# PRO3DCNN: Convolutional Neural Network for Mapping Protein Structure into Folds

---

A Thesis presented to the Faculty of the Graduate

School University of Missouri Columbia

---

In Partial Fulfillment

of the Requirement for the Degree

**Master of Science**

---

By

**Yechan Hong**

Dr. Jianlin Cheng, Thesis Supervisor

May 2019

The Undersigned, appointed by the Dean of the Graduate School, have examined the thesis entitled.

# PRO3DCNN: Convolutional Neural Network for Mapping Protein Structure into Folds

Presented by Yechan Hong

A candidate for the degree of Master of Science

And hereby certify that in their opinion it is worthy of acceptance.

---

Professor JianLin, Cheng

---

Professor Toni, Kazic

---

Professor Jan, Segert

## Acknowledgements

I would like to take this section to thank those who have been tremendous in their aid to the research.

Dr. JianLin Cheng, who suggested the research idea and gave me numerous encouragement and wisdom when I faced issues that I thought I could not overcome.

Dr. Jan Segert, who graciously and patiently helped me to understand the mathematics and also enthusiastically suggested new areas of improvement on the research.

The faculty in the Computer Science and the Mathematics Department, from whom I have learned many concepts that helped me to become a strong thinker.

Family and friends and Casey, whom have supported me when I was down or worried.

And to God, who watches over me, guiding each of my steps.

# Contents

# PRO3DCNN: Convolutional Neural Network for Mapping Protein Structure into Folds

Yechan Hong, Jan Segert, and Jianlin Cheng

University of Missouri, Columbia

May 8, 2019

## Abstract

**Motivation:** SCOPe 2.07 is a dataset 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 proteins and their folds. 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 for the entire SCOPe 2.07 dataset. 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: distance matrix and persistent homology images. 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 90% on the entire dataset. With persistence images of 100x100 resolution, we achieve an accuracy of 54% on SCOP 1.55.

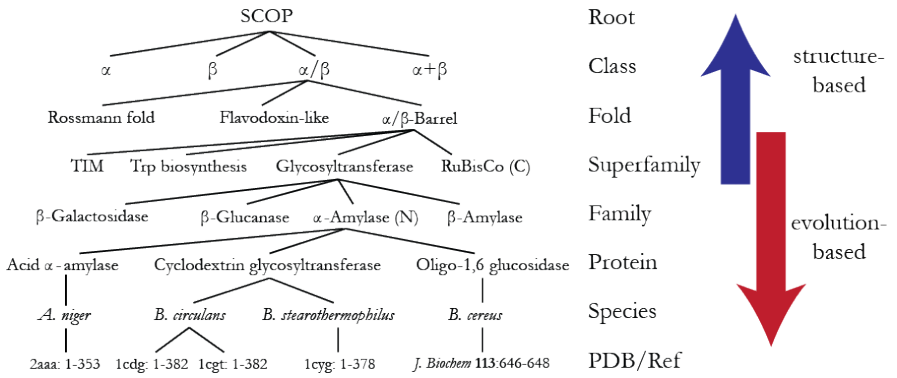



Figure 1: A partial view of the SCOP hierarchy.

# 1 Introduction

Structural Classification of Proteins (SCOP) is a database of protein structural relationships. The proteins are placed in a hierarchy in relative to their to structural and evolutionary relations [4]. SCOP was initially a manually curated ordering of proteins [4]. However, as many numerous proteins continue to be discovered at a rapid pace, a need for an automatic method of classification became necessary.

Sequence based classification using deep convolutional neural networks like DeepSF show that it is possible to predict protein fold with an accuracy up to 75.3% [2].

Structural Classification of Proteins - extended (SCOPe), extends the original SCOP database by using these types of rigorously validated automated methods to classify newly discovered proteins [4].

Similar to taxonomy, SCOP was created as a hierarchy with a basis in evolutionary relationships. Species, protein, family, and superfamily, the lower levels of the SCOP hierarchy represent the evolution-based relationships between the proteins. SCOP characterizes each of these levels with the following description.

- **Species** representing a distinct protein sequence and its naturally occurring or artificially created variants.

- **Protein** grouping together similar sequences of essentially the same functions that either originate from different biological species or represent dif-

ferent isoforms within the same species

- **Family** containing proteins with similar sequences but typically distinct functions
- **Superfamily** bridging together protein families with common functional and structural features inferred to be from a common evolutionary ancestor.

Fold and class, the higher levels in our hierarchy are grouped based on the similarity of structure and not necessarily evolutionary similarities [4].

- **Folds** grouping structurally similar superfamilies.
- **Classes** based mainly on secondary structure content and organization.

The highest level of the hierarchy, class, consists of 7 categories. Alpha proteins (a), beta proteins (b), Alpha/Beta proteins (c), Alpha+Beta proteins (d), multi-domain proteins (e), membrane and cell surface proteins and peptides (f), and small proteins (g). The original SCOP publication clarifies the members of class a-e [5]. Class (f) and (g)'s name are descriptive of their members.

- **Alpha proteins** For proteins whose structure is essentially formed by a-helices.
- **Beta proteins** For those whose structure is essentially formed by b-sheets.
- **Alpha/Beta proteins** For proteins with a-helices and b-strands that are largely interspersed.
- **Alpha+Beta proteins** For those in which a-helices and b-strands are largely segregated.
- **Multi-domain** For those with domains of different fold and for which no homologues are known at present.

Many works have been done on using the protein sequences and their evolutionary relationships with each other to predict the protein hierarchy. This is primarily due to the lack of structural information for newly discovered proteins.

In this paper, we shift the focus from the protein's sequence to the topological structure to predict the protein's fold and class. We expect the shift in focus to be quite successful in classifying the protein class and fold since these levels have been curated based on the similarity of structures between the proteins.

Since each fold belongs to only one class, predicting the protein fold also predicts the protein's class. Thus, we narrow the focus of our problem to predicting a protein's fold.

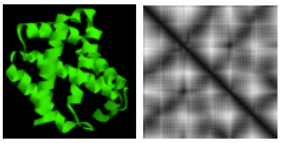
We introduce two methods for classifying protein fold. First, we use a distance matrix as an input matrix to a convolutional neural network. Second, we use persistent homology to extract topological features of our data and plot these features as an input matrix to a convolutional neural network.

**Distance Matrix**

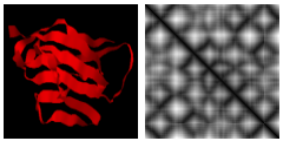
Each protein has a backbone structure that is formed by series of connecting points. The backbone structure of the protein is characteristic of the protein's general shape. A distance matrix of the protein's backbone is created, where the rth row and cth column of the matrix gives the distance between the rth point and the cth point of the protein backbone. Then this matrix is used as an input to our convolutional neural network.

There has been existing work (Fast SCOP Classification) on using the distance matrix to classify proteins from a very small subset of SCOP database [6] of 698 proteins. The distance matrix is computed for the protein and fed into a multiclass support vector machine (SVM) of the One-Versus-One (OVO) variant. This approach achieved an accuracy of 74.55% on this small subset of SCOP.

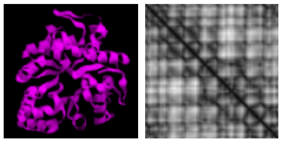
We take a look at few of the representative proteins from alpha proteins, beta proteins, alpha/beta proteins, and alpha+beta proteins. We juxtapose each pro-



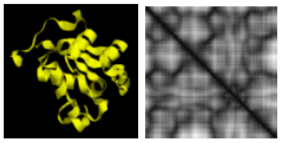
(a) d1hbga_, all α, Globin-like, chain length 147



(b) d1neua_, all β, Immunoglobulin-like, chain length 115



(c) d1ghra_, α/β, TIM-barrel, chain length 306



(d) d1npka_, α+β, Ferredozin-like, chain length 150

Figure 2: A comparison of the distance matrix from the proteins of different classes. We see that the distance matrix is able to pick up on characteristic alpha helix and beta sheets of each protein, making it likely that this feature will perform well.

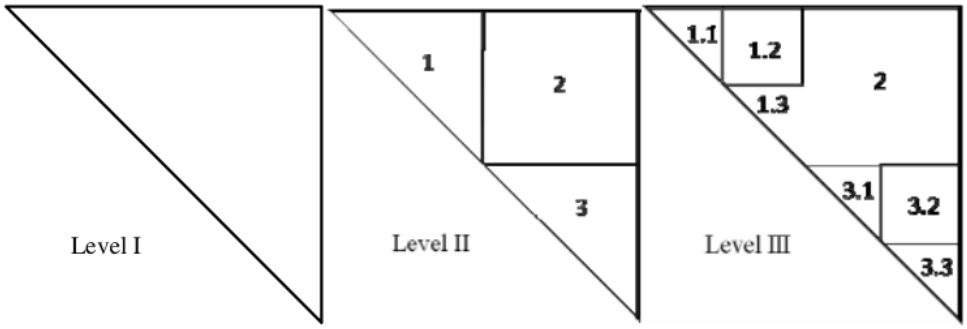



Figure 3: Fast SCOP Classification decomposing the upper triangle of a distance matrix into regions of interest (ROI) to extract important features such as alpha helices and beta sheets.

tein's distance matrix to the protein's structure to see if there are significant differences between the classes. (Fig.2) We see that the distance matrices do represent the alpha helices, the beta sheets, and also the relationship between these features.

Fast SCOP Classification extracts regions of interest (ROI) from the distance matrix by decomposing the upper triangle of the distance matrix into ROI (Fig.3). These regions attempt to capture the regions of the distance matrix where there are alpha helices and beta sheets. The relationships between these regions of interest are abstracted into a feature set which is given to the SVM for fold classification.

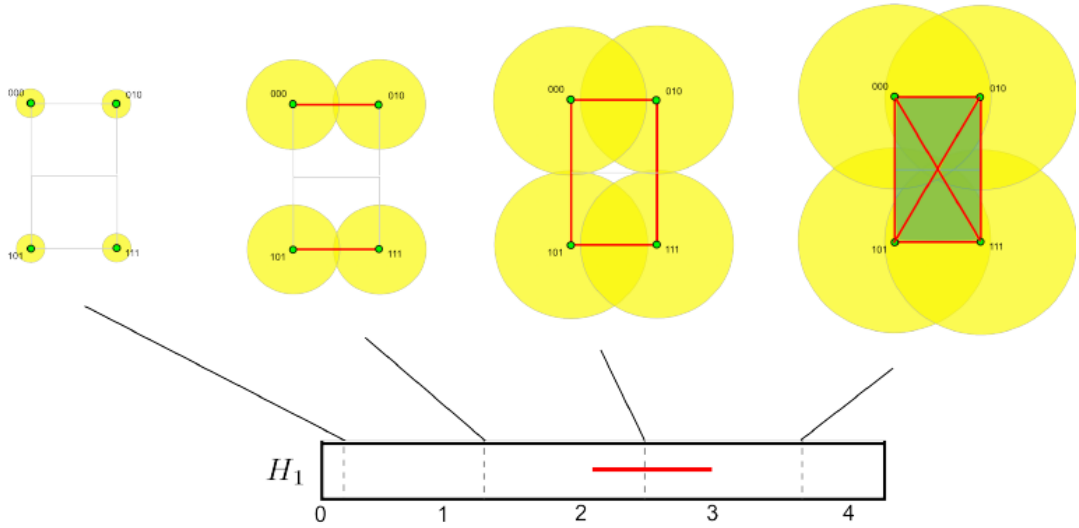



Figure 4: Here is persistent homology characterizing the point cloud of four points. We see that a rectangular cycle is formed when the 4 boundary edges of the rectangles are formed at $\epsilon = 2$. We see that this rectangular cycle is dies at $\epsilon = 3$

In comparison to our method, we employ a convolutional neural network to have the model learn the ROIs on it's own. The CNN has much more advantages over the Fast SCOP's method because complex relationships between regions of interest can be modeled by the hidden layer. Also the regions of interests are dynamically formed as the model is trained. There aren't supervisions into constructing the high-level feature set of ROIs, allowing the model to learn complex behaviors that may elusive or be difficult to define clearly. However, a downside to the CNN is that it is very computationally intensive, requiring a long training time and a large set of data to learn effectively.

### Persistent Homology

Persistent homology is a mathematical theory from algebraic topology that allows the characterization of a point cloud by the observing creations of holes, which we'll call cycles, in the data and how long these cycles persist. We iterate over a real number variable, $\epsilon$, and connect the edges in our point cloud whose length is smaller than $\epsilon$.

A hole is formed when a closed loop is formed by the path of the edges. In the example with 4 points (Fig.4), a cycle is formed when the four edges of the

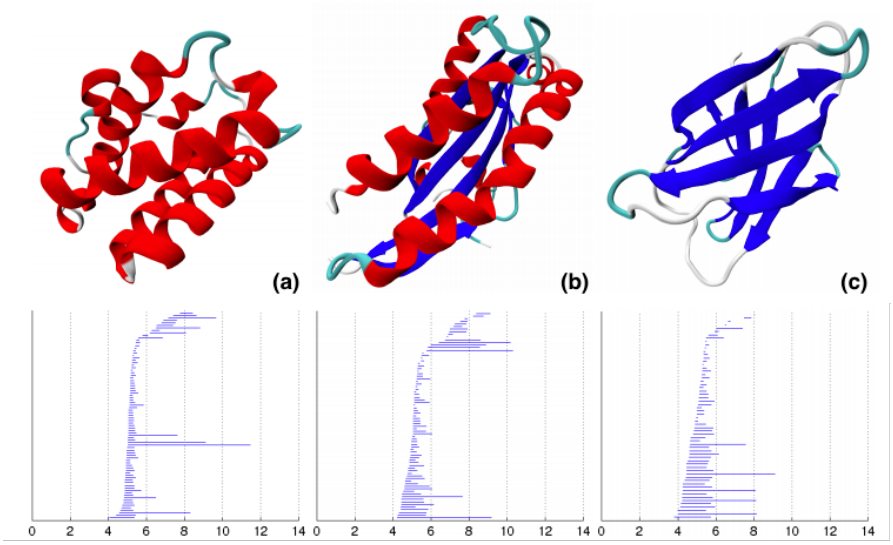



Figure 5: Persistent homology was used to calculate the barcodes on three proteins from Alpha protein (a), Alpha/Beta protein (b), Beta protein(c) classes. The proteins with alpha helices (red) have elongated structures at the top of the barcode while proteins with beta sheets (blue) are thick at the base.

rectangle's boundary is formed. The $\epsilon$ value at which the cycle is formed is called the *birth* of this cycle. This cycle is destroyed when triangles formed within our cycle to completely fill it in. The $\epsilon$ value associated with the cycle being filled in is called the *death* of the cycle. We say that the cycle persists for a duration of $birth - death$. Each cycle forms an interval, $[birth, death]$. The collection of all intervals from the cycles in the data is called a barcode.

There has been existing attempts at using persistent homology to classify classes, the highest level of SCOP hierarchy [8]. Cang applies persistent homology on a small set of 900 proteins selected from Alpha, Beta, and Alpha/Beta classes. Each class is represented by 300 proteins and 60 proteins were selected for testing. A SVM is used to classify the classes with an average accuracy of 85%.

When we observe the barcode of the proteins from the different classes we do see some distinguishing patterns (Fig.5). The proteins with alpha helices have a long curved edge towards the top of the barcode, while proteins with beta sheets have a thicker bar towards the bottom.

In comparison to our method, we classify proteins based on folds, a more specific level of the SCOP hierachy. Our task of classification is much more extensive

than the classification task performed in Cang. We classify 276,231 proteins into 1232 labels compared to the 900 proteins into 3 labels. We also use a convolutional neural network as opposed to a SVM.

In the examples we have discussed, we note that the distance matrices contain much more information than the barcodes. This difference in resolution may cause the matrices to perform better than the barcodes.

# 2 Materials

## 2.1 Datasets

The SCOP and SCOPe database is a database of proteins organized into hierarchical classes based on their shape and function. There are 4 levels of the important 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 and SCOPe 2.07 were downloaded from the Berkeley repository as tar files and unpacked. For each of the datasets, index files are provided.

SCOPe 2.07 has about 8 times more entries than SCOP 1.55 as well as twice the amount of protein folds that we need to classify. The rapid growth of SCOP 1.55 to SCOPe 2.07 can be attributed to the use of automatic classification that was applied to assign hierarchy to newly discovered proteins.

Because of computation restrictions early in our research, we could not apply our methods to the large SCOPe 2.07 dataset which had an unzipped size of about 40gb. The methods in our research initially started by applying fold classification on SCOP 1.55. Since SCOP 1.55 is a subset of SCOPe 2.07, there weren't any surprises in applying the methods from SCOP 1.55 to SCOPe 2.07.

Later in our research, when we were developing new methods, we were able to follow this similar workflow of working in the smaller dataset and then applying our method on the larger dataset. This workflow allowed us to quickly prototype and optimize our methods before running it on the larger dataset.

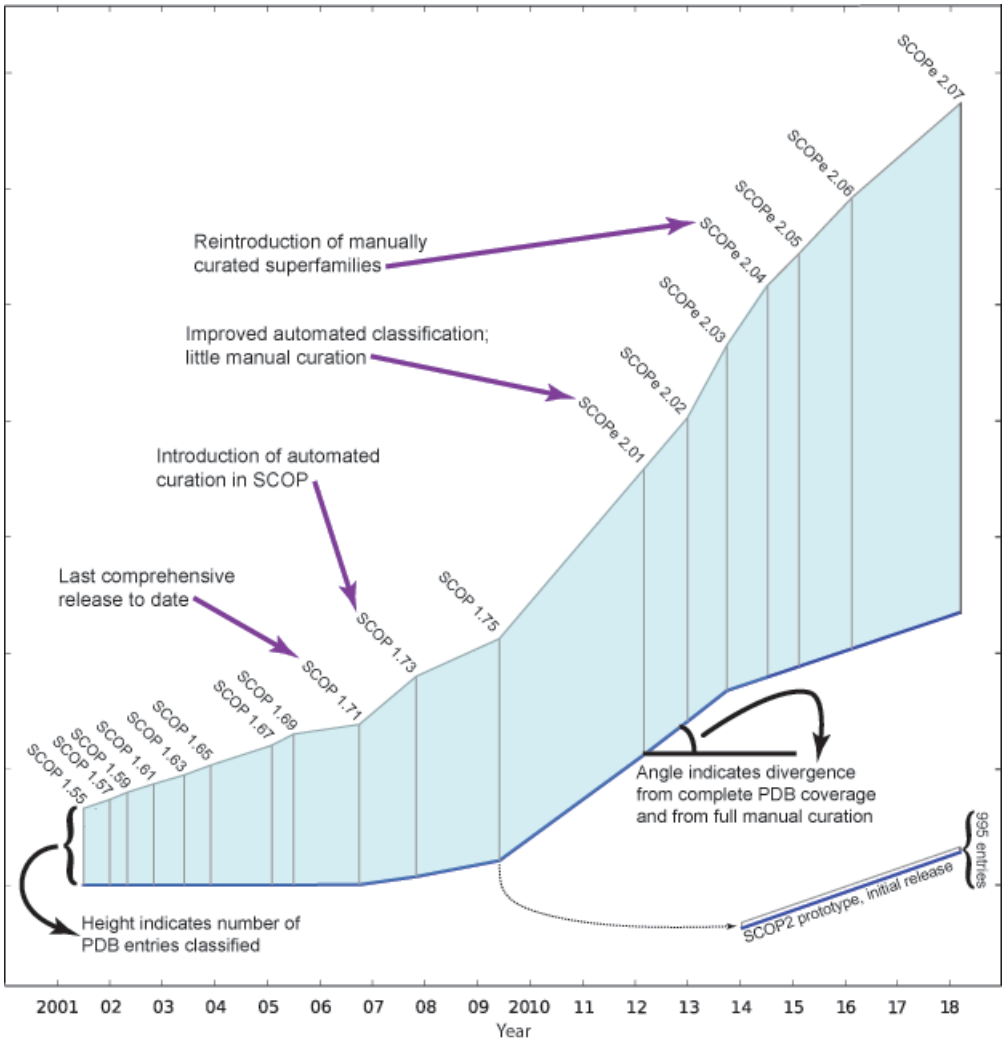



Figure 6: A graph of the development of our dataset from SCOP 1.55 to SCOPe 2.07

### 2.1.1 Small Dataset SCOP 1.55

| Class | Number of folds | Number of superfamilies | Number of families |
|---|---|---|---|
| a. Alpha proteins | 138 | 224 | 337 |
| b. Beta proteins | 93 | 171 | 276 |
| c. Alpha and Beta proteins (a/b) | 97 | 167 | 374 |
| d. Alpha and beta proteins (a+b) | 184 | 263 | 391 |
| e. Multi-domain proteins (alpha and beta) | 28 | 28 | 35 |
| f. Membrane and cell surface proteins and peptides | 11 | 17 | 28 |
| g: Small proteins | 54 | 77 | 116 |
| **Total** | **605** | **947** | **1557** |

Table 1: SCOP 1.55 statistics of 31474 entries

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 until 2001. This dataset is a subset of the SCOPe 2.07 dataset.

We inspect the distribution of the proteins across the different folds. We saw that most of the folds do not have a lot of proteins. The median number of proteins per fold was 10 and the histogram (Fig.7) shows that most of our proteins have less than 50 proteins per fold. This would likely impact our learning, since there are not many examples for the protein to learn.

We also inspect the length of the proteins in our database (Fig. 8). We define the size of the protein to be the length of the protein. We see that most of the proteins have a length less than 300. We do see a small amount of our proteins having lengths between 600 to 800.

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

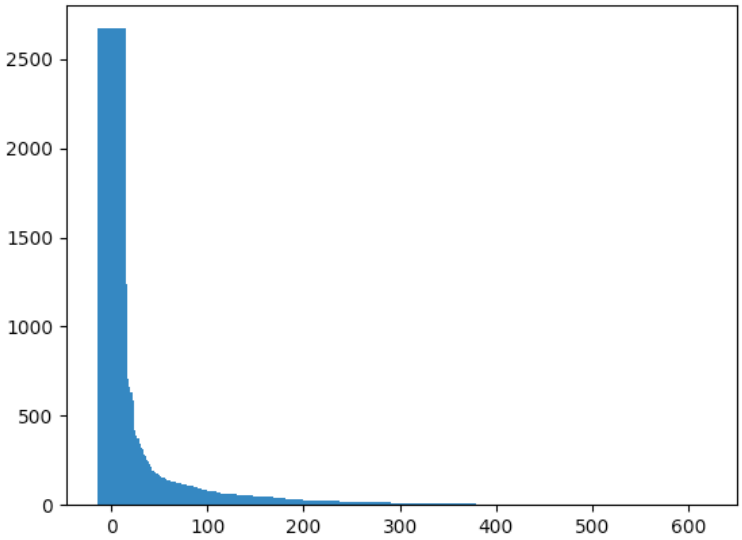



Figure 7: Here is a histogram of the number of proteins per fold for SCOP 1.55. We see that most of the folds do not have too many proteins. This could be a potential issue during learning.

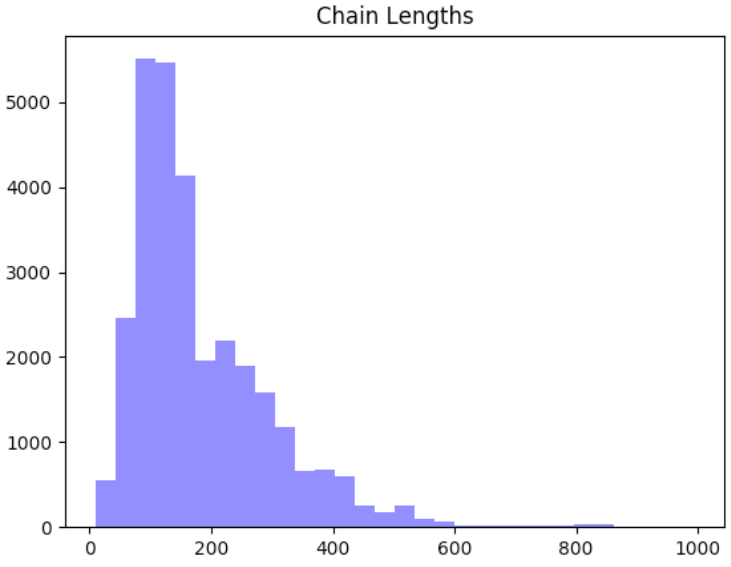



Figure 8: Here is a histogram of the protein lengths for SCOP 1.55. We see that 90% of the proteins have length less than 500

of folds are represented.

### 2.1.2 Large Dataset SCOPe 2.07

| Class | Number of folds | Number of superfamilies | Number of families |
|---|---|---|---|
| a. Alpha proteins | 289 | 516 | 1062 |
| b. Beta proteins | 178 | 370 | 968 |
| c. Alpha and Beta proteins (a/b) | 148 | 246 | 986 |
| d. Alpha and beta proteins (a+b) | 388 | 565 | 1338 |
| e. Multi-domain proteins (alpha and beta) | 71 | 71 | 118 |
| f. Membrane and cell surface proteins and peptides | 60 | 119 | 173 |
| g: Small proteins | 98 | 139 | 274 |
| **Total** | **1232** | **2026** | **4919** |

Table 2: SCOPe 2.07-stable statistics of 276231 entries.

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 8 times larger than the SCOP 1.55 dataset.

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

Once again, we inspect the length of the proteins in our dataset.

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.

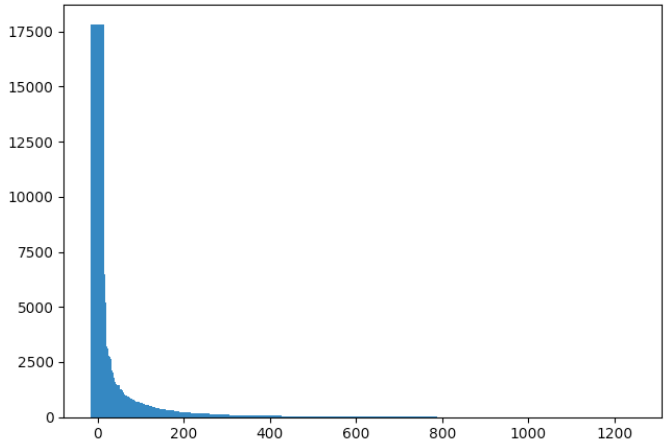



Figure 9: Here is a histogram of the number of proteins per fold for SCOP 2.07. We see that each fold has much more examples than SCOP 1.55. This may allow our model to learn better.

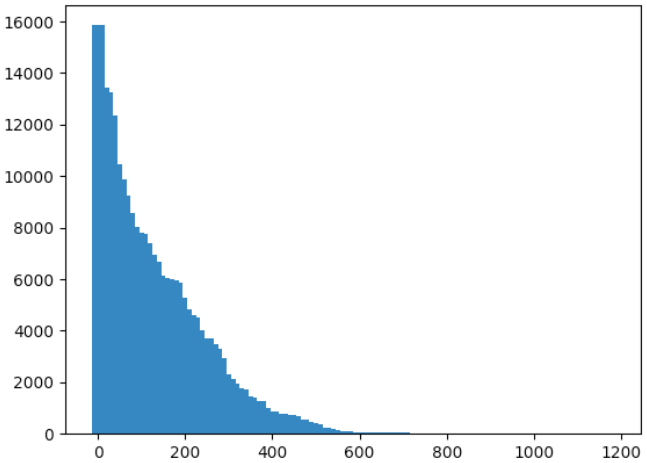
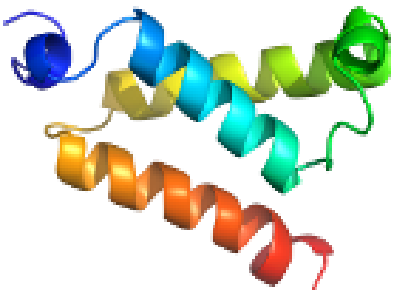
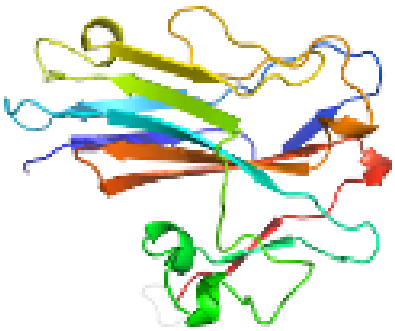



Figure 10: Here is a histogram of the protein lengths for SCOP 2.07. We see that 90% of the proteins have length less than 500



(a) Alpha protein d1kdxa_



(b) Beta protein d2giyb2

(c) Alpha/Beta protein d1btma_

(d) Alpha+Beta protein d1bc4a_

Figure 11: Here are sample proteins representative of the first 4 types of SCOP classes. We note that most of the proteins in these classes show clear structural similarity.

### 2.1.3 Members of SCOP Classes

We take a look at the members of the proteins in each of the 7 SCOP classes to understand their composition. When looking at the first 4 classes, (Fig.11) we see that most of the protein in the Alpha, Beta, Alpha/Beta, and Alpha+Beta classes fall in line with their descriptions in the SCOP paper. Alpha proteins consists almost exclusively of alpha helices, beta proteins consist almost exclusively of beta sheets, Alpha/Beta proteins consist of alpha helices and beta sheets very tightly intertwined, and Alpha+Beta proteins consist of alpha helices and beta sheets relatively in separate locations.

When we look at the remaining classes, we see a wider range of topological structures within each class (Fig.12). Within the Multi-domain protein class, we see protein d1i99i (Fig.12a), which has Alpha Helices and Beta sheets in relatively different locations, and protein d1x75a1 (Fig.12b), which consists only of alpha

helices. Within the Membrane and cell surface protein and peptide class, we see protein d1uaza_ (Fig.12c) which consists entirely of alpha helices and protein d1m3ia (Fig.12d) which consists of alpha helices and beta sheets interspersed throughout the protein. Finally, for the small protein class, we see protein d1ehsa_ (Fig.12e), which is comprised solely of alpha helices and protein d1wj2a_ (Fig.12f). which is comprised solely of beta sheets. Furthermore we note that protein d1i99i (Fig.12a) has structural similarity to proteins in Alpha+Beta Class and protein d1x75a1 (Fig.12b) has structural similarity to proteins in Alpha Class. These inter-class structural similarities may pose a challenge when trying to classify these proteins accurately.

## 2.2 SCOP Viewer

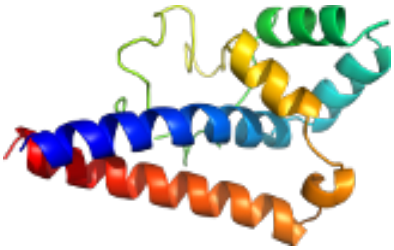
A number of tools were developed during the research to help view the data and guide the direction of the research. SCOP Viewer, a tool to visualize protein structure and the navigate through the different barcodes was developed to inspect the output of the protein backbone parse and the homology cycles generated by the persistent homology. The tool allows 3D navigation, allowing much more clarity in understanding more complex structures than a 2D image.

The tool is very flexible for general use, allowing the user to feed in a point cloud and allowing the user to view the generated persistence homology.

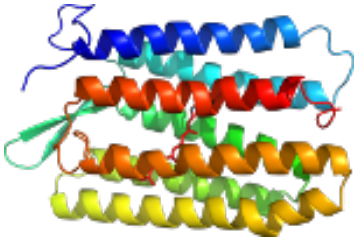
## 2.3 Third Party Tools

A number of third party tools were used as part of the research. ProDy's (Protein Dynamics & Sequence Analysis) and BioPython's 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 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.

(a) Multi-domain protein d1i99i_

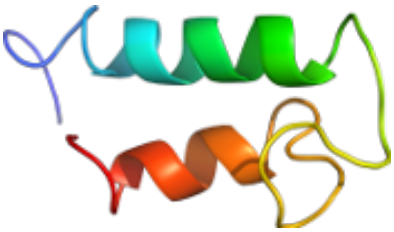


(b) Multi-domain protein d1x75a1

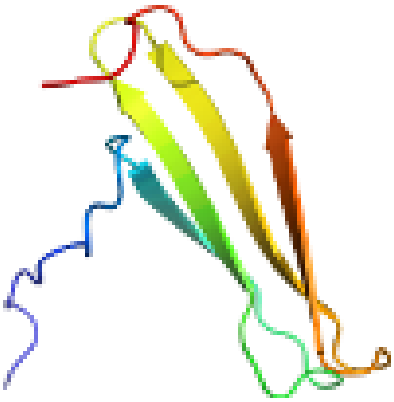


(c) Membrane and cell surface protein and peptide d1uaza_

(d) Membrane and cell surface protein and peptide d1m3ia



(e) Small protein d1ehsa_



(f) Small protein d1wj2a_

Figure 12: Here are pairs of very different proteins within each of the last three types of SCOP classes. We note that some of these proteins have structures very similar to some of the proteins in the first four SCOP classes. These variations indicate structural dissimilarity within the last three SCOP classes.

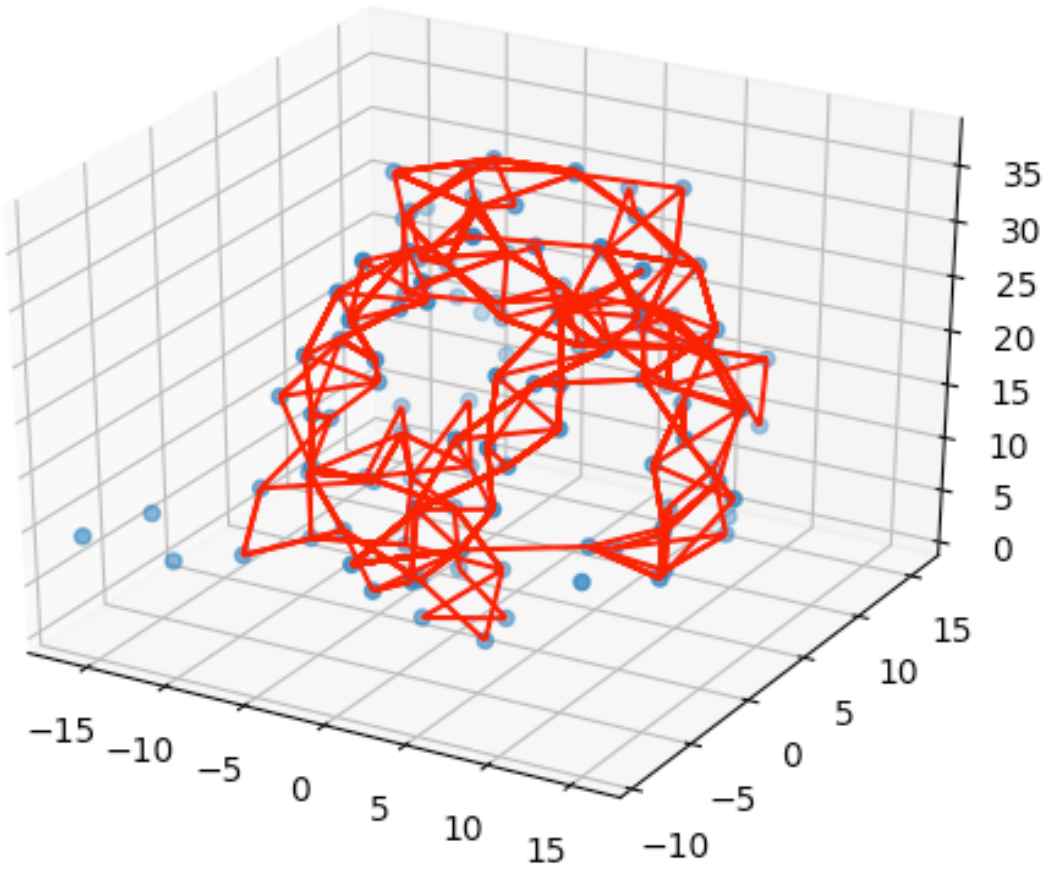



Figure 13: SCOP Viewer allows the user to view the generated persistence homology of the data at once. Here are the generated cycles of protein 1dly

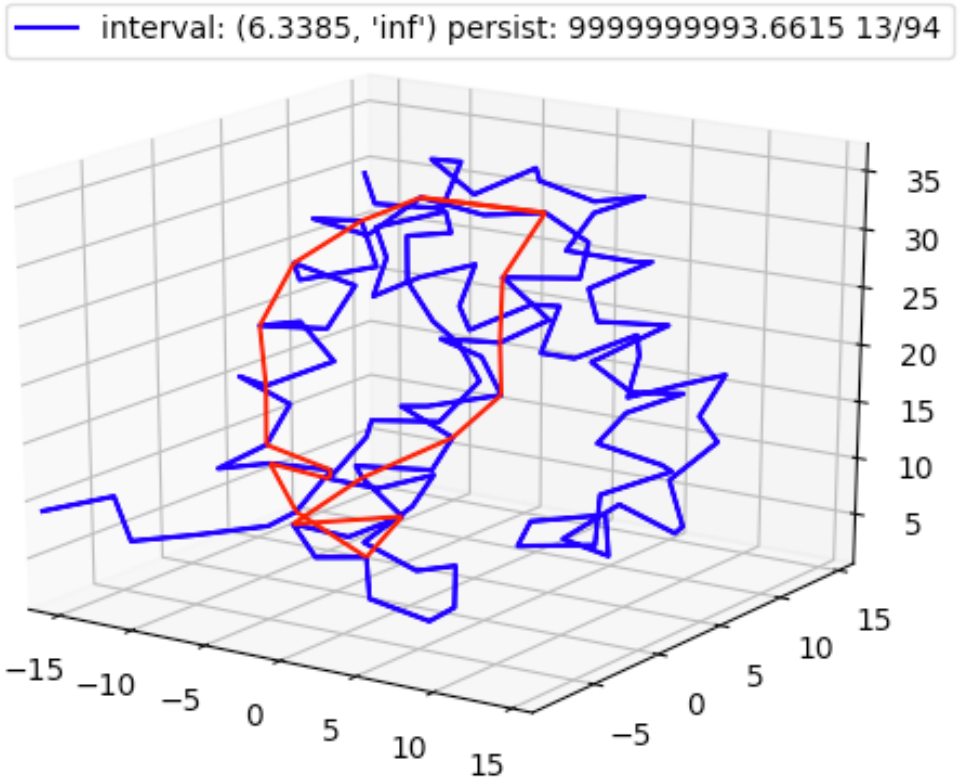



Figure 14: SCOP Viewer allows the user to cycle between the various types of persistent cycles generated for our protein. Here we have a cycle describing a beta sheet structure for protein 1dly.

CPickle package was used to serialize and deserialize processed data.

## 2.4 Computing Resources

Initially the research started on a personal laptop without a GPU and very limited storage space (Tbl.3). 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.

| | **Personal Laptop** | **Personal Workstation** |
|---|---|---|
| **CPU** | Intel Core i7-4650U | E5-2630 6-Core |
| **GPU (CUDA-enabled)** | None | Nvidia GTX 1060 6GB |
| **RAM** | 8GB | 16GB |
| **Storage** | 128GB SSD | 500GB SSD |
| **OS** | MacOSx | Ubuntu 16.02 |

Table 3: A comparison of the laptop and the workstation. Purchasing the workstation made possible significant advancements in the research by increasing the speed of computation and also allowing to work with larger datasets.

Due to the limitations of the personal laptop, a personal workstation was purchased at $500 in Winter of 2018 (Tbl.3). 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, a set of 22 compounds which link sequentially into a protein chain. The interaction between amino acids and the surrounding environment determine 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 (Tbl.4).

| | |
|---|---|
| Index Entry | d1dlwa_ 1dlw A: a.1.1.1 14982 cl=46456,cf=46457 ... |
| Name | d1dlwa_ |
| Class.Folds.Superfamily.Family | a.1.1.1 |

Table 4: Here is the relationship between the index entry of a protein and the protein in the dataset. The PDB file for this protein is 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 10,000. 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 distance provided in the protein PDB

files is Ångströms. 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.

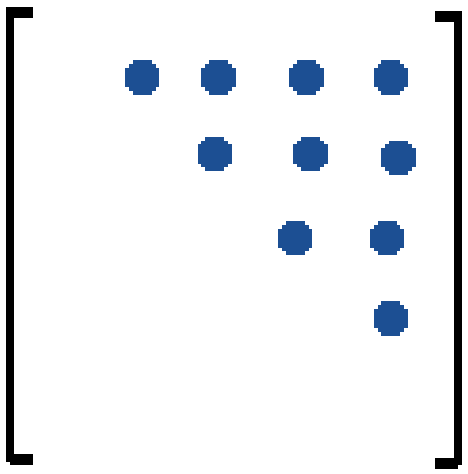


Figure 15: Here we have the entries in a matrix that correspond to the upper triangle. We note that since our distance matrix is symmetric with respect the diagonal line, we only need to calculate the distance values once for the entries in the upper triangle of our matrix.

We see that the distance matrix is symmetric 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 (Fig.15). This region is consists of $\{M_{ij}|i < j\}$. 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.

**Algorithm 1** We fill in our distance matrix by calculating the upper triangle of a NxN matrix. Our rows and columns range from 1 to N.

```
M is distance matrix set with zeroes
Euclid(a,b) is the distance between a and b
for r ¡= N do
  for c ¡ r do
    d=Euclid(r,c)
    M(r,c)=d
    M(c,r)=d
  end for
end for
```

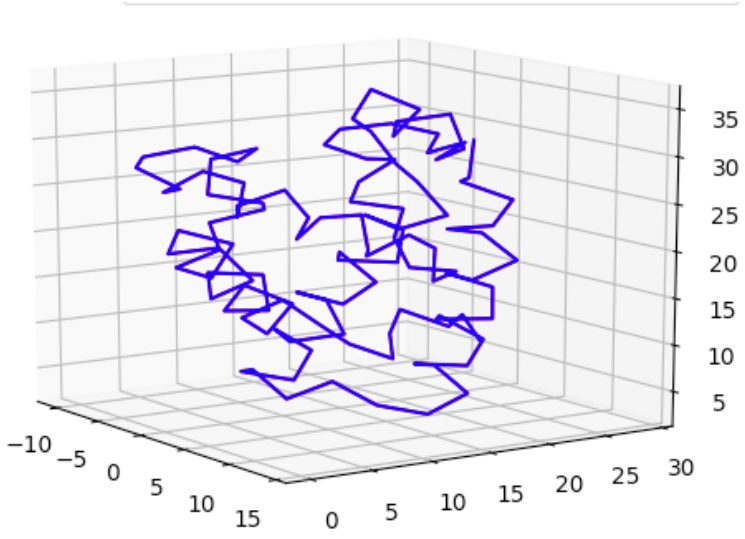



Figure 16: Here is the 3D shape of the 1ux8 protein's backbone chain. We see that there are notable features, alpha helices and beta sheets, that we would like to represent in our distance matrix.

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 (secondary structures) are known in molecular biology to be important features of a protein's shape.

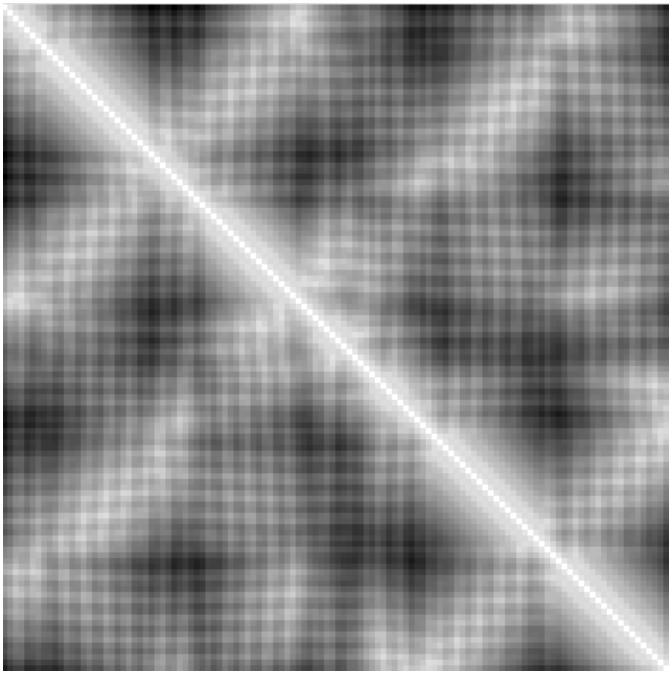


Figure 17: Here is the distance matrix of the 1ux8 protein. We see a thick diagonal line with corresponds to the protein backbone chain.

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 appear in white and furthest distances appear in black (Fig.17). 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. We note that this diagonal white line uniquely identifies the protein backbone chain (since only the distance between a point and itself has the distance of zero). 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 (Fig.18). These regions indicate that at a given point in the chain, it is in close proximity to the nearby neighbors. 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.

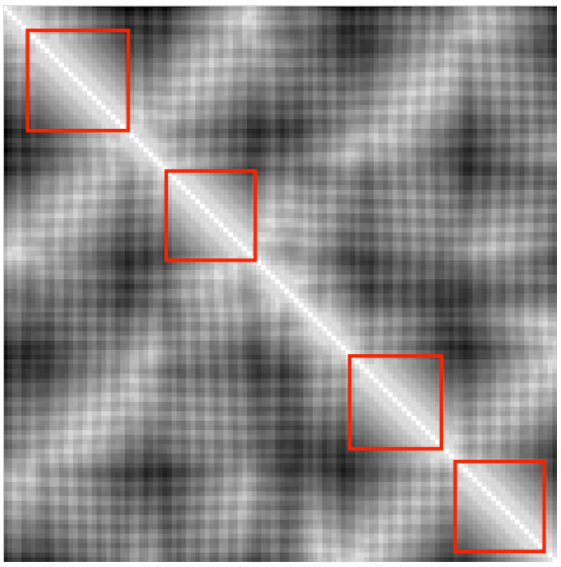


Figure 18: The red squares outline the discrete thick white regions running parallel to the diagonal line. These regions indicate that the neighbors of points in this region are in very close proximity, indicating that there may be a helix.

**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 $\{C_i | i \in H\}$.

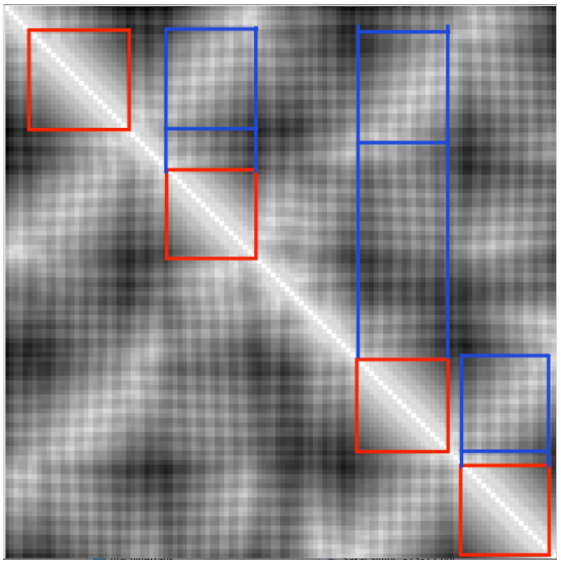


Figure 19: The blue squares outline the regions of proximity between points on different helices, suggesting that the helices are in close proximity. The direction of the lines indicate whether the two helices are parallel or anti-parallel.

**Feature Beta Sheet:**

We note patches of thick white lines in the intersection of the rows belonging to a helix and the columns belonging to another helix (Fig.19). 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.

### 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.

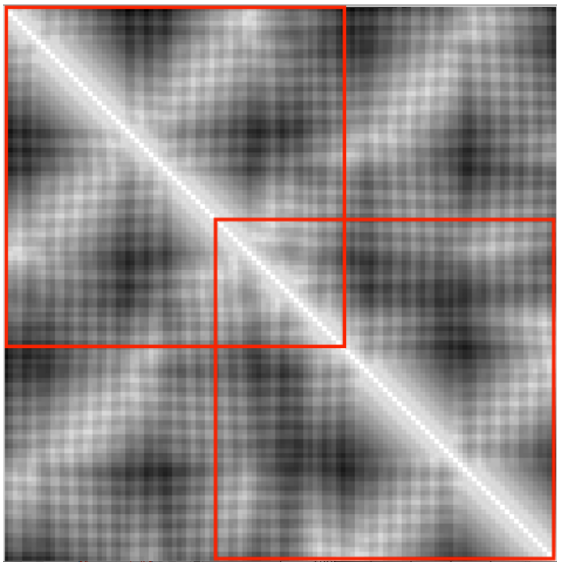


Figure 20: Due to computational restrictions, we crop our matrix by shifting the red crop window along the diagonal line of our matrix.

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 (Fig.20), 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.

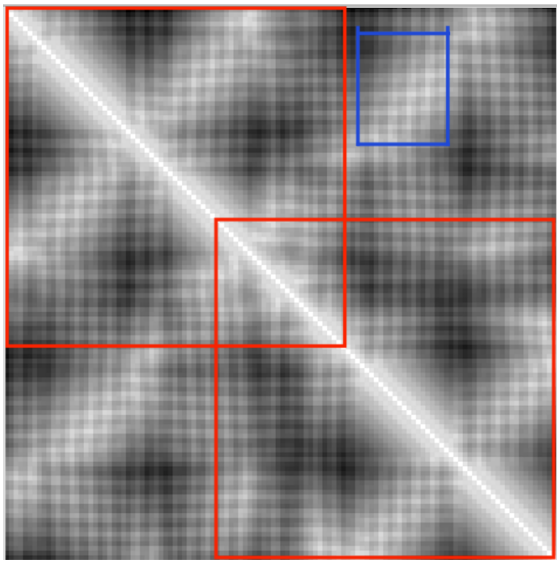


Figure 21: We note that cropping the distance matrix may cause a loss of long range contact information. In our protein 1ux8, the cropped matrices do not detect that the alpha helix 1 forms a beta sheet with alpha helix 3

We note that if a distance matrix is smaller than our cropping window size, we pad the distance matrix.

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.

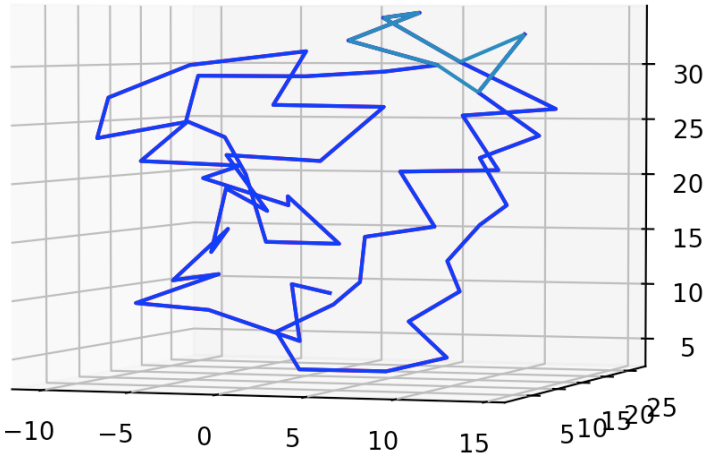



Figure 22: Due to computation restrictions, we sample every other point on our protein and compare it with the original structure. We see that, although diminished, most of the features of the protein are still distinguishable.

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 (Fig.22) and compare it to the original protein backbone (Fig.16). In comparison, we see that the general structure of the protein and its features are diminished but preserved.

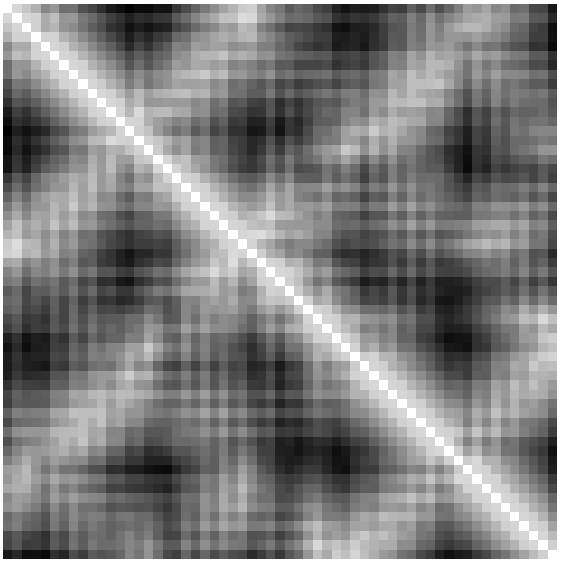


Figure 23: Here we have a distance matrix of the sampled 1ux8 protein backbone. We do see that the protein backbone, alpha helices, and beta sheets that were observed in the full distance matrix can also be observed in the sparse distance matrix.

From the sparse protein backbone, we construct a sparse distance matrix in the same fashion as a regular distance matrix. In example, we compare the sparse distance matrix (Fig.23) and the unmodified distance matrix (Fig.17) 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 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.

## 3.4 Persistent Homology

**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.

### 3.4.1 Mathematical Background

A good deal of effort went into trying to make the mathematical definitions used in this section self-contained. One could read through this section to get a general understanding of the theory behind persistence homology. However, there are some fundamental concepts in abstract and linear algebra that were not defined. These are, but not limited to the following: group, normal subgroup, quotient group, homomorphism, kernel, image, vector space, dimension of a vector space.

We discuss the mathematical theory behind persistent homology prior to introducing an algorithm to compute it. First, we describe the elementary mathe-

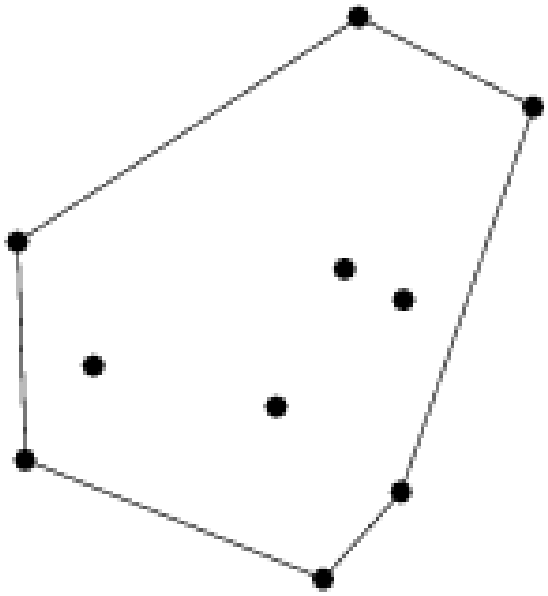


Figure 24: Illustration of a convex hull of 10 points. For a finite set of points this is just a a polygon connecting the outer most points.

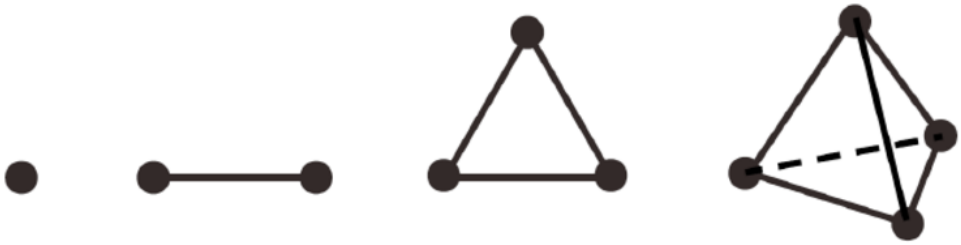


Figure 25: Here are illustrations of a 0-Simplex (a point), 1-Simplex (a line), 2-Simplex (a triangle), 3-Simplex (a tetrahedron)

matical objects. Then we describe the topological features of a data that we want to extract using the homology and the structure theorem. We also discuss the stability of the topological features described by the stability theorem.

**Definition 3.4.** A **convex set**, $C \subset \mathbb{R}^n$, is a set where $\forall a, b \in C$, $t \in [0,1]$ $ta + (1-t)b \in C$.

**Definition 3.5.** A **convex hull** $< A >$ of a set of points, $A \subset \mathbb{R}^n$ is defined by

$$< A >:= \bigcap \{C \mid C \text{ convex in } \mathbb{R}^n \ , \ A \subset C\}$$

**Definition 3.6.** An $n$**-simplex** is a convex hull of $n+1$ points. When we refer to varying sizes of n-simplices together or when the size is undetermined, we will call them just simplex/simplices.

We note that 0-simplex is a point, 1-simplex is an edge, and 2-simplex is a trangle.

**Definition 3.7.** Let $\sigma$ be a simplex.

- The **vertices** of $\sigma$ are its points.
- The **face** of $\sigma$ are the simplices formed by subset of the vertices of $\sigma$.
- A **n-face** is a face of $\sigma$ with $n+1$ vertices.

**Definition 3.8. Simplicial complex** is a set of simplices, $S$, such that

- Any face of a simplex $\sigma \in S$ is in $S$.
- The intersection of any two simplices in $S$ is in $S$

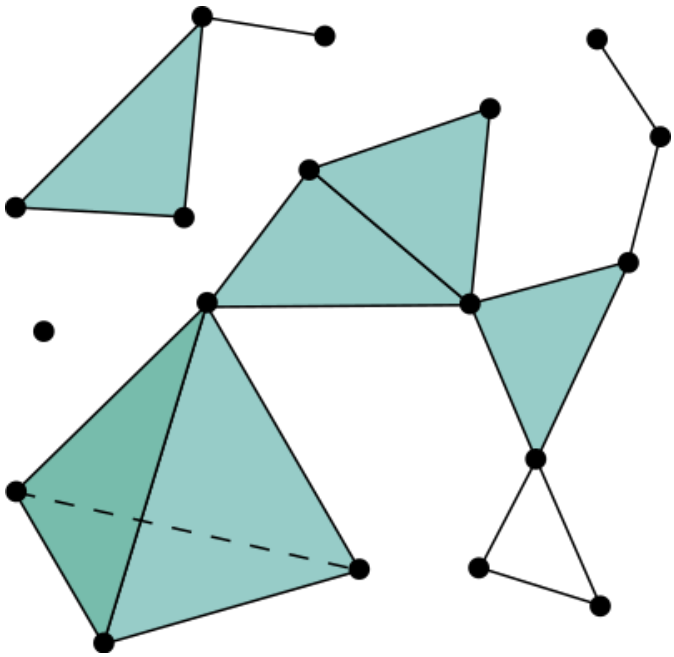


Figure 26: Here is an illustration of a simplicial complex. Notice that it is a collection of points,edges and triangles.

Now that we have defined the foundational definitions, we begin to define structures that can describe the topological characteristics of the points in a protein's backbone. First, we will observe the structure of our protein as we create

simplices by adding edges of increasing length between the points on the protein's backbone. We do this by forming Vietoris-Rips complexes.

**Definition 3.9.** Given a set of points with a metric, a **$\epsilon$-Vietoris-Rips complex** on the set of points is a simplicial complex which is formed by connecting the points with distance less than or equal to $\epsilon$.

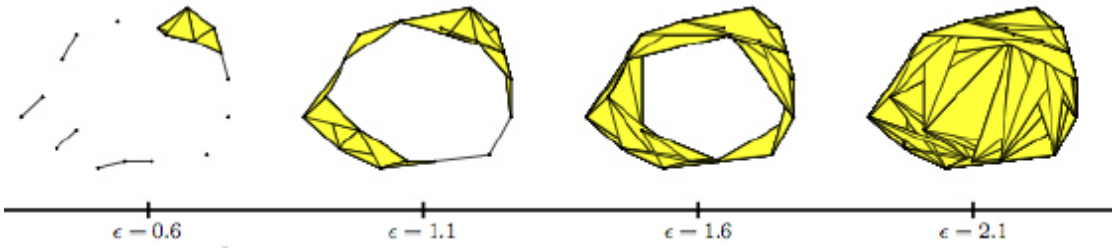



Figure 27: Here is a series of 5 Vietoris-Rips complexes. At $\epsilon = 0$ we only have a collection of points, while at $\epsilon = 1.1$ we being to see an interesting characteristic of our data beginning to form. Finally, at $\epsilon = 2.1$ we do not see much of a structure as most of the points become interconnected to each other.

First, we create $\epsilon$-Vietoris-Rips complexes of our points in the protein backbone for all intervals of $\epsilon$. We see how the state of our data changes as we add edges between points of increasing length by ordering the $\epsilon$-Vietoris-Rips complexes in increasing order of $\epsilon$. This ordering is a filtration on the set of all Vietoris-Rips complexes in our data. If $\epsilon_1 \leq \epsilon_2$ then $\epsilon_1$-Vietoris-Rips complex is contained in $\epsilon_2$-Vietoris-Rips complex. This is because edges less than $\epsilon_1$ are also less than $\epsilon_2$ and thus are formed in $\epsilon_2$-Vietoris-Rips complex.

**Definition 3.10. A filtration** is an ordering on $S_i \in \mathbb{S}$ such that

$$S_1 \subseteq S_2 \subseteq S_3 \subseteq S_4 \subseteq ...$$

.

We denote $\epsilon$-Vietoris-Rips complexes as $V_\epsilon$ and write our filtration of $V_\epsilon$s as the following.

$$V_{\epsilon_1} \subseteq V_{\epsilon_2} \subseteq V_{\epsilon_3} \subseteq V_{\epsilon_4} \subseteq ...$$

For each $V_\epsilon$, we convert it to an algebraic vector space to allow us to detect cycles and other topological features in our simplicial complex. We say we apply a homology on $V_\epsilon$. We will define the building blocks prior to defining the homology on a simplicial complex. We will also refer to cycles as holes because the latter word is more illustrative of the physical characteristics.

**Definition 3.11.** $k$**-Chain** of $S$ is an abelian group of elements consisting of

$$\sum z_i s_i$$

where $z_i \in \mathbb{Z}$ and $s_i$ is a $k$-simplex of S.

$C_k$ is also a vector space.

**Definition 3.12.** The **boundary operator** $\partial_k : C_k \to C_{k-1}$ is a homeomorphism defined as

$$\partial([v_1, v_2, ..., v_k]) := \sum_{i=0}^{k} (-1)^i [v_1, ..., \hat{v_i}, ..., v_k]$$

where $[v_1, ..., \hat{v_i}, ..., v_k]$ is a simplex of all elements $v_1$ to $v_k$ except $v_i$.

Since we are working with points in $\mathbb{R}^3$, we define the boundary operators that we will use.

$$\partial_1([ab]) := b - a$$

$$\partial_2([abc]) := ab - ac + bc$$

**Lemma 3.1.** $\partial_i \circ \partial_{i+1} = 0$.

*In other words, the composition of two subsequent boundary functions is zero.*

The above lemma implies that $Im\partial_{j+1} \subset Ker\partial_j$. Since every subgroup of a normal subgroup is normal and $Ker\partial_j$ is a normal subgroup, we can form the quotient group $\frac{Ker\partial_j}{Im\partial_{j+1}}$.

**Definition 3.13.** Given a simplicial complex, $S$, it's **j-homology** is the algebraic quotient group, $\frac{Ker\partial_j}{Im\partial_{j+1}}$,

with the boundary functions defined on the $k$-chains of $S$

We denote the j-homology of $S$ as $H_j(S)$

We note that the homology is a vector space since $Z_j \subset C_j$ is a vector space.

We note that the Homology of $S$ is nonzero if $Ker\partial_j \neq Im\partial_{j+1}$. For $H_1$, this means $H_1(S)$ is nonzero if any linear combination of boundaries of triangular simplices in $S$ is not equal to the hole formed by the edges in $S$. In an example with 6 points (Fig.28), we see that the $H_1$ of the simplex is nonzero because the hole created by the edges *abcd* is not the linear combinations of the 2 triangular simplices' boundaries, *ebf* and *dhg*.

The dimension of the homology of a simplicial complex tells us how many holes there are in the simplicial complex.

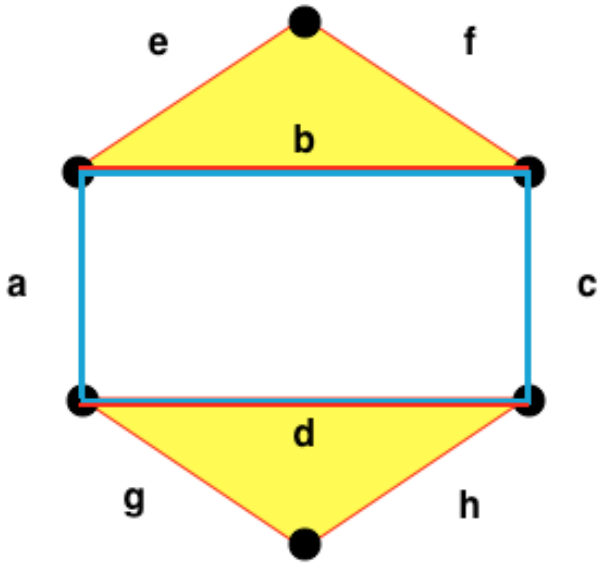



Figure 28: The homology of this simplicial complex consisting of 6 points is not zero because the hole (blue) created by the *abcd* edges in the simplex is not equal to the linear combinations of the 2 triangular simplices' boundaries (red)

So far, we can look at the homology of each $V_\epsilon$ individually and can tell how many holes are in each simplicial complex. We now move towards trying to tie together the information from homology of all the $V_\epsilon$s.

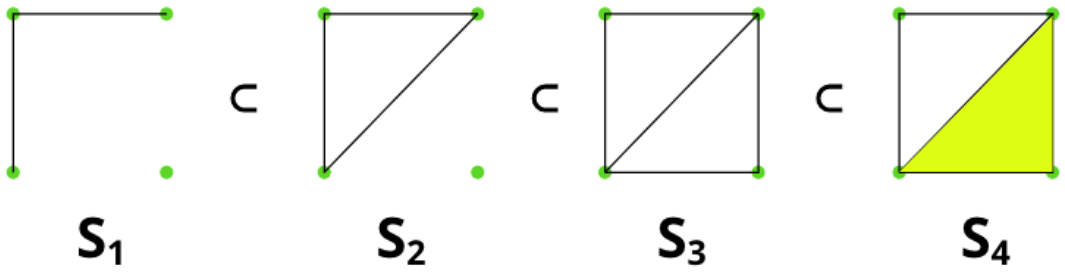
For each $V_\epsilon$ in our filtration, we apply a homology. For our research, we apply a 1-homology, $H_1$, to each of the $V_\epsilon$s. The inclusion maps between the

subsequent homologies induce a sequence of homomorphisms. This sequence for a given filtration is called the persistence module.

$$H_1(V_{\epsilon_1}) \to H_1(V_{\epsilon_2}) \to H_1(V_{\epsilon_3}) \to H_1(V_{\epsilon_4})...$$

**Definition 3.14.** Given a filtration of simplicial complexes, $S_1 \subseteq S_2 \subseteq S_3 \subseteq S_4 \subseteq ...$, the chain of homologies linked by the homomorphism induced with the inclusion map is called the **persistence module**,
$H_i(S_1) \to H_i(S_2) \to H_i(S_3) \to H_i(S_4) \to ....$

From the results of the structure theorem for persistence homology theory, we can decompose the persistence module into sums of 1 dimensional intervals. In the following example, we can reduce the persistence module for the simplicial complexes 4 points into two intervals.




$$\begin{array}{cccccccc} & H_1(S_1) & \to & H_1(S_2) & \to & H_1(S_3) & \to & H_1(S_4) \\ \simeq & 0 & \to & \mathbb{R} & \to & \mathbb{R} & \to & \mathbb{R} \\ \oplus & 0 & \to & 0 & \to & \mathbb{R} & \to & 0 \end{array}$$

The first summand, $0 \to \mathbb{R} \to \mathbb{R} \to \mathbb{R}$, corresponds to the upper triangular hole in $H_1(S_3)$ and is non zero from $H_1(S_2)$ to $H_1(S_4)$ because this hole persists through $S_1$ to $S_4$. The bottom summand, $0 \to 0 \to \mathbb{R} \to 0$, corresponds to the lower triangular hole in $H_1(S_3)$ and is nonzero only at $H_1(S_3)$ since this hole is formed at $S_3$ and dies at $S_4$.

We call summands of this form, where there is a single linked chain of $\mathbb{R}$ ,**intervals**. These intervals form the barcode that tell us the birth and the death of each hole.

We briefly mention a series of theorems that guarantee that we can decompose any persistence module into a sum of unique intervals as we did in our previous example. These theorems are referred to as structure theorems of persistence homology.

**Theorem 3.2.** *A persistence module is indecomposible if and only if it is an interval.*

**Theorem 3.3.** *A vector space is indecomposible if and only if it is 1 dimensional. An interval is 1 dimensional.*

**Theorem 3.4.** *Any persistence module is a finite sum of indecomposible subgroups.*

**Theorem 3.5.** ***(Krull-Schmidt)*** *The decomposition of a persistence module is written uniquely as as a sum of indecomposible subgroups.*

The previous four theorems give us the following. Any persistent module can be written uniquely as a sum of intervals. The consequence is that given any set of points, we can construct a filtration such that we get a unique barcode that gives us a structural characterization of the points. We now give a definition of the barcode graph for persistence homology on filtrations of $\epsilon$-Vietoris-Rips Complexes.

**Definition 3.15.** Given a persistence module $P = \oplus_{i=1}^{n} Interval_i$, the **barcode graph** of the persistece module is

$$\{(birth_i, death_i) \mid 1 \leq i \leq n\}$$

- $birth_i$ is the $\epsilon$ value of the $V_\epsilon$ corresponding to the first nonzero element in the chain of $Interval_i$

- $death_i$ is the $\epsilon$ value of the $V_\epsilon$ corresponding to the last nonzero element in the chain of $Interval_i$

Supposed we had two sets of points that were similar. They would generate very similar filtrations of simplicial complexes. We would hope that these filtrations have similar barcodes. We first define a metric on two barcodes.

**Definition 3.16.** Given, two barcodes $X$ & $Y$, the **bottleneck distance**, $W_\infty(X, Y)$ is

$$W_\infty(X, Y) := \inf_{\phi: X \to Y} sup_{x \in X} ||x - \phi(x)||_\infty$$

The stability theorem gives us an upper bound for the bottleneck distance between two filtrations of simplicial complexes. Thus that a small differences between sets of points lead to small differences in the generated filtrations of simplicial complexes. This in turn leads to a small difference between the barcodes.

In addition to stability of the barcodes, we note that barcodes are translation and rotation invariant features of a point cloud. This is because the barcodes are formed from the filtrations of $\epsilon$-Vietoris-Rips complexes, which ared generated by the distances between the points. Since the distances between the points are translation and rotation invariant, our barcodes are also translation and rotation invariant.

The stability and translation and rotation invariance of the barcodes make it an excellent feature for machine learning.

### 3.4.2 Computation Algorithm

We describe our implementation of the persistent homology, in addition to variations to the method to improve performance and collect more protein specific holes. In practice, we will use the N points of a protein backbone in this algorithm but for illustrative purposes, we consider a simple collection of 17 points in the $\mathbb{R}^2$. (Fig.29) Because we are looking at the distances between our points, the

method performs exactly the same in $\mathbb{R}^3$ or with any collection of objects with a well defined metric.

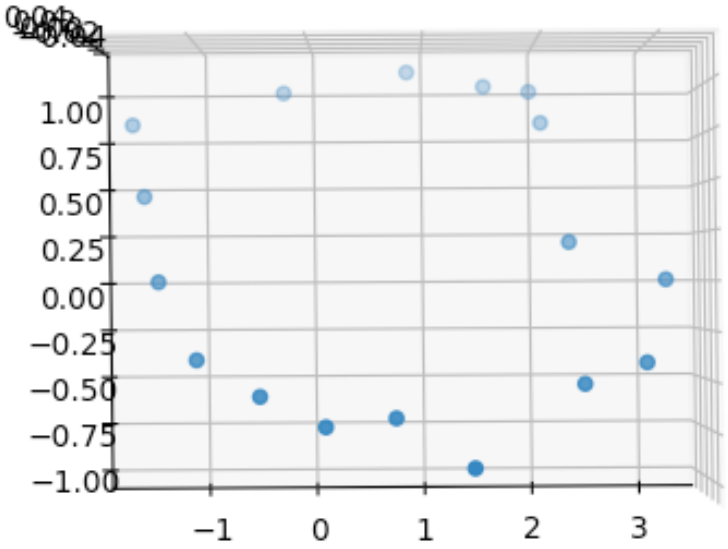



Figure 29: Here is the collection of the 17 points we are considering in 2D. Although our protein is in 3D, the persistence homology is concerned with the distances between points, allowing in to work for points in any dimension as long as there is a well-defined metric.

**Definition 3.17.** We call the (i,j) edge the edge constructed by connecting the ith and the jth point in our set of N points.

A distance matrix of the our points is calculated. We collect the upper triangular elements of the distance matrix to get the unique edges of our points (unique disregarding direction, making (i,j) edge equal to (j,i) edge). These edges are sorted by length.

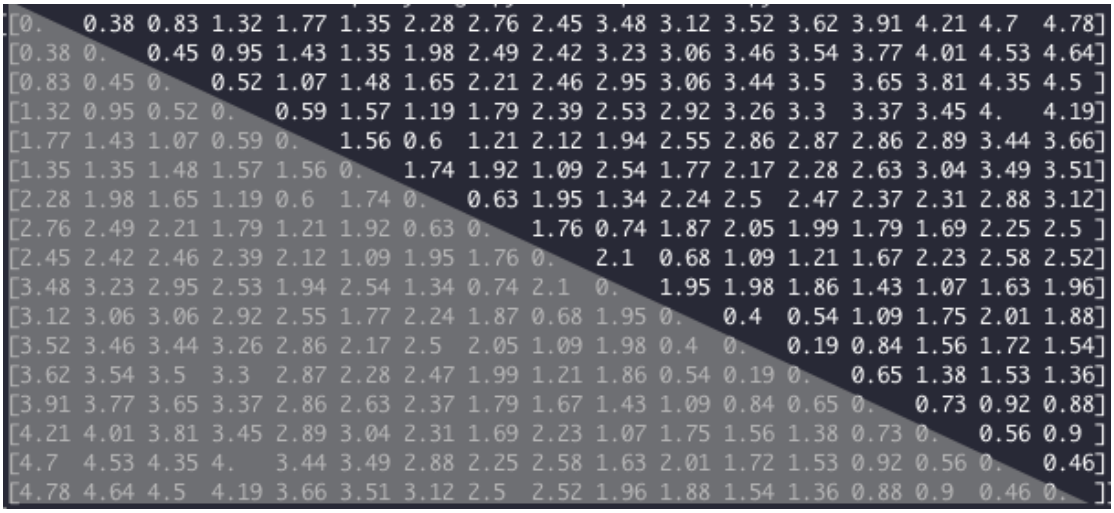



Figure 30: Here is a distance matrix of our set of 17 points. We will only consider the edges in the upper triangular matrix, because the (i,j) edge represents the same line as the (j,i) edge

Consider a protein with length 300. The total number of edges for a protein of length 300 is $\sum_{n=1}^{299} n = 44851$. The total number of triangular simplices formed by these edges $\binom{300}{3} = \dfrac{300!}{3!297!} = 4455100$. We see that the numbers of triangular simplices increase exponentially and very quickly, even for small structures. This demands a need to reduce the computation load.

A cutoff distance is determined, where we discard the edges that are larger than this value. A similar concept exists in bioinformatics, where a point of a protein are determined to be in contact with another point if their distance is within the contact distance (3 Å). The loss of information by discarding the edges larger than this value isn't an issue since because when we start considering larger edges, we start converging towards a structure where every point is connected to each other (Fig.31)

As long as we set the cutoff distance at an appropriate value, we would still get the holes of the protein that represent the characteristic features. Furthermore, setting of a cutoff distance has an added advantage that it filters out uninteresting cycles that are too large.

We experimentally determined the cutoff distance at 6.5 Å, because 6 Å is the approximate distances between two alpha helices in a beta sheet [6]. Further-

more, most sequential distances between the backbone points, $P_i and P_{i+1}$, and the distances within the alpha helices are both less than 6.5, allowing us to extract features related to them as well.

For our set of 17 points, the cutoff distance is determined as $\frac{1}{6}$ of the longest length, which is around 2. Below are the collection of points

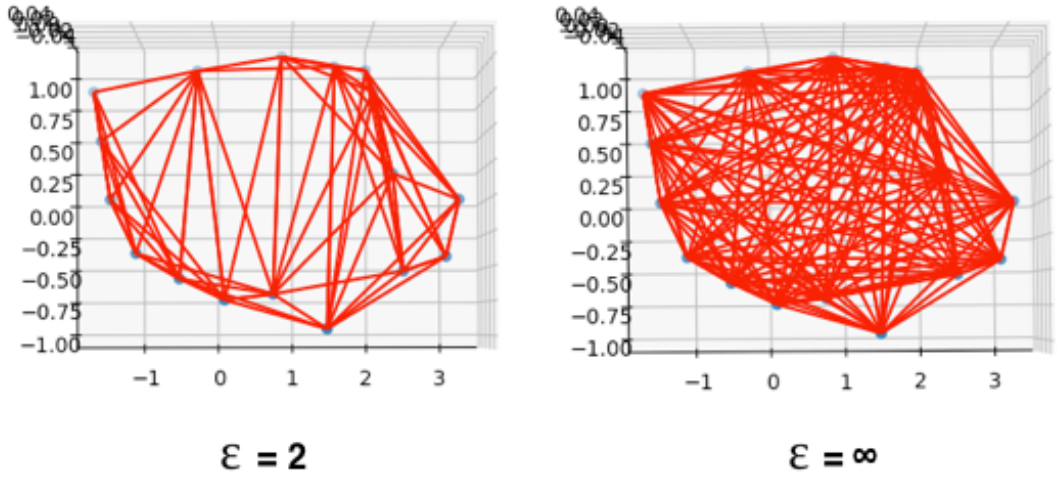



Figure 31: Here we see the effect of the cutoff distance on the triangular simplicies we form. When a cutoff distance is not set ($\epsilon = \infty$), all the edges are considered, forming triangular simplicies where every point is connected to each other. These simplices don't convey too much about the structure of our protein. When a cutoff distance is set properly ($\epsilon = 2$), the important distances considered, making the simplices less convoluted and increasing computation time.

Once we have the edges that we are considering, we find the triangular simplicies formed by our edges. As we loop over the edges, we index the edges from 1 to N based on the endpoints of the edge (For an edge (2,4), it is indexed as 2 and 4). Then, we check if each edge forms triangular simplicies by looking at the indexed edges related to our edge. (For an edge (2,4), we look at the edges that have been index as 2 or 4 to see if we formed a 2-simplex). Because we are forming the triangular simplicies as we add increasing edges, triangular simplicies are ordered by when they are created.

Given a triangular simplex, we can compute when it was born by looking at

**Algorithm 2** Finding triangular simplicies formed by the edges

```
for (i,j) in Edges do
  for edges indexed i and indexed j do
    See if these edges form a triangular simplicies with (i,j)
  end for
  Index (i,j) as i and j
end for
```

it's edges. For example, a triangular simplex (1,2,3) has the edges (1,2), (1,3), and (3,1). Suppose the lengths of these edges were 3,5, and 7 respectively. Then the triangular simplex was formed at radius 7.

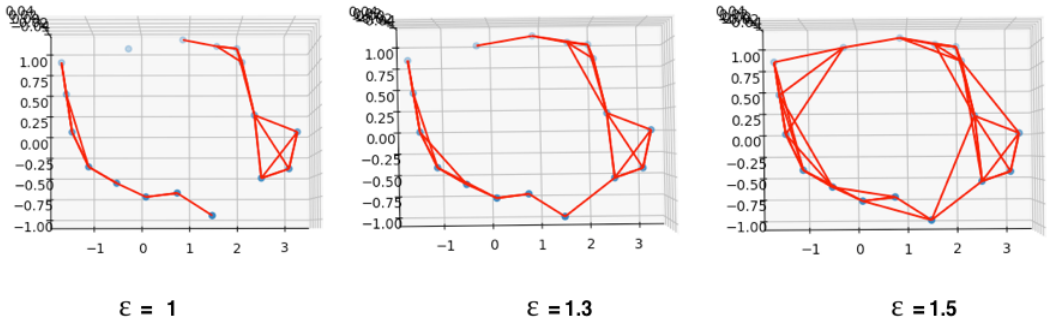



Figure 32: The triangular simplices are discovered as we add edges. Because we are adding the edges in increasing order, the triangular simplices are discovered in the order of their birth.

From the edges and triangular simplices, we create two matrices, D1 and D2. Let M be the number of edges we are considering and L be the number of triangular simplices we are considering.

D1 is a NxM matrix where the rows correspond to the backbone points and the columns corresponds to the edge in increasing order. We fill in D1 in the following order. For the hth edge with endpoints (i,j), we set $D1_{h,i} = -1$ and $D1_{h,j} = 1$. Suppose an edge (2,4) is the 10th smallest edge. The corresponding entries in would be $D1_{10,2} = -1$ and $D1_{10,4} = 1$.

D2 is a MxL matrix where the rows correspond the edges in increasing order and the columns correspond to the triangular simplicies in created order. We fill in D2 in the following order. For the hth triangular simplex with endpoint (i,j,k),

**Algorithm 3** Constructing D1

**for** hth edge (i,j) in Edges **do**
  $D1_{h,i} = -1$
  $D1_{h,j} = 1$
**end for**

we set $D2_{h,i} = 1$, $D2_{h,j} = 1$ , and $D2_{h,k} = -1$. Suppose an edge (2,4,9) is the 10th triangular simplex. The corresponding entries in would be $D2_{10,2} = 1$, $D2_{10,4} = 1$, and $D2_{10,9} = -1$.

**Algorithm 4** Constructing D2

**for** hth edge (i,j,k) in Edges **do**
  $D2_{h,i} = 1$
  $D2_{h,j} = 1$
  $D2_{h,k} = -1$
**end for**

Each of the matrices, D1 and D2, are reduced to echelon form with the following algorithm. A pivot at a given column is given by the last nonzero entry in the column. We try to make all the columns not share any pivots. We loop over the columns and set check if the previous columns share a pivot with the current column. If they share a pivot, a multiple of the sharing column is subtracted from the current column such that they do not share pivots. This process is repeated until the current column does not share a pivot with any of the previous columns. Every time the matrix is modified the same operation is performed to an identity matrix, MxM identity matrix for D1 and LxL identity matrix for D2. After we finish reducing the matrix, we call the reduced matrix R1 and R2 and the corresponding modified identity matrices V1 and V2.

Finally, we can construct the main matrix that gives us the persistence homology of our protein. First we construct matrix B from the nonzero columns of matrix R2. Second we construct matrix Z from columns of V1 corresponding to zero columns of R1. The main matrix is then constructed by taking matrix B and appending columns from matrix Z which do not share pivots with columns of our main matrix until the total number of columns is equal to the number of columns

**Algorithm 5** Reduction of a matrix

**for** column in Columns of D2 **do**
  **for** pcolumn in Columns before column **do**
    **if** column share the same pivot as pcolumn **then**
      k=$\frac{column[pivot]}{pcolumn[pivot]}$
      column -= k*pcolumn
    **end if**
  **end for**
**end for**

in matrix Z.

**Algorithm 6** Construction of Main Matrix

Main = B
**for** column in Columns of Z **do**
  **if** number of columns in Main equals number of columns in Z **then**
    Exit Loop
  **end if**
  **if** column doesn't share pivots with columns of Main **then**
    Append column to Main
  **end if**
**end for**

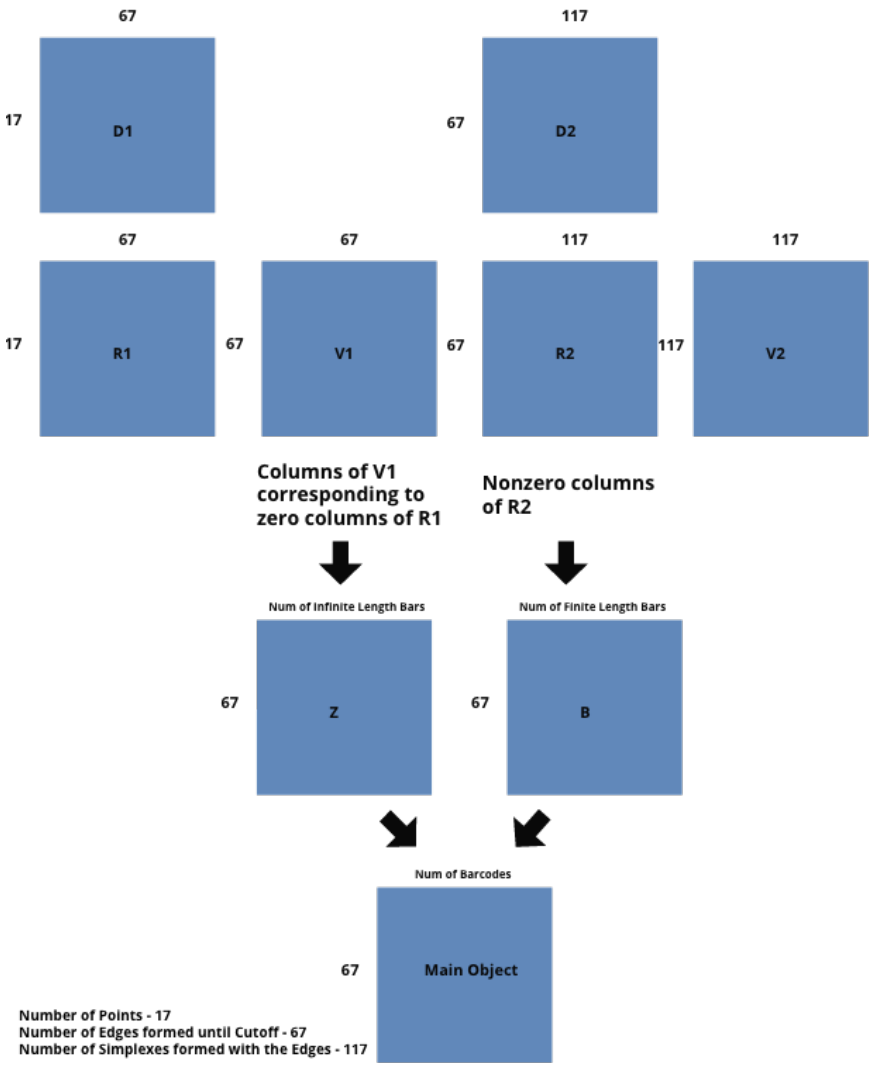



Figure 33: Here we have a diagram of the matrices that we have constructed and manipulated. The final product, the main matrix, is a MxH matrix where M is the number of edges we have considered and H is the simplex with persistent homology.

The columns in the main matrix that come from B correspond to the simplices which get created and filled in before our cutoff distance. The columns in the main matrix which come from Z correspond to the simplices that get created but not filled in before our cutoff distance.

From each column,we extract information about hole and it's birth and death from the column in the following manner. For each column, a hole is formed by the edges in the non-zero entries in the rows. A hole is born when the edge corresponding to the last non-zero row is added. If our column comes from B,

let d be the column of R2 that our column comes from. Then, the simplex is destroyed/filled in when the dth simplex is formed. If the column comes from Z, then the simplex is not destroyed within our cutoff distance. We notate this by denoting the death as $\infty$. We say that the simplex persists for $death - birth$.

We look at a column of the main matrix for our example with 17 points.The column is all zero except at rows 13,45,46,50 where it takes the values 1,1,-1,1 respectively, The indices of the non-zero rows correspond to the indices of the edges that belong to the hole. The values 1 and -1 correspond to the orientation of the edge. The longest edge of the hole tells us when the hole was formed. The death of the simplex occurs when the longest edge of the simplexes which fill in our hole is created. This hole/cycle corresponds to the smallest hole found in our data (Fig.34). This hole does not persist for a long time, indicating that it is not a significant feature in our data.

We look at another column of the main matrix for our example with 17 points.The column is all zero except at rows 8,9,10,11,13,14,20,22,25,27,31 where it takes the values 1,1,1,-1,-1,1,1,1,-1,-1 respectively. This hole/cycle corresponds to the largest hole found in our data with 11 points (Fig.34). This hole persists for a long time, indicating that it is a significant feature in our data.

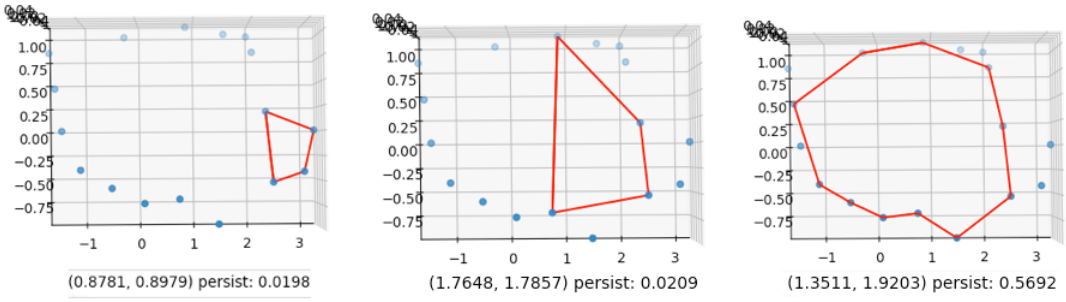



Figure 34: Here we have the 3 holes generated for our example with 17 points. These holes are representative of the features in the data. We see that the first two holes do not persist for a long time, making them less significant feature of the data than the last hole, which persists for a long time.

### 3.4.3 Persistence Homology Example

We introduced the theory behind persistence homology and the algorithm to compute it for a set of points. Now we apply persistence homology to our data set. We will first generate the barcode for a protein 1ux8 and see if the backbone chain, alpha helices, and betasheets are represented in the barcodes. We will also compare the features generated by persistence homology and the distance matrix.

We inspect holes generated by persistence homology of our protein 1ux8, a protein of length 118. At each point on the protein backbone, we search an $\epsilon$-radius around the point to connect edges of increasing length to form the $\epsilon$-Vietoris-Rips complexes. Below are a series of $\epsilon$-Vietoris-Rips complexes.

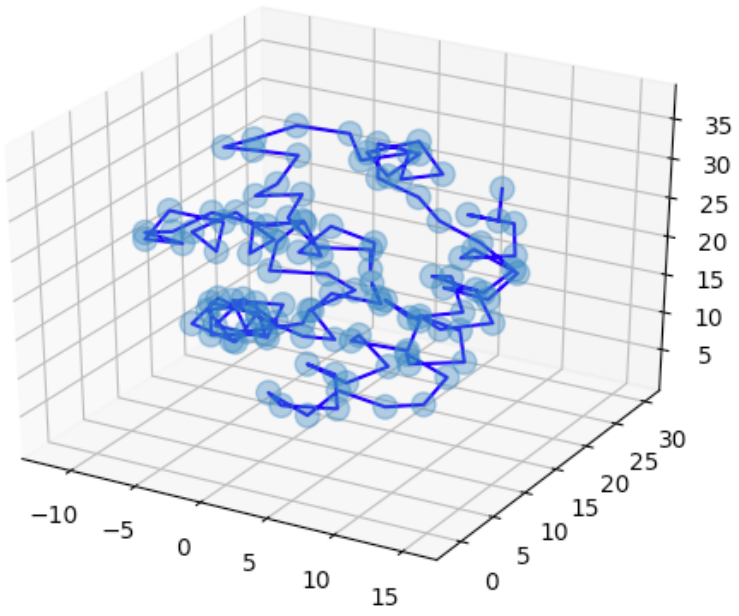



Figure 35: $\epsilon = 1$ A search is done at each point at increasing radii to connect the nearby points

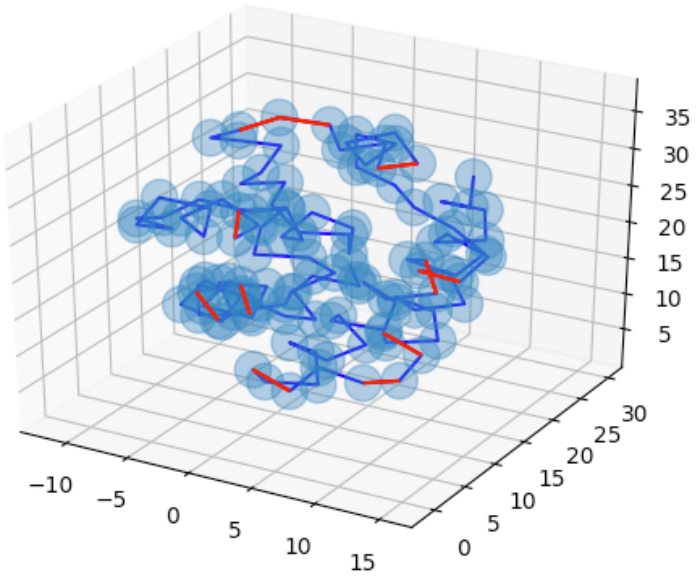



Figure 36: $\epsilon = 3.5$ The red edges indicate that the two points are less than 3.5 distance away,making them connected. As we increase the $\epsilon$, we connect more and more points, forming more simplexes.

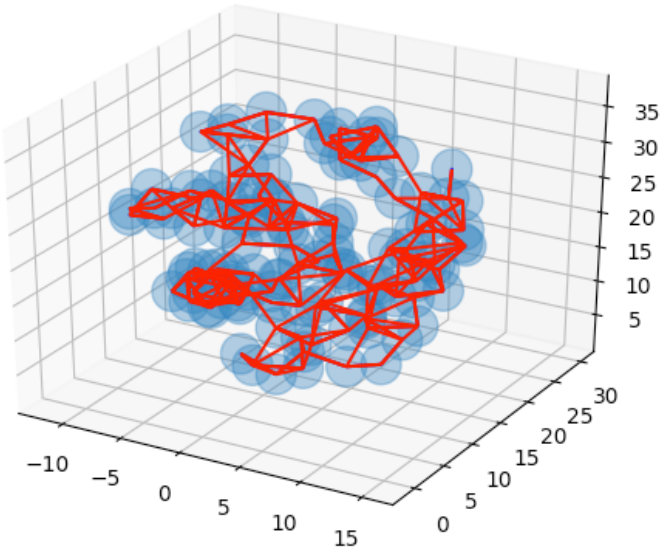



Figure 37: The general structure of the protein is detected as we increase when $\epsilon$ is around 6 Å

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, revealing not much about the structure of our protein. We set the cutoff distance at 6.5 Å.

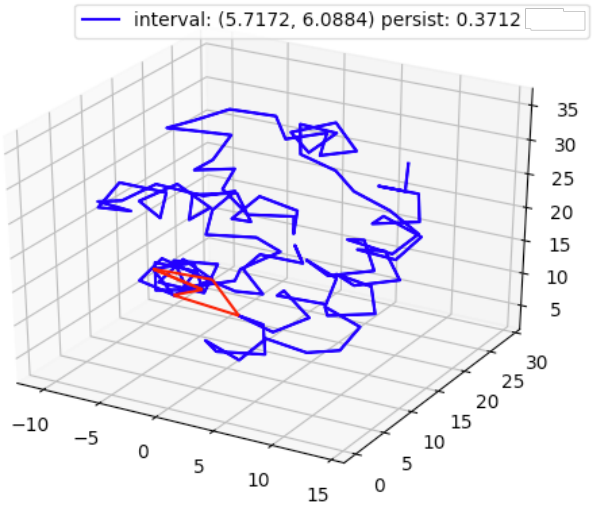



Figure 38: Persistent Homology detecting a small cycle. This small hole doesn't seem indicative of any important feature of our protein.

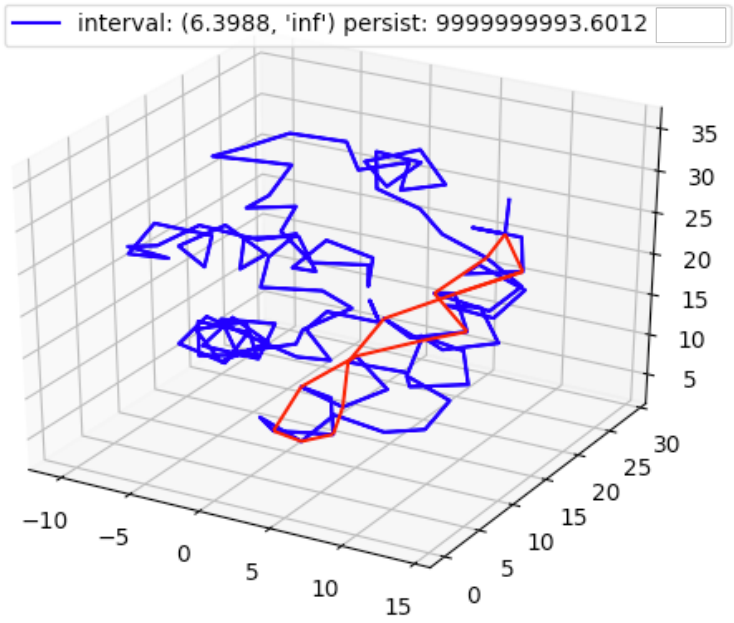



Figure 39: Persistent homology detecting an alpha helix. The hole is a significant feature of our data since the hole persists for a long time.

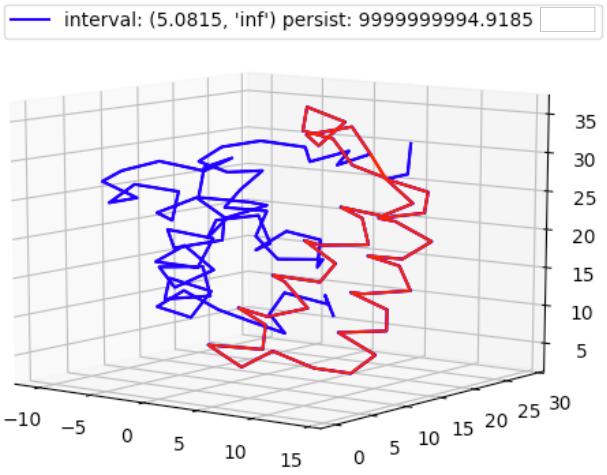



Figure 40: Persistent homology detecting a beta sheet. This hole is a significant feature of our data since the hole persists for a long time.

We inspect some key holes generated by the persistent homology to see if it captures the features of our data well. We see that persistent homology is able to detect the presence of alpha helices (Fig.39) and beta sheets (Fig.40). The persistent homology is almost on par with the distance matrix because it can represent alpha helices and beta sheets. However, the information of the birth and the death of the holes alone aren't enough to describe spatial relationships between multiple pairs of alpha helices.

### 3.4.4 Sparse Matrix

A series of matrices were used during the computation of the persistence homology. We note that in D1, D2, R1, R2, V1, V2, B, Z, Main matrices we see that the most of the matrix consists of zero elements (Fig.41). A performance evaluation of the homology algorithm showed that a lot of time was spent on manipulating the matrix object which was implemented in Numpy. This is because most matrices have dimensions which are well over 1000x1000.

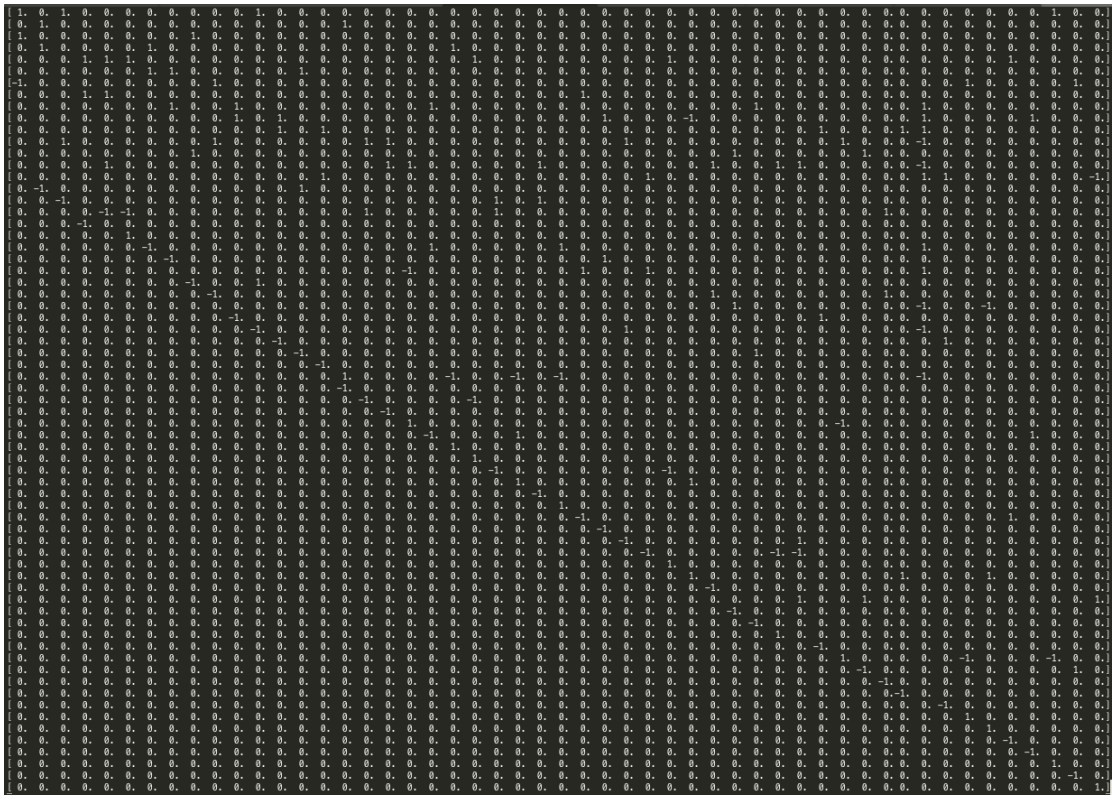


Figure 41: Here we see the main matrix of persistent homology for our collection of 17 points. We note that most of the matrix consists of zero elements

A sparse matrix was implemented so that we do not keep track of the zero values. For a NxM matrix, a list holds M dictionaries whose keys range over the row numbers less than or equal to N and whose values are the values of the nonzero elements. For a matrix given in (Fig.42), the corresponding sparse matrix would be (Fig.43).

$$\begin{bmatrix} 1.1 & 0 & 0 & 0 & 0 & 0 & 0.5 \\ 0 & 1.9 & 0 & 0 & 0 & 0 & 0.5 \\ 0 & 0 & 2.6 & 0 & 0 & 0 & 0.5 \\ 0 & 0 & 7.8 & 0.6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.5 & 2.7 & 0 & 0 \\ 1.6 & 0 & 0 & 0 & 0.4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.9 & 1.7 \end{bmatrix}$$

Figure 42: Here is small matrix where most of the matrix consists of zeroes. We will implement this into a sparse matrix to improve the computation time of the persistence algorithm.

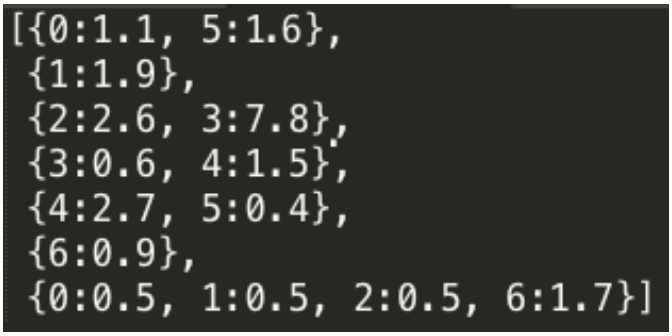



Figure 43: Here is the implementation of the matrix with zeroes as a sparse matrix. Although the computational benefits of using the sparse matrix over a regular matrix isn't huge for small matrices, the matrices used in persistent homology much larger than 1000x1000

When we add a new value at the (i,j) coordinates of the sparse matrix, we update our matrix using the jth column dictionary and the i key. Whenever an operation is performed that can change the ij th value of a matrix to zero, i key is removed from the jth column dictionary.

Implementing these changes to the matrix made the computation speed of the persistent homology for SCOP 1.55 10 times faster.

### 3.4.5 Backbone Aware Persistence Homology

The computation of persistent homologies for $\frac{1}{10}$ of SCOP 1.55, took around 15 hours. Upon inspection of the computation times, we noted that proteins that took longer than 50 seconds to compute accounted for about 11 hours of the 15 hours.

The reason why persistent homology had a difficult time was due to a improperly set cutoff value. Initially we set the cutoff value as the maximum length of the edges found in the protein backbone. This was because we wanted to include the edges of the protein backbone as part of the filtration of $\epsilon$ Vietoris-Rips simplicial complexes because it is a very important feature of the protein. However, some of the proteins had very long edges in the protein backbone chain, making the cutoff value very large and making our algorithm compute the persistent homology for all the edges in our protein. When we adjusted the cutoff value to be just around

6 Å, we noticed that beta helices along the protein were not being captured very well by the holes.

To remedy this issue, we multiply a weight of .25 on the edges belonging to the protein backbone. This would ensure that protein backbone edges that are reasonably small would get included as part of the filtration of $\epsilon$ Vietoris-Rips simplicial complexes. Placing this modification greatly sped up the speed of the computation of persistent homologies, computing $\frac{1}{10}$ of SCOP 1.55 from 15 hours to 2.6 seconds.

### 3.4.6 Persistence Images

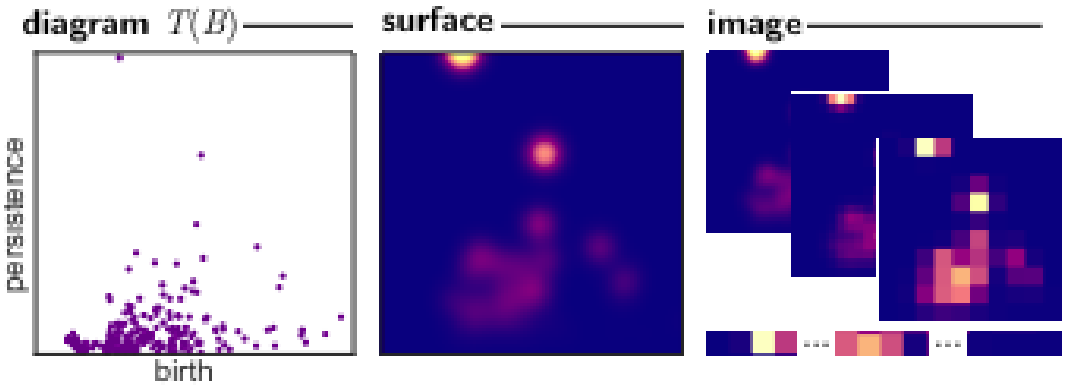



Figure 44: For a protein, a plot of the barcodes is constructed on a 2D plane where x-coordinate is the birth and the y-coordinate is the persistence. Then, a blurred resolution of the plot is taken as the persistent image.

The persistent homology of each protein is converted into an input feature called persistent images (Fig.44). For each bar in the barcode, we plot it on a 2-dimensional vector space with the x coordinate being the birth and the y coordinate being the persistence of the hole. For the simplices that persist infinitely, we plot the y coordinates well above the maximum value for persistence. After the plot is constructed, a 2D histogram is created of the data. The resolution of the histogram is 100x100 and the x interval and the y interval have been selected by looking at the x and y values of the plots for all proteins (Fig.45) & Fig.46)

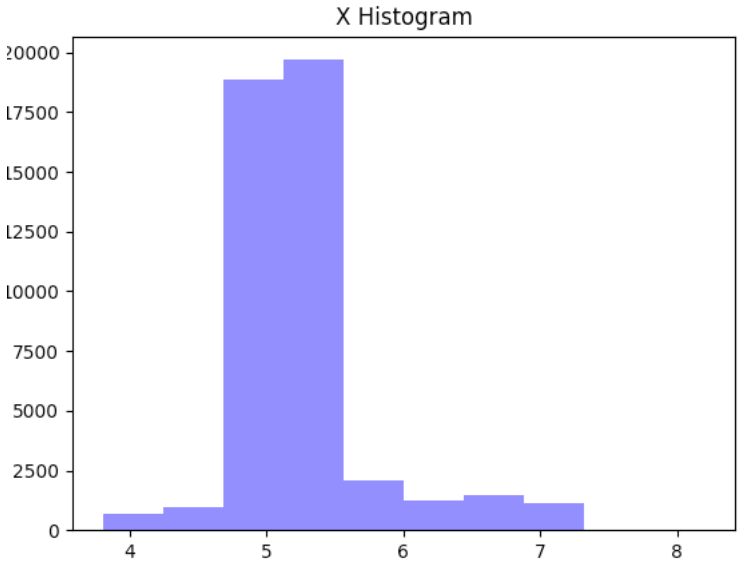



Figure 45: We inspect the x values, birth values, of barcodes of all the proteins prior to selecting an x interval for the 2D histogram.

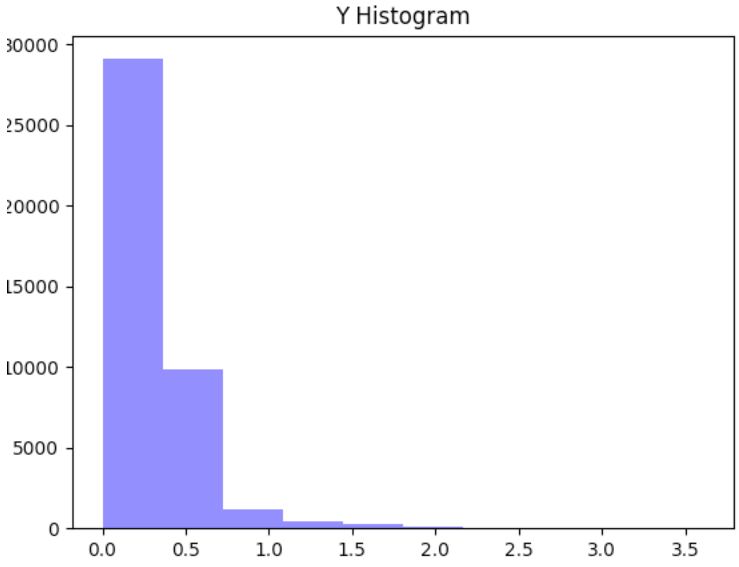



Figure 46: We inspect the y values, persistent values, of the barcodes of all the proteins prior to selecting an y interval for the 2D histogram.

This representation captures the essential information about each simplex as well as ordering them. The more persistent simplices are higher and the simplices that are born later appear towards the right. Fig.47 shows this plot for the protein 1ux8.

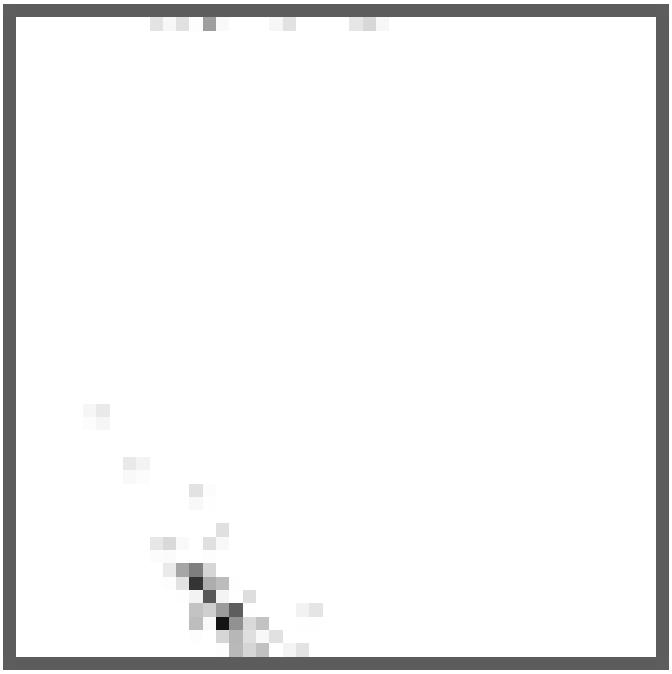


Figure 47: Here is the plot of the persistence homology for the protein 1ux8. The x axis is the birth of the hole and the y axis is the persistence of the hole. At the very top we plot the simplices with infinite persistence.

## 3.5 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.

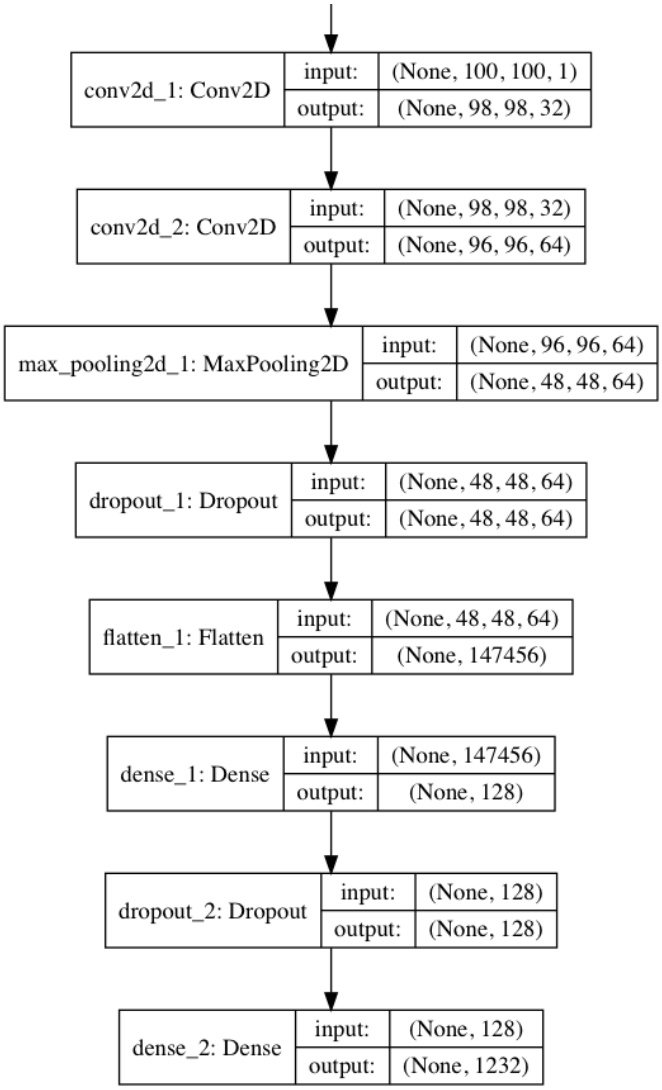



Figure 48: The architecture of a 2D deep convolutional network for protein fold classification. Convolutional layers were added as part of a traditional image classification network. Dense layers were added to help the network use the discovered features in the images. A dropout layer was added to make our network learn an ensemble of different models

This is a relatively simple network. However, for our task of classifying quarter million of proteins, it performs exceptionally well. 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 allows for the network to train at a reasonable speed.

We outline the hyper-parameters that were used for training our convolutional neural network. For all of the training, a batch size of 50 was used. Stochastic gradient descent with a learning rate of .01 was used. The labels of the protein folds were one-hot encoded as a vector of size 605 for SCOP 1.55 and 1232 for

SCOPe 2.07. For SCOP 1.55 the models were trained for 20 epochs. SCOPe 2.07, the models were trained for 30 epochs.

# 4 Results

Overall, the distance matrix out performs persistent homology as an input feature on both SCOP 1.55 and SCOPe 2.07. On SCOP 1.55 the distance matrix has an accuracy of 86% and the persistence homology has an accuracy of 62%. On SCOPe 2.07, the distance matrix has an accuracy of 90% and the persistence homology has an accuracy of 54%.

The distance matrix performs a lot better than the persistence homology because the distance matrix conveys the spatial relationship between the alpha helices and beta sheets while the persistence images only identify their existence. Specifically, persistence homology would have trouble distinguishing between proteins of Alpha/Beta and Alpha+Beta classes since these proteins are both composed of alpha helices and beta sheets but their spatial orientations are different.

We note that the distance matrix performs better on SCOPe 2.07 than SCOP 1.55. This is likely due to the increase in the amount of training data per fold. The number of classification labels increased 2 times but the amount of data increased 8 times between SCOP 1.55 and SCOPe 2.07.

Finally, the persistence homology performs worse in SCOPe 2.07 than SCOP 1.55. Although there are more training data per fold in SCOPe 2.07, there are more proteins that require more information about the spatial orientation of beta sheets and alpha helices that were added in SCOPe 2.07. This makes it more difficult for persistence homology to be accurate.

We found that the sparse distance matrix and the regular distance matrix perform similarly. The sparse distance matrix does take more epochs for it to reach the similar level of accuracy as the regular distance matrix.

| Method | SCOP 1.55 | SCOPe 2.07 |
|---|---|---|
| **Distance Matrix** | 86% | 90% |
| **Persistence Homology** | 62% | 54% |

Table 5

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

[5] Murzin AG, Brenner SE, Hubbard TJP, Chothia C. (1995). *SCOP: a structural classification of proteins database for the investigation of sequences and structures.* Journal of Molecular Biology 247:536-540.

[6] Shi JY., Zhang YN. (2009). *Fast SCOP Classification of Structural Class and Fold Using Secondary Structure Mining in Distance Matrix.* In: Kadirkamanathan V., Sanguinetti G., Girolami M., Niranjan M., Noirel J. (eds) Pattern Recognition in Bioinformatics. PRIB 2009. Lecture Notes in Computer Science, vol 5780. Springer, Berlin, Heidelberg

[7] (2008). *Protein Data Bank Changes Guide New Changes in Version 3.20* Nucleic Acids Research 42:D304-309. doi: 10.1093/nar/gkt1240

[8] Cang Z, Mu L, Wu K, Opron K, Xia K, Wei GW. A topological approach for protein classification. Molecular based Mathematical Biology. 2015;3:140–162.

[9] [Untitled illustration of convex hull]. Retrieved April 23, 2019 from http://mathworld.wolfram.com/ConvexHull.html

[10] [Untitled illustration of simplices]. Retrieved April 23, 2019 https://www.researchgate.net/figure/Simplices-from-0-simplex-to-3-simplex-11_fig7_290190525

[11] [Untitled illustration of Vietoris-Rips-complex]. Retrieved April 23, 2019 https://www.researchgate.net/figure/Computation-of-PH-for-a-point-cloud-using-the-Vietoris-Rips-complex_fig1_279633447

[12] [A simplicial 3-complex]. Retrieved April 23, 2019 https://www.wikiwand.com/en/Simplicial_complex

[13] [Barcode of a simplex on 4 points]. Retrieved April 23, 2019 https://www.researchgate.net/figure/Persistent-homology-of-a-sample-of-genetic-sequences-Barcode-and-simplicial-complexes_fig4_277022824