots, {x_n}^{(i)}\right)$. For an example, $(64, 3,3,3)$ is converted to a point cloud in 27 dimension, which has 64 points in it.

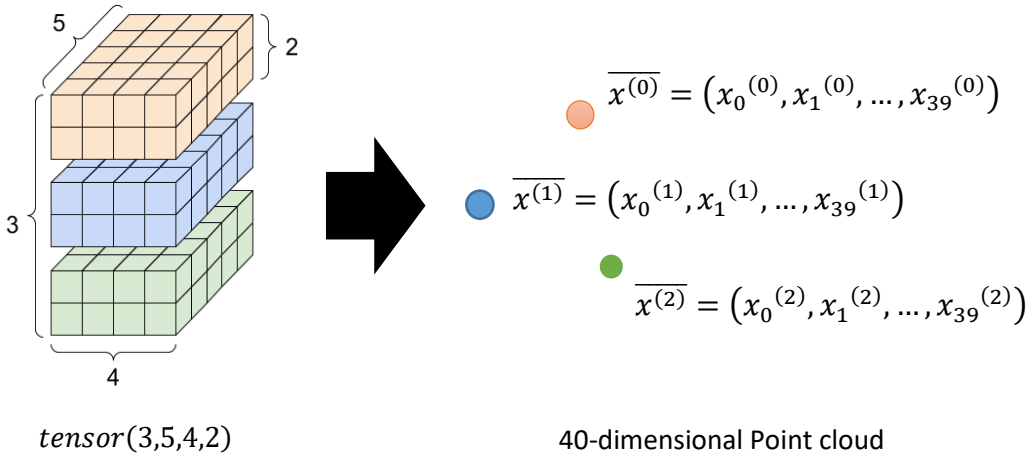


Figure 7.1: Generating a Point Cloud

Python implementation of the TDA techniques used in this research is found in **Appendix II: Topological Data Analysis**

7.5 Metric Calculator implementation

With the help of the GUDHI library, point cloud generated by the data collector module is converted to a persistence diagram. Each point of the persistence diagram (see Figure 7.2) is a triple (k, b, d) where k, b, d represents dimension, birth and death of a topological feature respectively.

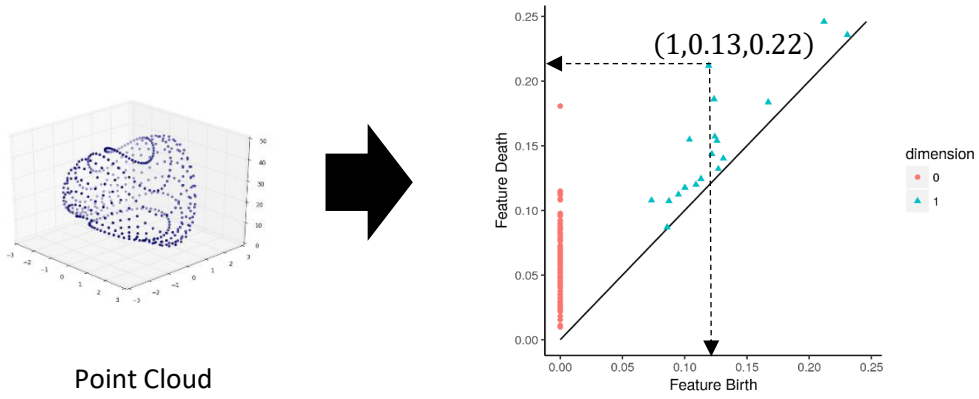


Figure 7.2: Generate persistence diagram

Python implementation of the metric calculator used in this research in given in **Appendix III: Topological Pruner**.

7.6 Evolutionary Algorithm Implementation

Initial population is generated by taking subset of the point cloud generated in by the data collector module. Each of these point clouds can be represented using a mask of the original point cloud. Each individual here is analogues to a possible prune in the neural network.

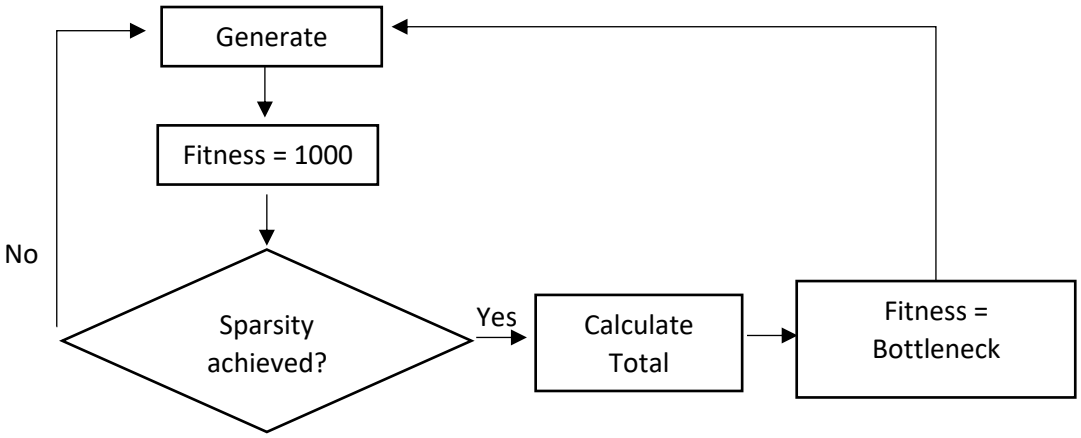


Figure 7.3: Fitness Function

Fitness function of this evolutionary algorithm has two parts namely bottleneck distance, which ensures the pruned network, is almost equal to the original model and number of parameters. If we only optimize using this part, best individual would be equal to original model as there is no other model similar to itself. To prevent this issue we add number of parameters based on the sparsity (see Figure 7.3).

Number of parameters required is calculated with the use of sparsity level. If a particular individual is having neurons than required, its fitness is kept high so that it will never be selected as the best individual. Once the best individual is selected, it is decoded to a mask that indicates which neurons need to be marked for deletion.

Generic algorithm configurations were kept as follows:

- population size = 10
- number of generations = 10
- crossover probability = 0.8
- mutation probability = 0.05
- elitism = True
- maximize fitness = False

Python implementation of the genetic algorithm used in this research in given in **Appendix I: Topological Genetic Algorithm**.

7.7 Sparsity Allocator Implementation

As the mask returned by the evolutionary module is not a tensor, it is first converted to a tensor. Then these masks are expanded to connector masks explained in 6.6.1. Finally, the connector masks is handed over to the speedup module for the actual pruning. Python implementation of the sparsity allocator used in this research in given in **Appendix III: Topological Pruner**.

7.8 System overview

Python implementation of the topological pruner is as shown in Figure 7.4. It is constructed by using the reusable modules data collector, metric calculator and sparsity allocator following component based software engineering principals.

```
class TopologicalPruner(BasicPruner):

    def reset_tools(self):
      if self.data_collector is None:
          self.data_collector = WeightDataCollector(self)
      else:
          self.data_collector.reset()
      if self.metrics_calculator is None:
          self.metrics_calculator = TopologyMetricsCalculator()
      if self.sparsity_allocator is None:
          self.sparsity_allocator = DirectSparsityAllocator(self, dim=0)
```

Figure 7.4: Implementation of Topological Pruner

Topological pruner being a subclass of Basic Pruner inherits all the compression related functionality. Therefore, the compression pipeline can be directly started by calling the compress method (see Figure 7.5).

```
config_list = [{
    'sparsity': 0.5,
    'op_types': ['Conv2d']
}]
pruner = TopologicalPruner(model, config_list)
_, masks = pruner.compress()
```

Figure 7.5: Compression pipeline

At the end of the compression pipeline, as shown in Figure 7.6, model speed up process is started. To prevent the loss of accuracy due to pruning, pruning model is trained until it gains its original accuracy before pruning.

```
pruner.show_pruned_weights()
pruner._unwrap_model()
ModelSpeedup(model, dummy_input=torch.rand([10, 3, 32, 32]).to(device), masks_file=masks).speedup_model()
```

Figure 7.6: Model Speedup process

7.9 Summary

The practical realization of the Topological Pruner was explicated here. Framework used for implementation was briefly explained. Next, the implementation of each module is described in detail with diagrams, code segments, configurations, data types, etc. The upcoming chapter describes how the novel pruner was evaluated against the desired prospects.

Chapter 8

8. Evaluation – A Quantitative Analysis

8.1 Introduction

In Chapter 7, we provided with the implementation of our novel pruner, 'Topological Pruner'. This chapter evaluates the implementation of the topological pruner. In doing so, we have structure the evaluation under several sub headings, namely, evaluation strategy under which an overview of experimental strategies were given, experiments setup, Standard comparison and recoverability comparison. This chapter concludes by assessing the convergence requirement of the novel pruner.

8.2 Evaluation Strategy

Standard comparison of pruners is done using floating-point operations per second (FLOPS) and number of parameter of the pruned model [39]. An overview of evaluation strategies used in this research is given below.

8.2.1 FLOPS

Inferencing from a convolutional neural network involves a high level of floating-point operations (FLOPS). If a model requires a considerable amount of FLOPs, then in cannot be used for applications with a limitation of computational power. Therefore, a lover level of FLOP rate is expected.

8.2.2 Number of parameters left

For most of the pruning techniques, number of connections being pruned is not an independent variable. Although, the sparsity level can impose restrictions on the number of parameters, it cannot determine the exact number of parameters to be pruned. Therefore, number of parameters left in the pruned network can act as an evaluation criterion. The lower the number of parameters left, better the pruning technique is.

8.2.3 Recoverability

Pruning techniques result in an architectural damage. Therefore, immediately after the pruning accuracy of the model is low. If the pruning technology is carefully define to reduce the damage, a model can regain the accuracy by training it for few epochs. Number of epochs and the highest accuracy it can achieve within a fixed number of epochs serves as two measurements for the recoverability.

8.2.4 Convergence of the network

Some pruners work well with pre-trained networks but fails to cope up with less trained networks. In the early stages of the training, connections between the neurons are not well established. Then the pruner criterion should be carefully selected.

8.3 Experimental Setup

NNI by Microsoft provide a standard setup for testing Pruners written according to their framework. Since topological pruner was written according to their standards, NNI experimental setup could be reused. CIFAR10 was used as the data set and VGG16 was

used as the model. VGG implantation used in this research is available at **Appendix IV: VGG Model Implementation**.

Table 8.1: Models used in Evaluation

Model	Accuracy
CIFAR10_VGG_38.40.pth	38.40%
CIFAR10_VGG_55.96.pth	55.96%
CIFAR10_VGG_75.41.pth	75.41%
CIFAR10_VGG_89.49.pth	89.49%
CIFAR10_VGG_90.03.pth	90.03%

Since each training session generates different models, a fixed set of pre-trained models with various levels of accuracy were generated and saved for the later used. Table 8.1 presents the accuracies of the pre-trained models being used.

Level Pruner, L1 Norm Pruner, L2 Norm Pruner were used in comparison with novel pruner. For this comparison, model is trained from scratch and saved for used across pruners.

Since the actual implementation of the algorithm requires a considerable amount of computational power as well as memory, following limitations were implemented for comparison.

I. Total bottleneck distance was calculated only considering maximum dimension as three.
II. Population size of the genetic algorithm was restricted to 10.
III. Number of generations of the genetic algorithm was restricted to five.

8.4 Standard Tests

FLOPS and the number of parameters were as shown in Table 8.2. Since each attempt result in different FLOPS and Number of Parameters average was considered. Figure 8.1 represents the same data in a Column chart for better visualization.

Table 8.2: FLOPS and Number of Parameters

Condition	FLOPS	Number of Parameters
Original Model	313.46M	14.98M
Level Pruner	307.29M	14.97M
L1 Norm Pruner	78.88M	3.81M
L2 Norm Pruner	78.88M	3.81M
Topological Pruner	76.56M	3.74M

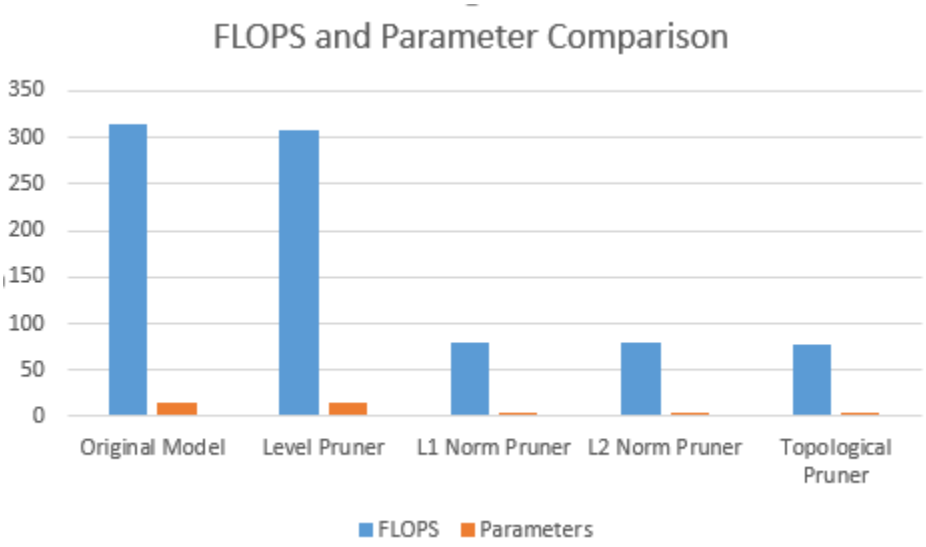


Figure 8.1: Standard Comparison

As shown in Figure 8.1, Topological Pruner has the lowest number of FLOP and Parameters.

8.5 Recoverability Test

Data shown in Table 8.3 was obtained by using the CIFAR10_VGG_89.49.pth model and CIFAR10_VGG_75.41.pth model.

Table 8.3: Recoverability Test Results

Model/ Pruner	CIFAR10_VGG_89.49.pth			CIFAR10_VGG_75.41.pth		
	Initial Accuracy after Pruning	Best accuracy within 10 epochs	Epochs needed to recover	Initial Accuracy after Pruning	Best accuracy within 10 epochs	Epochs needed to recover
Level	87.52%	89.84%	3	72.65%	79.85%	8
L1	83.96%	88.91%	8	70.18%	79.17%	10
L2	84.75%	88.57%	7	71.91%	75.30%	>10
Topological	81.2%	88.09%	6	74.05%	79.81%	6

As shown in the table, architectural damage for low accuracy models is relatively low for Topological Pruner. Although, the level pruner works relatively well, its FLOPS and Parameters are not good as discussed in 8.4. Therefore, epoch needed to recover when using topological pruner is relatively low with respect to L1 Norm Pruner and L2 Norm Pruner.

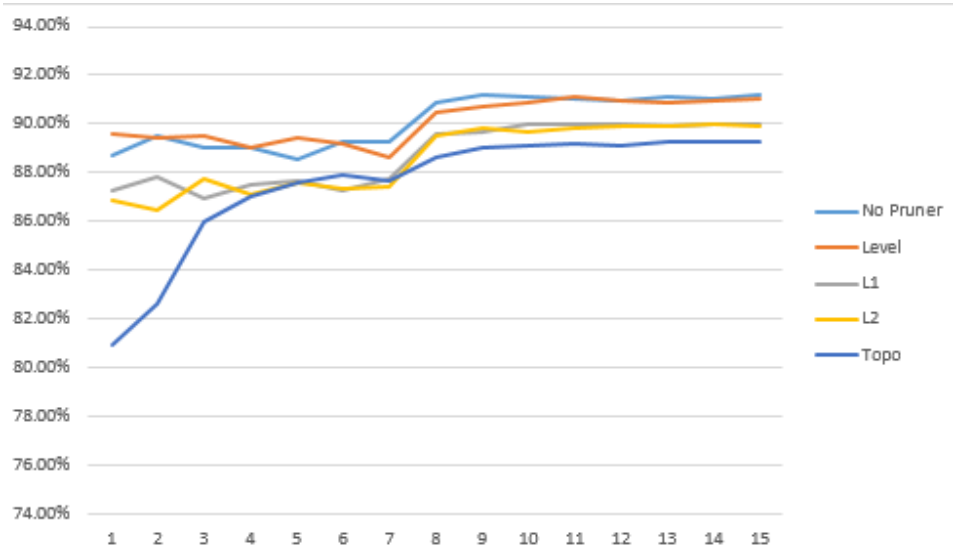


Figure 8.2: Recoverability Comparison

As shown in Figure 8.2, Initial damage done by the Topological pruner is high with respect to other three pruners. This damage is beneficial to low number of parameters in the pruned model. However, is clearly seen that the damage is been recovered in few epochs.

8.6 Convergence Test

As shown in Table 8.4, Topological pruner behavior depends on the sparsity level of the compressor and the accuracy of the model. For 'CIFAR10_VGG_55.96.pth' and 'CIFAR10_VGG_75.41.pth' it works well without much effect from the sparsity level. However, if the model accuracy is low as in 'CIFAR10_VGG_38.40.pth', for high sparsity levels Topological pruners fails to prune well. This is because, if the model is not converge, what topological pruner identifies as features is not correct. High values for 'best accuracy within 10 epochs' column in this case is actually due to training but not due to pruning.

Table 8.4: Convergence Test Results

Model	Sparsity	Epoch to recover	Best accuracy within 10 epochs
CIFAR10_VGG_38.40.pth	10%	2	73.20%
	40%	5	60.65%
	70%	6	52.39%
CIFAR10_VGG_55.96.pth	10%	2	61.56%
	40%	3	59.18%
	70%	3	57.11%
CIFAR10_VGG_75.41.pth	10%	2	80.12%
	40%	2	77.30%
	70%	4	76.01%
CIFAR10_VGG_89.49.pth	10%	4	89.84%
	40%	6	89.91%
	70%	>10	88.30%
CIFAR10_VGG_90.03.pth	10%	4	91.16%
	40%	8	90.20%
	70%	>10	89.29%

On the other hand, topological pruner does not work well with high accuracy models with high sparsity values either. This is because if the model is already generalized and a high sparsity is expected then, the topological pruner fails to find a mask with low bottleneck distance.

8.7 Summary

Evaluation of the novel pruning algorithm was done in this chapter. By doing so, its performance was tested using various criterion, namely FLOPS, number of parameters and recoverability. This chapter concludes by discussing, under which sparsity and accuracy levels, the novel pruner works well. In next chapter, a retrospect will be done on this research.

Chapter 9

9. Conclusion

9.1 Introduction

In this research, a novel pruner, which uses topological features, was introduced. To the best of our knowledge, this was the first attempt of using persistence homology for pruning. Unlike the previous literature, this pruner does not discriminate neurons based on any attribute of the neurons themselves. Instead, it consider the redundancy of neurons in a particular layer and mark them for deletion based on the topological features of the layer to which they belong. In the previous chapter, topological pruner was evaluated using some common metrics for pruning. This chapter will conclude the thesis by doing a retrospect with support of the previous chapter.

9.2 Achievements

The overall project process was aligned to achieve the objectives of this research mentioned under the introduction chapter. All the objectives were reached as expected by following the proper project process.

At the critical review of neural network pruning, early developments of the knowledge area break through and challenges were identified. Next issues in available solutions were also studied in depth by paving a way to the identification of research problem.

By achieving the third objective, a novel pruner was designed and implemented successfully. This novel pruner was named as ‘Topological Pruner’ as there was no other evidence of a topological feature based pruner. Without using the ‘Mapper’, the most common tool from TDA used for machine learning, Persistence homology and bottleneck distance were used in designing the novel pruner.

Finally, the last objective was achieved; Topological pruner was evaluated against other pruners. By doing so, it was identified that the Topological Pruner is capable of achieving high level of FLOPS and number of parameters while maintaining recoverability. This is an evidence that the bottleneck distance can in fact be used as a pruning criterion. In addition, architectural damage and under what conditions the topological pruner can overcome was discussed.

With the evaluation results, it can be concluded that the hypothesis 'architectural damage caused by neural network pruning can be explained and mitigated by Topological Data Analysis' is true. Therefore, the aim of the project, which is to develop a solution for architectural damage due to neural network pruning with the use of topological data analysis, is successfully achieved.

9.3 Limitations

Although Topological pruner works well with partially trained neural networks, its current configuration perform well with neither fully trained networks nor networks in early stages of learning. Therefore, its configurations need to be optimized and generalized before using it for practical use.

9.4 Future work

Even though the novel pruner shows experimental evidence for high recoverability, a proper mathematical proof need to be constructed. Few of the counter examples need to be identified for other pruners as well.

Current implementation of the Topological pruner is a one-time pruner. Further research need to be done to find out its usage as an iterative pruner.

Further, accuracy gain is usually done at the end of the pruning process by retraining the pruned for few interactions. However, training layer by layer can be beneficial for topological pruner.

9.5 Summary

By wrapping up the thesis, this chapter describes, to which extent the objectives were attained, the achievements and the limitations and the further works to be done. Few modifications and evaluations were identified before bring this pruner to the practical ground

References

[1] D. Blalock, J. J. G. Ortiz, J. Frankle, and J. Guttag, “What is the State of Neural Network Pruning?,” *ArXiv200303033 Cs Stat*, Mar. 2020, Accessed: Jun. 06, 2021. [Online]. Available: http://arxiv.org/abs/2003.03033

[2] P. Chou, “The Capacity of the Kanerva Associative Memory is Exponential,” in *Neural Information Processing Systems*, 1988. Accessed: Jun. 12, 2021. [Online]. Available: https://proceedings.neurips.cc/paper/1987/file/c81e728d9d4c2f636f067f89cc14862c-Paper.pdf

[3] M. C. Mozer and P. Smolensky, “Skeletonization: A Technique for Trimming the Fat from a Network via Relevance Assessment,” p. 9.

[4] A. See, M.-T. Luong, and C. D. Manning, “Compression of Neural Machine Translation Models via Pruning,” *ArXiv160609274 Cs*, Jun. 2016, Accessed: Jun. 13, 2021. [Online]. Available: http://arxiv.org/abs/1606.09274

[5] T.-W. Chin, C. Zhang, and D. Marculescu, “Layer-compensated Pruning for Resource-constrained Convolutional Neural Networks,” p. 13.

[6] T. Elsken, J. H. Metzen, and F. Hutter, “Neural Architecture Search: A Survey,” p. 21.

[7] J. Xu and D. W. C. Ho, “A new training and pruning algorithm based on node dependence and Jacobian rank deficiency,” *Neurocomputing*, vol. 70, no. 1–3, pp. 544–558, Dec. 2006, doi: 10.1016/j.neucom.2005.11.005.

[8] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, “A Survey of Model Compression and Acceleration for Deep Neural Networks,” *ArXiv171009282 Cs*, Jun. 2020, Accessed: Feb. 20, 2022. [Online]. Available: http://arxiv.org/abs/1710.09282

[9] M. M. Pasandi, M. Hajabdollahi, N. Karimi, and S. Samavi, “Modeling of Pruning Techniques for Deep Neural Networks Simplification,” p. 6.

[10] S. Han, J. Pool, J. Tran, and W. J. Dally, “Learning both Weights and Connections for Efficient Neural Networks,” *ArXiv150602626 Cs*, Oct. 2015, Accessed: Jun. 13, 2021. [Online]. Available: http://arxiv.org/abs/1506.02626

[11] Y. LeCun, J. Denker, and S. Solla, “Optimal Brain Damage,” in *Advances in Neural Information Processing Systems*, 1989, vol. 2. Accessed: Mar. 03, 2022. [Online]. Available: https://proceedings.neurips.cc/paper/1989/hash/6c9882bbac1c7093bd25041881277658-Abstract.html

[12] B. Hassibi, D. G. Stork, and G. J. Wolff, “Optimal Brain Surgeon and general network pruning,” in *IEEE International Conference on Neural Networks*, Mar. 1993, pp. 293–299 vol.1. doi: 10.1109/ICNN.1993.298572.

[13] B. Hassibi, D. Stork, and G. Wolff, “Optimal Brain Surgeon: Extensions and performance comparisons,” in *Advances in Neural Information Processing Systems*, 1993, vol. 6. Accessed: Mar. 03, 2022. [Online]. Available: https://proceedings.neurips.cc/paper/1993/hash/b056eb1587586b71e2da9acfe4fbd19e-Abstract.html

[14] G. Carlsson and R. B. Gabrielsson, “Topological Approaches to Deep Learning,” *ArXiv181101122 Cs Math Stat*, Nov. 2018, Accessed: Jun. 13, 2021. [Online]. Available: http://arxiv.org/abs/1811.01122
[15] R. Srinivasan and A. Chander, “Understanding Bias in Datasets using Topological Data Analysis,” p. 7.
[16] J. Lin, Y. Rao, J. Lu, and J. Zhou, “Runtime Neural Pruning,” p. 11.
[17] H. Liu, B. Xin, S. Mu, and Z. Zhu, “Pruning the deep neural network by similar function,” *J. Phys. Conf. Ser.*, vol. 1187, no. 4, p. 042006, Apr. 2019, doi: 10.1088/1742-6596/1187/4/042006.
[18] S.-K. Yeom, “Pruning by explaining: A novel criterion for deep neural network pruning,” *Pattern Recognit.*, p. 14, 2021.
[19] X. Dong, S. Chen, and S. Pan, “Learning to Prune Deep Neural Networks via Layer-wise Optimal Brain Surgeon,” p. 11.
[20] B. Hassibi, “Optimal Brain Surgeon.”
[21] D. Jean and Nice, “Introductory Remarks on Algebra, Topology and Analysis,” *Historia Mathematica*, vol. 2, pp. 537–548, 1975.
[22] L. Górniewicz, “Solving Equations by Topological Methods,” *Opuscula Mathematica*, vol. 25, 2005.
[23] J. Munkres, “Topological Spaces and Continuous Functions,” in *Topology*, 2nd edition., Upper Saddle River, NJ: Pearson College Div, 2000, p. 76.
[24] J. Munkres, “Closed Sets and Limit Points,” in *Topology*, 2nd edition., Upper Saddle River, NJ: Pearson College Div, 2000, p. 94.
[25] J. Munkres, “Connectedness and Compactness,” in *Topology*, 2nd edition., Upper Saddle River, NJ: Pearson College Div, 2000, p. 146.
[26] J. R. Munkres, “Homology Group of a Simplicial Complex,” in *Elements Of Algebraic Topology*, 1st edition., Boca Raton London New York: CRC Press, 1993, p. 2,3,7.
[27] J. R. Munkres, *Elements Of Algebraic Topology*, 1st edition. Boca Raton London New York: CRC Press, 1993.
[28] K. Childers, “Examples of persistent homology,” p. 9, 2017.
[29] “GUDHI: Bottleneck distance.” https://gudhi.inria.fr/doc/latest/group__bottleneck__distance.html (accessed Feb. 27, 2022).
[30] G. Singh, F. Mémoli, and G. Carlsson, “Topological Methods for the Analysis of High Dimensional Data Sets and 3D Object Recognition,” p. 11.
[31] E.-G. Talbi, “Common Concepts for Metaheuristics,” in *Metaheuristics: from design to implementation*, Hoboken, N.J: John Wiley & Sons, 2009, pp. 1–3.
[32] E.-G. Talbi, “Simulated Annealing,” in *Metaheuristics: from design to implementation*, Hoboken, N.J: John Wiley & Sons, 2009, pp. 124–127.
[33] E.-G. Talbi, “Evolutionary Algorithms,” in *Metaheuristics: from design to implementation*, Hoboken, N.J: John Wiley & Sons, 2009, pp. 200–220.
[34] S. Katoch, S. S. Chauhan, and V. Kumar, “A review on genetic algorithm: past, present, and future,” *Multimed. Tools Appl.*, vol. 80, no. 5, pp. 8091–8126, Feb. 2021, doi: 10.1007/s11042-020-10139-6.
[35] “PyTorch documentation — PyTorch 1.10 documentation.” https://pytorch.org/docs/stable/index.html (accessed Mar. 07, 2022).

[36] “Model Compression — An open source AutoML toolkit for neural architecture search, model compression and hyper-parameter tuning (NNI v2.6.1).” https://nni.readthedocs.io/en/stable/model_compression.html (accessed Mar. 07, 2022).
[37] “GUDHI Python modules documentation — gudhi documentation.” https://gudhi.inria.fr/python/latest/ (accessed Mar. 07, 2022).
[38] “pyeasyga — pyeasyga 0.3.1 documentation.” https://pyeasyga.readthedocs.io/en/latest/ (accessed Mar. 07, 2022).
[39] “Comparison of Filter Pruning Algorithms — An open source AutoML toolkit for neural architecture search, model compression and hyper-parameter tuning (NNI v2.6.1).” https://nni.readthedocs.io/en/stable/CommunitySharings/ModelCompressionComparison.html (accessed Mar. 13, 2022).
[40] “MUNKRES-ETA.pdf.” Accessed: Feb. 26, 2022. [Online]. Available: https://people.dm.unipi.it/benedett/MUNKRES-ETA.pdf
[41] “00298572.pdf.” Accessed: Mar. 03, 2022. [Online]. Available: https://www.ee.caltech.edu/Babak/pubs/conferences/00298572.pdf

Appendix I: Topological Genetic Algorithm

Python implementation of the topological genetic algorithm is as follows.

```
from pyeasyga.pyeasyga import GeneticAlgorithm
import tda

class TopologicalGeneticAlgorithm(GeneticAlgorithm):

  def fitness(self, individual, data):
    fitness = 1000
    if individual.count(1) <= len(individual) * (1 - self.sparsity) :
      pcl = tda.apply_mask(data, individual)
      stl = tda.generate_simplex_tree(pcl)
      max_dimension = 3 # pcl.shape[1]
      fitness = tda.calculate_bottleneck(self.simplex_tree, stl, max_dimension)
    return fitness

  def __init__(self, data,
               sparsity=0.0,
               population_size=10,
               generations=20,
               crossover_probability=0.8,
               mutation_probability=0.05,
               elitism=True):
    super(TopologicalGeneticAlgorithm, self).__init__(data,
                                                      population_size=population_size,
                                                      generations=generations,
                                                      crossover_probability=crossover_probability,
                                                      mutation_probability=mutation_probability,
                                                      elitism=elitism,
                                                      maximise_fitness=False)
    self.sparsity = sparsity
    self.fitness_function = self.fitness
    self.simplex_tree = tda.generate_simplex_tree(data)
```

Appendix II: Topological Data Analysis

Python implementation of the topological data analysis supportive methods is as follows.

```
import torch
import gudhi as gd

def generate_simplex_tree(point_cloud):
  rc = gd.RipsComplex(points=point_cloud)
  st = rc.create_simplex_tree(max_dimension=2)
  st.compute_persistence()
  return st

def apply_mask(point_cloud, mask):
  mask = torch.tensor(mask)
  indices = torch.nonzero(mask)
  i = torch.flatten(indices)
  return point_cloud[i]

def calculate_bottleneck(simplex_tree_1, simplex_tree_2, max_dimension):
  total_bottleneck_distance = 0
  for dimension in range(max_dimension):
    invl_1 = simplex_tree_1.persistence_intervals_in_dimension(dimension)
    invl_2 = simplex_tree_2.persistence_intervals_in_dimension(dimension)
    bd = gd.bottleneck_distance(invl_1, invl_2)
    total_bottleneck_distance += bd
  return total_bottleneck_distance
```

Appendix III: Topological Pruner

Python implementation of the Topological Pruner class and its supportive classes Topology Metric Calculator and Direct Sparsity Allocator are is as follows.

```python
from nni.algorithms.compression.v2.pytorch.pruning.basic_pruner import BasicPruner
from nni.algorithms.compression.v2.pytorch.pruning.tools.data_collector import WeightDataCollector
from nni.algorithms.compression.v2.pytorch.pruning.tools.base import MetricsCalculator
from nni.algorithms.compression.v2.pytorch.pruning.tools.base import SparsityAllocator

from typing import Dict
import torch
from torch import Tensor
from ga import TopologicalGeneticAlgorithm

class TopologyMetricsCalculator(MetricsCalculator):

  def calculate_metrics(self, data: Dict[str, Tensor]) -> Dict[str, Tensor]:
    metrics = {}
    for name, tensor in data.items():
      pc0 = tensor.reshape(tensor.shape[0], -1)
      ga = TopologicalGeneticAlgorithm(pc0, sparsity=0.5, population_size=10, generations=5)
      ga.run()
      best = ga.best_individual()
      print (name, best)
      metrics[name] = best[1]
    return metrics

class DirectSparsityAllocator(SparsityAllocator):

  def generate_sparsity(self, metrics: Dict[str, Tensor]) -> Dict[str, Dict[str, Tensor]]:
    masks = {}
    for name, wrapper in self.pruner.get_modules_wrapper().items():
      metric = metrics[name]
      mask = torch.tensor(metric, device=torch.device('cuda:0'))
      masks[name] = self._expand_mask(name, mask)
    return masks

class TopologicalPruner(BasicPruner):

    def reset_tools(self):
      if self.data_collector is None:
        self.data_collector = WeightDataCollector(self)
      else:
        self.data_collector.reset()
      if self.metrics_calculator is None:
        self.metrics_calculator = TopologyMetricsCalculator()
      if self.sparsity_allocator is None:
        self.sparsity_allocator = DirectSparsityAllocator(self, dim=0)
```

Appendix IV: VGG Model Implementation

Python implementation of the VGG model used for the experiment is as follows.

```
import math
import torch
import torch.nn as nn
import torch.nn.functional as F

defaultcfg = {
    11: [64, 'M', 128, 'M', 256, 256, 'M', 512, 512, 'M', 512, 512],
    13: [64, 64, 'M', 128, 128, 'M', 256, 256, 'M', 512, 512, 'M', 512, 512],
    16: [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 'M', 512, 512, 512, 'M', 512, 512, 512],
    19: [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 256, 'M', 512, 512, 512, 512, 'M', 512, 512, 512, 512],
}


class VGG(nn.Module):
    def __init__(self, depth=16):
        super(VGG, self).__init__()
        cfg = defaultcfg[depth]
        self.cfg = cfg
        self.feature = self.make_layers(cfg, True)
        num_classes = 10
        self.classifier = nn.Sequential(
            nn.Linear(cfg[-1], 512),
            nn.BatchNorm1d(512),
            nn.ReLU(inplace=True),
            nn.Linear(512, num_classes)
        )
        self._initialize_weights()

    def make_layers(self, cfg, batch_norm=False):
        layers = []
        in_channels = 3
        for v in cfg:
            if v == 'M':
                layers += [nn.MaxPool2d(kernel_size=2, stride=2)]
            else:
                conv2d = nn.Conv2d(in_channels, v, kernel_size=3, padding=1, bias=False)
                if batch_norm:
                    layers += [conv2d, nn.BatchNorm2d(v), nn.ReLU(inplace=True)]
                else:
                    layers += [conv2d, nn.ReLU(inplace=True)]
                in_channels = v
        return nn.Sequential(*layers)

    def forward(self, x):
        x = self.feature(x)
        x = nn.AvgPool2d(2)(x)
        x = x.view(x.size(0), -1)
        y = self.classifier(x)
        return y

    def _initialize_weights(self):
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                n = m.kernel_size[0] * m.kernel_size[1] * m.out_channels
                m.weight.data.normal_(0, math.sqrt(2. / n))
                if m.bias is not None:
                    m.bias.data.zero_()
            elif isinstance(m, nn.BatchNorm2d):
                m.weight.data.fill_(0.5)
                m.bias.data.zero_()
            elif isinstance(m, nn.Linear):
                m.weight.data.normal_(0, 0.01)
                m.bias.data.zero_()
```