# Topological Classification of SCOPe

Yechan Hong, Jan Segert, and Jianlin Cheng
University of Missouri, Columbia

April 22, 2019

## Abstract

**Motivation:** SCOPe 2.07 is a database of 276,231 protein domains that have been partitioned into varying folds according to their shape and function. Since a protein's fold reveals valuable information about it's shape and function, it is important to find a mapping between protein's and it's fold. There are existing techniques to map a protein's sequence into a fold [2] but none to map a protein's shape into a fold. We focus on the topological features of a protein to map it into a fold. We introduce several new techniques that accomplish this.
**Results:** We develop a 2D-convolutional neural network to classify any protein structure into one of 1232 folds. We extract two classes of input features for each protein's carbon alpha backbone: distance matrix and the persistent homology barcodes. Due to restrictions in our computing resources, we make sample every other point in the carbon alpha chain. We find that it does not lead to significant loss in accuracy. Using the distance matrix, we achieve an accuracy of 86% on the entire dataset.

We extract significant topological simplices of the protein by using persistent homology. We format the persistent homology data into various input features: persistence images [1], simplex distance map, and simplex grouping. With persistence images of 100x100 resolution, we achieve an accuracy of 62% on SCOP 1.55. With simplex distance maps of 100x100 resolution, we achieve an accuracy of 70%. With simplex groupings, we achieve an accuracy of :TODO%.

## 1 Introduction

Add introduction in literature and background information here. Future work to be added here. Discussion of topological data analysis will also be added here.

## 2 Material

### 2.1 Datasets

The SCOP database is a database of proteins organized into hierarchical classes

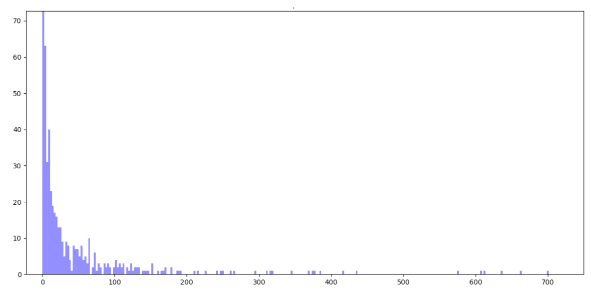

Figure 1: Frequency of classes

based on their shape and function. There are 4 levels of the hierarchy (top down): Class, Folds, Superfamily, Family. We will be primarily concerned with the Fold.

Two different versions of the dataset were used, SCOP 1.55 and SCOPe 2.07. SCOP 1.55 is a smaller dataset which is a subset of SCOPe 2.07, a larger and more recent dataset. SCOP 1.55 [Ref: Biblio] and SCOPe 2.07 [Ref: Biblio] were downloaded from the Berkeley repository as tar files and unpacked. For each of the datasets, index files [Ref:Biblo] are provided.

### 2.1.1 Small Dataset SCOP 1.55

SCOP 1.55 is a dataset of 31,474 proteins that have been organized into 7 Classes, 605 Folds, 947 Superfamilies, and 1557 Families. The dataset was released and updated till 2001. This dataset is a subset of the SCOPe 2.07 dataset.

We inspect the distribution of the proteins across the different folds. We see that most of the folds do not have a lot of proteins. The median number of proteins per fold was 10 and the histogram [Ref: Figure] show that most of our proteins have less than 50 proteins per fold. This would likely impact our learning, since there are not too many examples for the protein to learn.

We split up 70% of the dataset for training, 15% for validation and 15% for testing. We adjust the sampling of the validation and testing so that a wide range of folds are represented.

New methods were tested on the small dataset. This allowed us to quickly prototype and optimize our methods to run on the larger dataset.

### 2.1.2 Large Dataset SCOPe 2.07

SCOPe 2.07 is a database of 276,231 proteins that have been organized into 7 Classes, 1232 Folds, 2026 Superfamilies, and 4919 Families. The dataset was released and updated till 2017. This dataset contains and is about 9 times larger than the SCOP 1.55 dataset.

[Fig: Freq of data across folds]

We inspect the distribution of the proteins across the different folds. We see that there are much more proteins per fold.

Similar to the small dataset, we split up 70% of the dataset for training, 15% for validation and 15% for testing. We adjust the sampling of the validation and testing so that a wide range of folds are represented.

## 2.2 Third Party Tools

A number of third party tools were used as part of the research. ProDy's (Protein Dynamics & Sequence Analysis) python package was used to extract the backbone structure from each protein's PDB file. NumPy package was used for matrix and statistical operations. Matplotlib package was used to generate histograms and 3D plots of the proteins and the barcodes. Pillow was used

| | |
|---|---|
| CPU | Intel Core i7-4650U |
| GPU (CUDA-enabled) | None |
| RAM | 8GB |
| Storage | 128GB SSD |
| OS | MacOSx |

Table 1: Personal Latop

| | |
|---|---|
| CPU | E5-2630 6-Core |
| GPU (CUDA-enabled) | Nvidia GTX 1060 6GB |
| RAM | 16GB |
| Storage | 500GB SSD |
| OS | Ubuntu 16.02 |

Table 2: Personal Workstation

to generate the 2D images of the distance matrix and the barcode images. Keras and Tensorflow were used to setup and train the convolutional neural network. CPickle package was used to serialize and deserialize processed data.

### 2.3 Computing Resources

Initially the research started on a personal laptop without a GPU and very limited storage space [Ref: Table]. Since the unpacked data of SCOPe 2.07 took up around 40GB and the laptop ran out of storage space, the data was stored on a flash drive.

The lack of a GPU made the training very slow. Even relatively simple methods on the smaller, SCOP 1.55, dataset took around 40 hours. Furthermore, the small storage space made it difficult to unpack and save the processed data on the larger SCOPe 2.07 dataset. These factors significantly slowed the progress of the research.

Due to the limitations of the personal laptop, a personal workstation was purchased at $500 on Winter of 2018 [Ref: Table]. Although it has modest computing power relative to industry standards, the purchased machine led to significant increases in speed and efficiency of the research. The most important changes in computer resources were the GPU, which led to increases in training speed, and the increased storage, which made it possible to work on the larger SCOPe 2.07 dataset. It is also worth nothing that Ubuntu has better support than MacOSx for running CUDA. Most of the optimized methods took around 16 hours at most to complete on the large SCOPe 2.07 database, which is 8 times larger than the smaller SCOP 1.55.

## 3 Methods

### 3.1 Background Information: Proteins

A protein is an sequence of amino acids, 22 compounds which link into a protein chain. The interaction between amino acids and the surrounding environment determine the how the protein folds into its structure.

For a protein that we are tasked to classify, we are provided with many information: Protein sequence and the sequential coordinates of every atom on our protein. Since we are primarily interested in the topological features of our data, we characterize the protein's shape with the protein backbone. The protein's backbone is constructed with a sequence of points, where each point represents each amino acid in a 3D space. The point representation of each amino acid is determined by the algorithm 'parsePDB' in the ProDy package.

Since we will be dealing primarily with the protein's backbone, we introduce the following notation.

**Definition 3.1.** A protein, '$P$', with sequence of N amino acids will be a called a protein with length N. It's backbone will be denoted as a sequence of 3D coordinate points '$\{P_i\}_{i=1}^N$' where '$P_i \in \mathbb{R}^3$'

For each coordinate point '$P_i$', the x-coordinate is referred as '$P_i(1)$', y-coordinate as '$P_i(2)$', and z-coordinate as '$P_i(3)$'.

We extract two distinguishing types of input features from the protein's backbone chain: Distance matrix and Persistence Homology. These two features are very robust. They are both rotation and translation invariant, meaning that the features remain the same even if the protein is rotated or moved. They are also very stable: minor changes to the data does not create a significant variation in the feature.

## 3.2 Backbone Chain

Each the dataset of proteins are stored as PDB (Protein Data Bank) files, which describe the shape of the 3D protein. The dataset tar [Ref] files unpack into main directory, pdbstyle-1.55 and pdbstyle-2.07 for SCOP 1.55 and SCOPe 2.07 respectively. Each PDB file is stored in a subdirectory under the main directory. The name of the subdirectory is determined by the protein's name.

The index files for SCOP 1.55 and SCOPe 2.07, 'dir.cla.scop.1.55.txt' and 'dir.cla.scope.2.07-stable.txt', provide important information for each protein in our database [Ref: index example].

| | |
|---|---|
| Index Line | d1dlwa_ 1dlw A: a.1.1. |
| Name | d1dlwa_ |
| Class.Folds.Superfamily.Family | a.1.1.1 |

Table 3: Index Entry. The PDB file for this protein found under the subdirectory 'dl' as 'd1dlwa_.ent'

For each protein, protein backbone chain is parsed from the PDB file using the 'parsePDB' function in the ProDy package. The extracted coordinates of the protein backbone is saved as a list.

The class and the fold number of the protein uniquely identifies a protein's fold. We create a one to one mapping between class-fold to the integers. These integers are the labels of our protein-fold classification problem.

Because all of the protein coordinates do not fit on the RAM, they are saved in batches of 10000. In each batch, the protein coordinates and the fold labels are saved into a dictionary under the keys b'x' and b'y' respectively.

## 3.3 Distance Matrix

With the points in the protein backbone, we construct a distance matrix of the distances between the points. The unit of the distances Ångströms, the units of provided in the protein PDB files. [Protein Data Bank]. We used the Euclidean Distance for the distances between the points.

**Definition 3.2.** For a protein $P$ with length N, we denote it's distance as matrix $M_P$. We construct it as follows. $M_P :=$

$[M_{ij}]$ where

$$M_{ij} = EuclideanDistance(P_i, P_j) = \sqrt{P_i(1) - P_j(1))^2 + (P_i(2) - P_j(2))^2 + (P_i(3) - P_j(3))^2}$$

**Remark**
We note the following for any distance matrix $M_P$.

- $M_{ii} = 0$
- $M_{ij} = M_{ji}$
- The intersection of the ith row and the jth column corresponds to $M_{ij}$, the distance between the ith and the jth point.

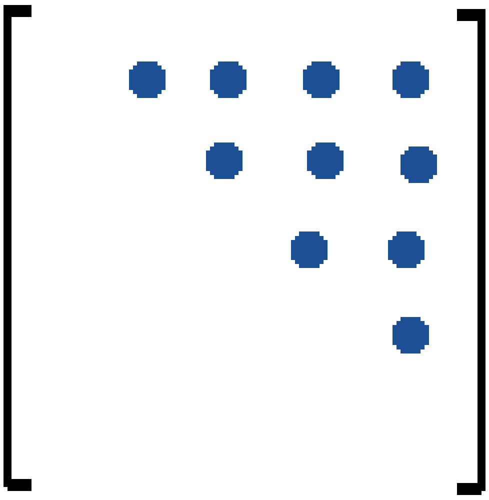

Figure 2:

We see that the distance matrix is symmetric [Ref: Remark] since the distance between the ith and the jth point is the same as the distance between the jth point and the ith point. Because of this, we only need to compute the distance between these pairs of points once. Also, the distance between a point to itself is zero. So the pairs of ith and jth's points we need to compute the distances for lie in the upper triangular region of the distance matrix. This region is consists of $\{M_{ij} | i < j\}$ [Ref:]. Computing the distance matrix in this fashion divides the computation time by about half. Because the distance matrices of the entire dataset are larger than the size of the RAM, the data is split up into 1000 sized batches.

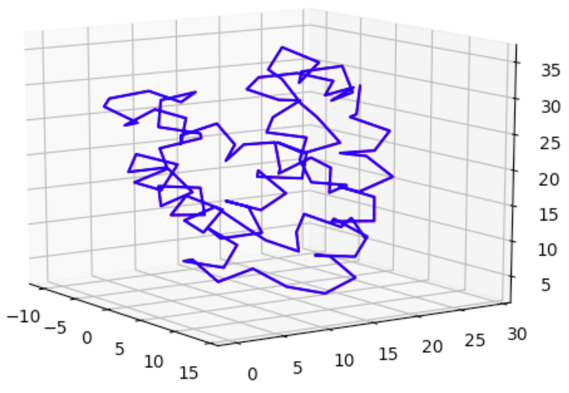

Figure 3:

We inspect the topological structure of our protein '1ux8'. It is a protein of length 118. This protein has some spiral structures (alpha helix). These spiral structures sometimes lie in close promixity in parallel or anti parallel direction (beta sheet). These types of structures (secondar structures) are known in molecular biology to be important features of a protein's shape.

We analyze the distance matrix to see if important structural features are represented in the matrix. To help with visualization, the distance matrix is mapped to an image of equal size, where closest distances

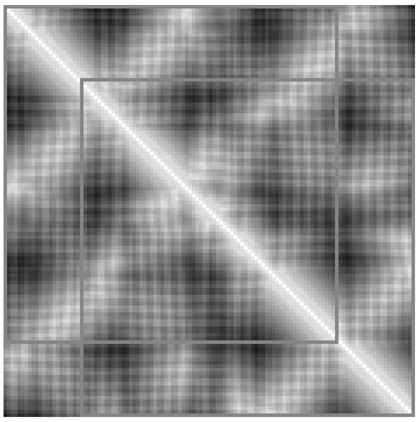

Figure 4:

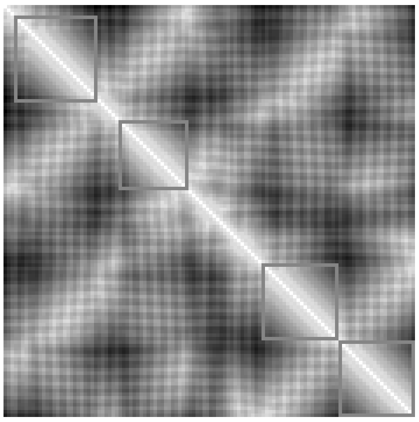

Figure 5:

appear in white and furthest distances appear in black. Distances in between take a gray hue with the intensity based on its value.

**Feature Protein Backbone:** Along the diagonal line of the image, the distance matrix is completely white. This is because the distance between a point to itself is zero [Ref: Remark]. We note that this diagonal white line uniquely identifies the protein backbone chain (Since the distance between. Having a clear representation of the protein backbone is important because it is a central structure that other features can be spatially oriented around. **Feature Alpha Helix:** We note thick white regions running parallel along the diagonal line of the image [Ref: Image]. These regions indicate that at a given point in the chain, it is in close proximity to the nearby neighbors [Ref Diagram]. We also note that in for a point in an Alpha Helix, it is also in close proximity to it's nearby neighbors. In our example, these four thick white regions correspond the the four helix structures on our protein.

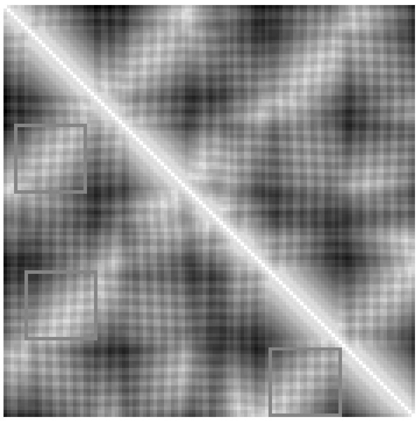

Figure 6:

**Definition 3.3.** A helix is composed of points in the protein backbone. Suppose $H$ is the set of indices of these points along the length of the protein. We call the row belonging to this helix as the collection of rows, $R_i$, of the distance matrix such that the rows contain elements of the helix. $\{R_i | i \in H\}$. Similarly we define the column belonging to the helix as $\{C_i | i \in H\}$.

**Feature Beta Sheet:** We note patches of thick white regions in the intersection of the rows belonging to a helix and the columns belonging to another helix [Ref Diagram].

This indicates that the points of the two helices, and hence the two helices, are in close proximity. In particular, the regions are close sequentially: the ith point in helix A is close to the jth point in helix B and the i+1 th point in helix A is close to the j-1 th point in helix B. This sequential relationship describes a Anti-parallel Beta Sheets. For Parallel Beta Sheets, the i+1 th point would be close to the j+1 th point. In our example, the 3 pairs of Beta-sheets formed the 4 helices are represented in our distance matrix.

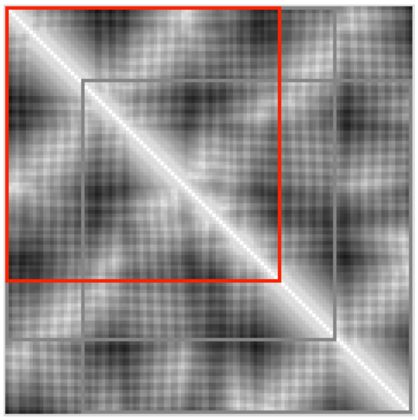

Figure 7:

### 3.3.1 Cropped Distance Matrix

Some proteins have a length of 600, making the distance matrix have a size of 600x600. However, due limitations on the GPU memory, it is not possible to construct a convolutional network with our input being 600x600. We would also like to crop the distance matrix such that the central backbone of the protein runs through the diagonal of our matrix. To crop and preserve the diagonal protein backbone, we take a 100x100 window and crop our matrix by shifting row and columns at the same time by 50 indices.

Because of the limitations set by the windowed distance matrix, a point in the backbone can only see information about, on average, half of the window size forwards and backwards. This limitation affects the cropped matrix's ability to detect longer range contact information, which can be critical in determining the protein's overall shape. In our example [Ref], the cropped distance matrix would not see information about the proximity between the first alpha helix and the third alpha helix because they are too far away in the backbone index.

Cropping the dataset increases the number of examples in the dataset. Because we don't want our training data to have similarities, we make sure that the training, validation, and testing groups do not share cropped matrices from the same protein.

### 3.3.2 Sparse Distance Matrix

In our example, we see that cropping the distance matrix diminishes it's ability to represent long range contact information. We develop an alternative approach to reduce the size of the distance matrix while preserving it's ability to represent long range contact information.

Given, a protein backbone with length N, $\{P_i\}_{i=1}^{N}$, we sample every other point, $\{P_i | \text{i is odd}\}$, to create a sparse protein backbone. We graph the sparse protein backbone in 3D space [Ref ] and compare it to the original protein backbone [Ref: Original Chain]. In comparison, we see that the general structure of the protein and its features are diminished but preserved.

From the sparse protein backbone, we construct a sparse distance matrix in the

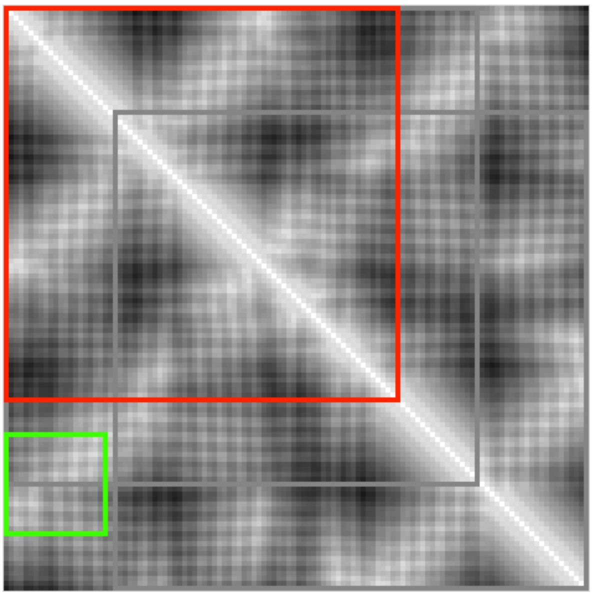
Figure 8:

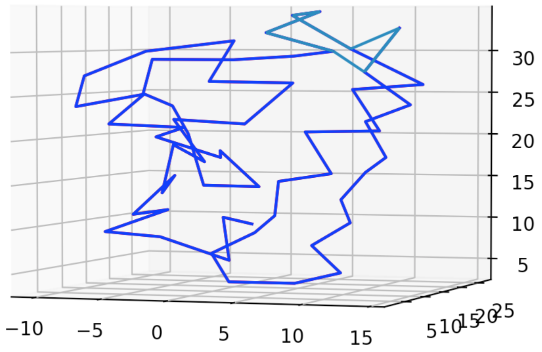

Figure 9:

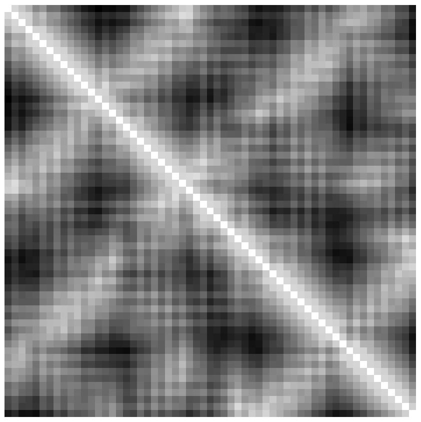
Figure 10:

same fashion as a regular distance matrix. In example, we compare the sparse distance matrix [Ref:] and the unmodified distance matrix [Ref:] and see if the backbone chain, alpha helix and beta sheet features are preserved in the distance matrix. First, we see that the diagonal backbone is preserved in the matrix. Second, we do see that the alpha helix features are preserved, even though it is less clearly defined. Finally, we see that the beta sheet features are preserved as well.

For the protein of length 600, the sparse distance matrix would reduce the matrix's size from 600x600 to 300x300. This is still too big for due to the limitations on the GPU memory. So we crop the sparse distance matrix in the same manner as a regular distance matrix. We also take care such that the training, validation, and testing groups do not share cropped matrices from the same protein.

### 3.3.3 Convolutional Network Model

The features from each of these subsections were fed into a a 2D-convolutional network. The architecture of the 2D-convolutional network for mapping protein structure to folds contains, in order, 32 3x3 convolutional layers, 64 3x3 convolutional layers, 2x2 max pooling, 128 dense layer, and the output layer.

This is a relatively simple network. However, for our task of classifying quarter million of proteins, it performs exceptionally well.

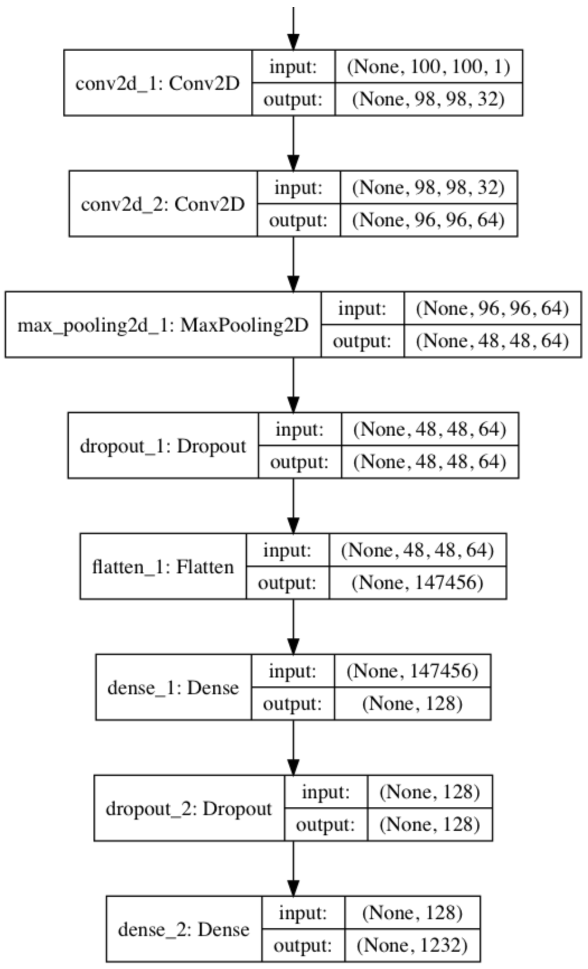
conv2d_1: Conv2D
input: (None, 100, 100, 1)
output: (None, 98, 98, 32)
conv2d_2: Conv2D
input: (None, 98, 98, 32)
output: (None, 96, 96, 64)
max_pooling2d_1: MaxPooling2D
input: (None, 96, 96, 64)
output: (None, 48, 48, 64)
dropout_1: Dropout
input: (None, 48, 48, 64)
output: (None, 48, 48, 64)
flatten_1: Flatten
input: (None, 48, 48, 64)
output: (None, 147456)
dense_1: Dense
input: (None, 147456)
output: (None, 128)
dropout_2: Dropout
input: (None, 128)
output: (None, 128)
dense_2: Dense
input: (None, 128)
output: (None, 1232)

Figure 11: The architecture of a 2D deep convolutional network for fold classification.

There are a lot of benefits for a simple network performing well. First, it ensures that the model is not overfitting the data. Second, it is much more feasible to inspect the network to understand how it is learning.

## 3.4 Topological Data Analysis

**Persistent Homology:** Topological data analysis is applied to the points in the protein backbone to produce persistent barcodes. The barcodes indicate when simplexes are formed and are destroyed. Significant topological features of the data are represented by these barcodes.

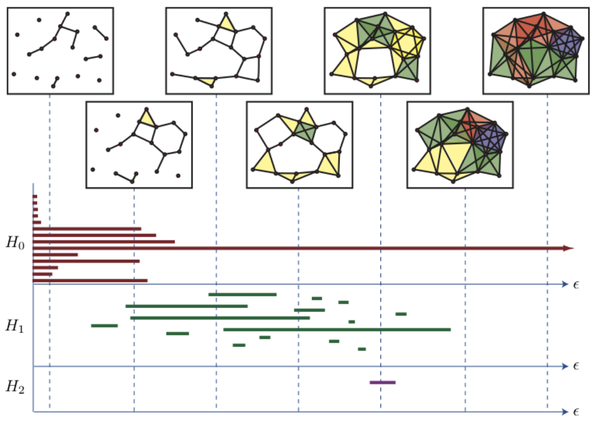

Figure 12: Topological data analysis on a set point cloud

### 3.4.1 Mathematical Background

**Definition 3.4.** K-Simplex is the convex hull of the standard basis of $\mathbb{R}^{k+1}$

**Definition 3.5.** Let $\sigma$ be a K-Simplex. The vertex of $\sigma$ is the points. The face of $\sigma$ are the simplices formed by subset of the vertices of $\sigma$. 2-face is a face of $\sigma$ with $2+1$ vertices.

**Definition 3.6.** Simplicial complex is a topological space composed of set of simplices, $\sum$, such that - Any face of a simplex $\sigma \in \sum$ is in $\sum$. - The intersection of any two simplices in $\sum$ is in $\sum$

### 3.4.2 Persistence Homology

We inspect the persistence homology of our protein '1ux8'. It is a protein of length 118.

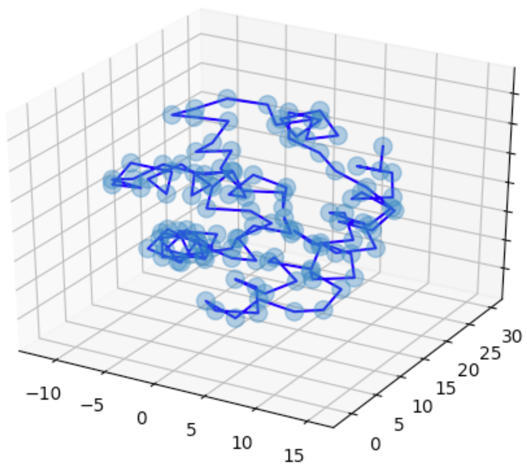

Figure 13: TDA 2b7h detecting alpha helices

Here is our protein chain. At each point, we search the radius around the point to find nearby points.

The red edges indicate that the two points are less than 3.5 distance away. Making them connected. As we increase the radius, we connect more and more points.

The connections between these points form polygons (simplexes). After a certain point we want to cut off our search radius because we have extracted all the notable features from our structure. (If we keep increasing the search radius then all the points will be connected to each other)

A polygon is filled in when the inner polygons of the larger polygons are filled in. We note that triangles are filled as soon as they are formed but larger structures take more time to get filled in. Persistent homology tells us when (the radius) at which these polygons are created and filled in.

Below we have two examples of the polygons.

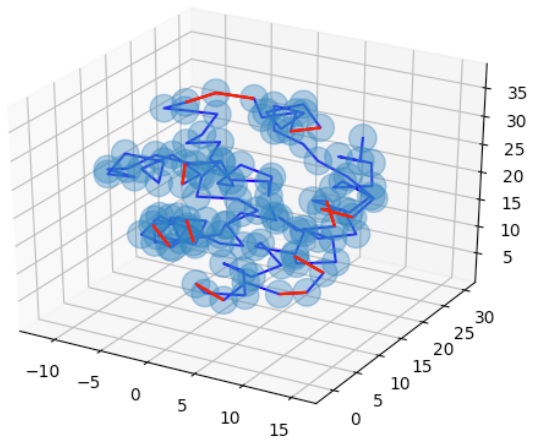

Figure 14: TDA 2b7h detecting alpha helices

### 3.4.3 Sparse Persistence Homology

### 3.4.4 Backbone Aware Persistence Homology

### 3.4.5 Persistence Images

We format the barcode data into three separate input features: persistent images, simplex distance map, and simplex grouping
**Persistent Images:** The barcodes for each simplex are mapped into a 2-dimensional vector space with the first coordinate being the birth and the second coordinate being the persistence of the simplex. The plot cropped and converted to a blurred image at a set resolution. The range of the first and second coordinates to crop the plot were derived from the distribution of the data.

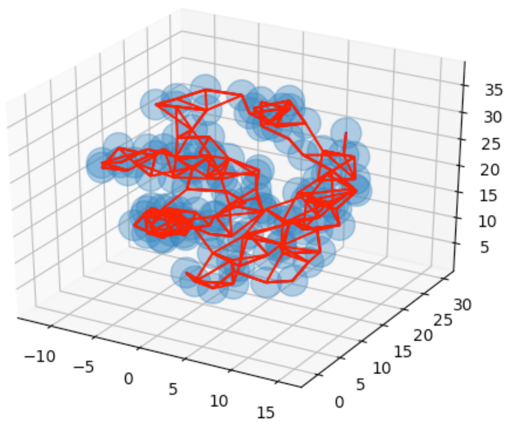
Figure 15: TDA 2b7h detecting alpha helices

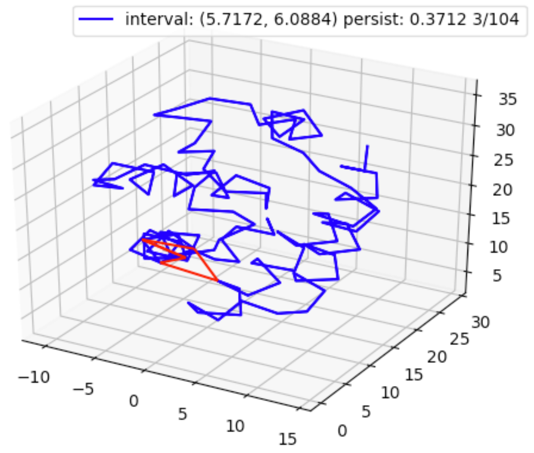

Figure 16: TDA 2b7h detecting alpha helices

## 4 Results

### 4.1 Distance Matrix Algorithm

The distance matrix as the input feature worked pretty well for classifying the data. The best method, subsampling every other point with a window size of 50x50 gives an accuracy of 96%

The other two methods perform pretty well, at around 86%. The subsampling method probably works best because it gives the neural network model to view the long range dependencies. The smaller window size of 50x50 performs much better than the 100x100 window for the subsampling methods.

## Test

## References

[1] Henry Adams, Tegan Emerson, and Michael Kirby. 2017. *Persistence Images: A Stable Vector Representation of Persistent Homology.* Journal of Machine Learning Research 18 (2017) 1-35

[2] Jie Hou, Badri Adhikari and Jianlin Cheng. 2018. *DeepSF: deep convolutional neural network for mapping protein sequences to folds* Bioinformatics, 34(8), 2018, 1295–1303

[3] Katsuro Sakai. 2010. *Simplicial Homology — A Short Course* Institute of Mathematics University of Tsukuba

[4] Fox NK, Brenner SE, Chandonia JM. 2014. *SCOPe: Structural Classification of Proteins—extended, integrating SCOP and ASTRAL data and*

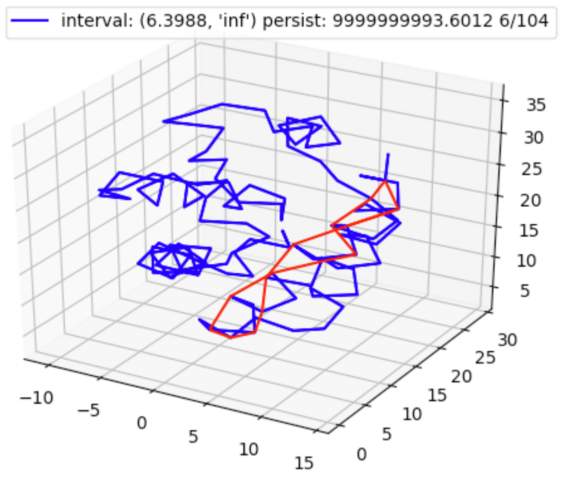

Figure 17: TDA 2b7h detecting alpha helices

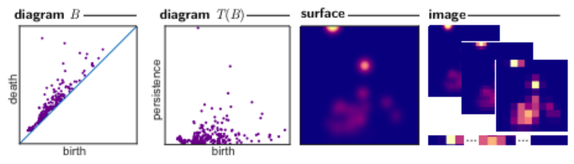

Figure 18: Persistence Images

*classification of new structures.* Nucleic Acids Research 42:D304-309. doi: 10.1093/nar/gkt1240

[5] Protein Data Bank Guide *Protein Data Bank Changes Guide New Changes in Version 3.20* Nucleic Acids Research 42:D304-309. doi: 10.1093/nar/gkt1240 https://cdn.rcsb.org/wwpdb/docs/docum format/changesv3.20.pdf

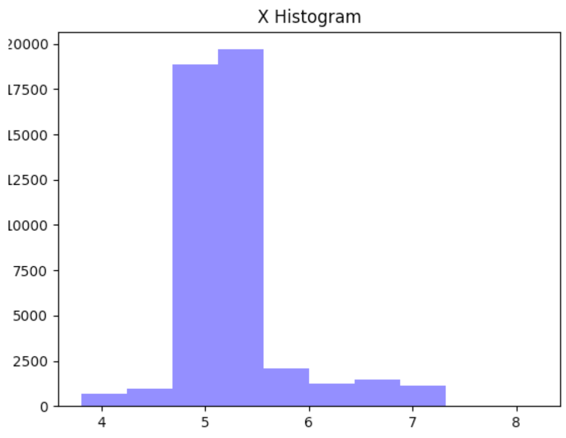

Figure 19: x range

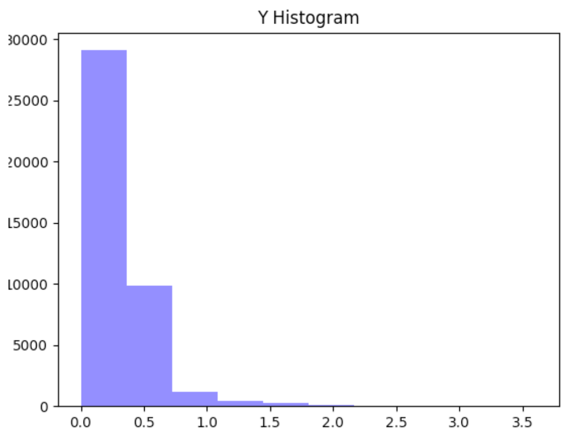

Figure 20: y range

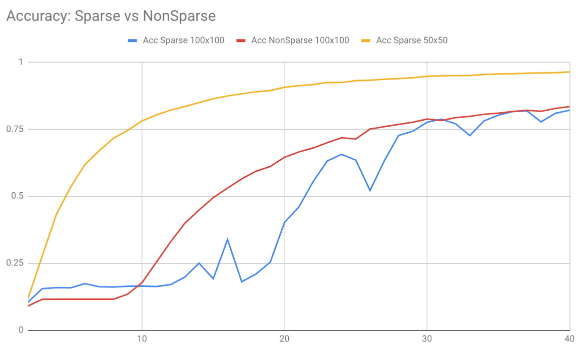

Figure 21:

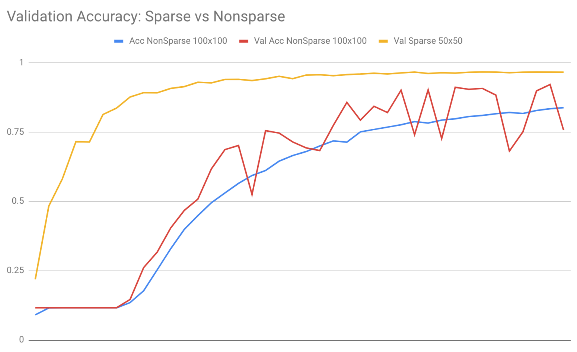
Validation Accuracy: Sparse vs Nonsparse
Acc NonSparse 100x100
Val Acc NonSparse 100x100
Val Sparse 50x50
1
0.75
0.5
0.25
0

Figure 22: