STRUCTURAL CLASSIFICATION OF PROTEINS USING IMAGE BASED MACHINE LEARNING

by

Daniel Franklin

Bachelor of Computer Science (Honours), York University 2003

A thesis

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Science

in the program of

Computer Science

Toronto, Ontario, Canada, 2019

Author's Declaration

Abstract

Structural Classification of Proteins Using Image Based Machine Learning

Daniel Franklin
Master of Science, Computer Science
Ryerson University, 2019

Classification of proteins is an important area of research that enables better grouping of proteins either by their function, evolutionary similarities or in their structural makeup. Structural classification is the area of research that this thesis focuses on. We use visualizations of proteins to build a machine learning class prediction model, that successfully classifies proteins using the Structural Classification of Proteins (SCOP) framework. SCOP is a well-researched classification with many approaches using a representation of a proteins secondary structure in a linear chain of structures. This thesis uses a novel approach of rendering a three dimensional visualization of the protein itself and then applying image based machine learning to determine a protein's SCOP classification. The resulting convolutional neural network (CNN) method has achieved average accuracies in the range 78-87% on the 25PDB dataset, which is better than or equal to the existing methods.

Acknowledgements

I would like to express my gratitude to my advisor Prof. Eric Harley for the continuous support of my MSc study and research, for his patience, motivation, enthusiasm, and immense knowledge. He has helped me in all of the research and writing of this thesis. I truly enjoyed our collaborating on the work for my MSc study.

Table of Contents

List of Tables

List of Figures

List of Acronyms

SCOP Structural Classification of Proteins

CNN Convolutional Neural Network

RNA Ribonucleic Acid

SVM Support Vector Machine

PSSM Position Specific Scoring Matrix

BLAST Basic Local Alignment Search Tool

MODAS MODular Approach to Structural class prediction

PSIPRED Predict Secondary Structure

GUI Graphical User Interface

PDB Protein Data Bank

HTML Hypertext Markup Language

Chapter 1

Introduction

My thesis is that image based classification of proteins can be used to predict the secondary structure class of proteins, as defined by the Structural Classification of Proteins database (SCOP). The method introduced in this paper is a novel approach, not tied by any other researchers, as far as I know. Proteins are an integral part of the basic building blocks of life. Proteins perform many functions in our cells, such as reducing the activation energy of a chemical reaction, facilitating the transportation of molecules in and out of cell walls, and communicating adaptations needed to survive in a changing environment by stimulating other proteins to be manufactured. Many diseases are the result of damage to proteins, where the damage hinders their ability to perform the above functions. To combat these failures, prescription drugs and medical treatments can be devised that will compensate for the inability of a protein to perform a particular task. Research into how proteins work, their role in life process and how they may fail that role is an important scientific challenge.

To aid in the understanding of a protein's nature, research into classifying and categorizing the ever growing number of different proteins is an important step. The Structural Classification of Proteins (SCOP) is a simple four-class system for distinguishing a protein by the amount and location of both alpha helix and beta-pleated sheet secondary protein structures in the protein. The SCOP classification allows a user to find proteins of related class, since it forms grouping for searching. The goal of this paper is to explore the biological underpinnings of proteins, understand how machine learning can be leveraged to solve a visual classification problem, and improve the current challenge of protein classification using the SCOP classes.

A protein is composed of a chain of amino acids, of which the human cell has 20 different kinds. Each amino acid has a similar structure but they all have a unique side chain that dictates its properties. Ribonucleic Acid (RNA) is translated into an amino acid chain by a cell's ribosomes. The initial state of the protein is a linear chain of amino acids, where each amino acid corresponds to a codon. A codon represents three ribonucleotides in the RNA. Immediately following this translation stage, the chain of amino acids starts to fold onto itself and create three dimensional structures of either alpha-helix, beta-pleated sheets, or coils. In 1951 Linus Pauling and Robert Corey [1] proposed these structures with the following definitions:

- The alpha helix, shown in Figure 1 below, is a rod like structure with a tightly coiled backbone that forms the inner part of the rod, and the side groups extend outward in a helical array.
- The beta-pleated sheet structure shown in Figure 2, has a fully extended strand shape instead of a coiled shape. Each strand can be then joined to other strands to form a sheet, where the strands can be parallel in direction to each other or antiparallel to each other.
- The random coil structures are shown in grey in both Figures 1 and 2. They do not have regular, periodic structures and are often found on outside of the proteins, where they may interact with other proteins and molecules.

The SCOP classification consists of four classes, which are: (a) all-alpha helix, (b) beta-sheet, (c) mixed alpha helix and beta-sheets and (d) segregated alpha-helix with beta-sheet. These classes represent the four main structural compositions that a protein may have.

Figure 1.1 depicts a protein from the all-alpha class, in literature the class may be shortened to read as ‘a’ or ‘all-a’. The all-alpha class contains proteins with mostly alpha secondary structures, but there may be a small amount of beta-sheet structures.

Figure 1.2 depicts a protein from the all-beta class may be read as ‘b’ or ‘all-b’, and like the all-alpha class is a protein that consists mostly of beta-sheet secondary structures, but may have a small amount of alpha-helix structures.

Figure 1.3 mixed alpha/beta class, shown in Figure 3 may be read as ‘c’ or ‘a/b’, and consists of a protein with both alpha-helix and beta-sheet interspersed randomly throughout the protein.

Figure 1.4 depicts a protein from the segregated alpha beta class, shown in Figure 4 may be read as ‘d’ or ‘a+b’, and consists of both alpha-helix and beta-sheet but they are not interspersed. The two structures of alpha and beta are split across the protein into two regions.

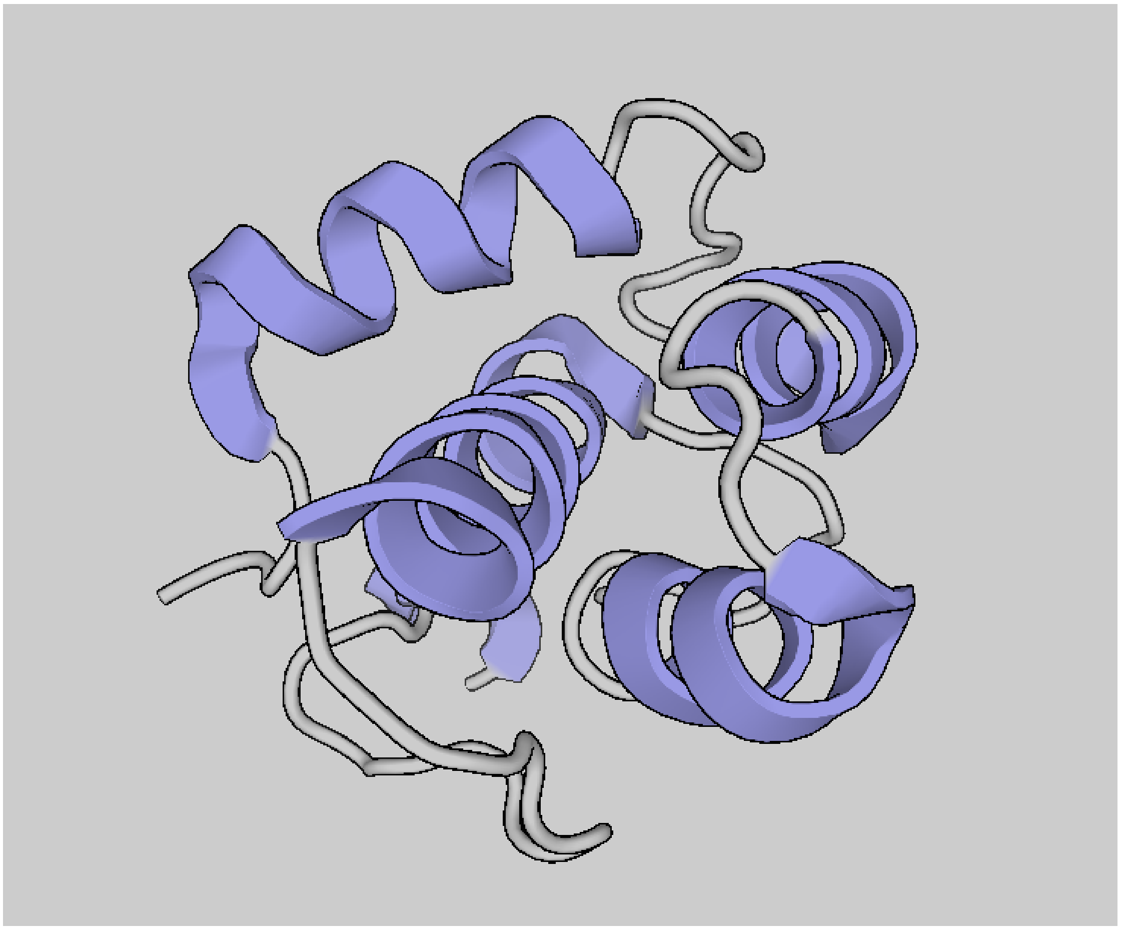

Figure 1.1: Protein 1FLIA chain A, taken from the image produced by PV program. The ribbon-like helices denote alpha-helices.

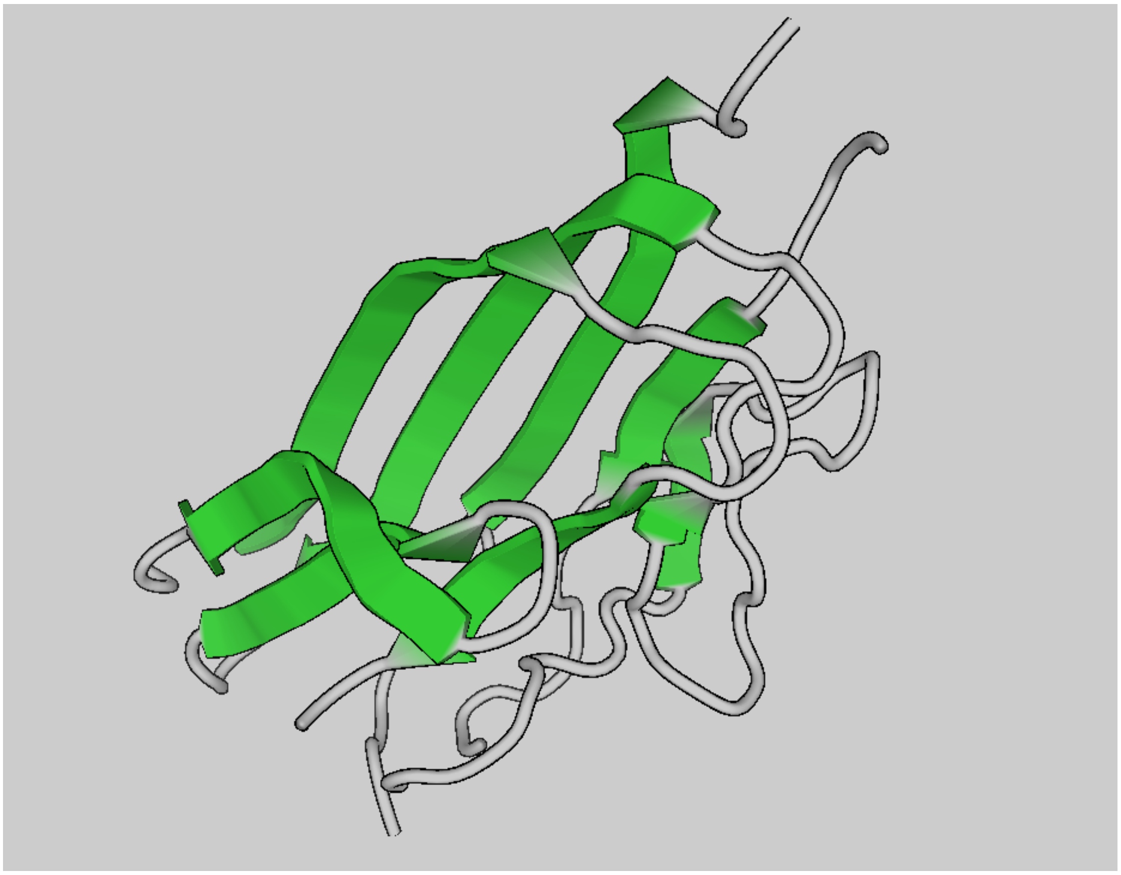

Figure 1.2: Protein 1C28, taken from the image produced by PV program. The flat ribbon-like arrows denote beta-strands and the arrows show its direction (direction in the chain is from an amino group to carboxyl group).

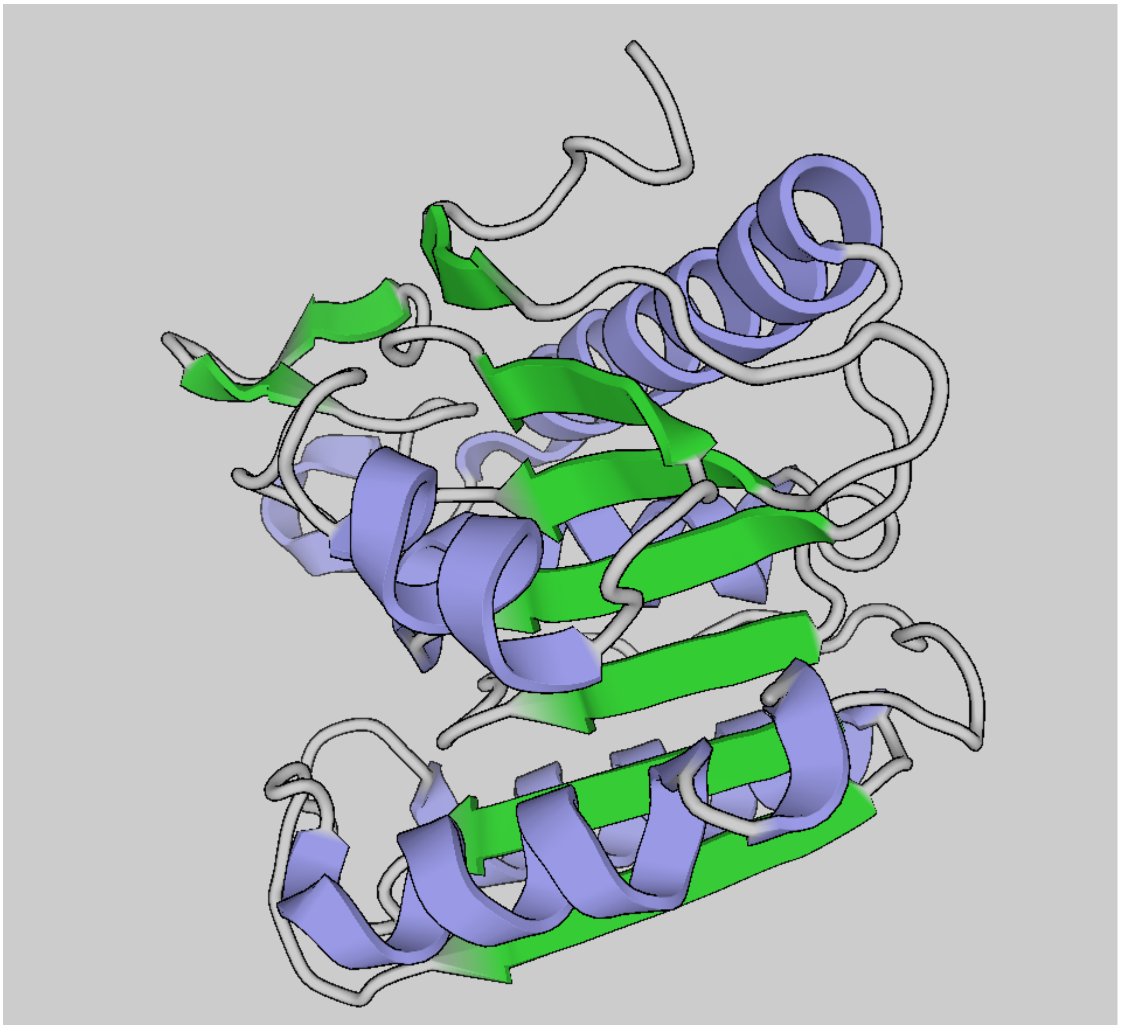

Figure 1.3: Protein 2R8BA chain A amino acid range 44-2466, taken from the image produced by PV program. This protein is an example of class c.

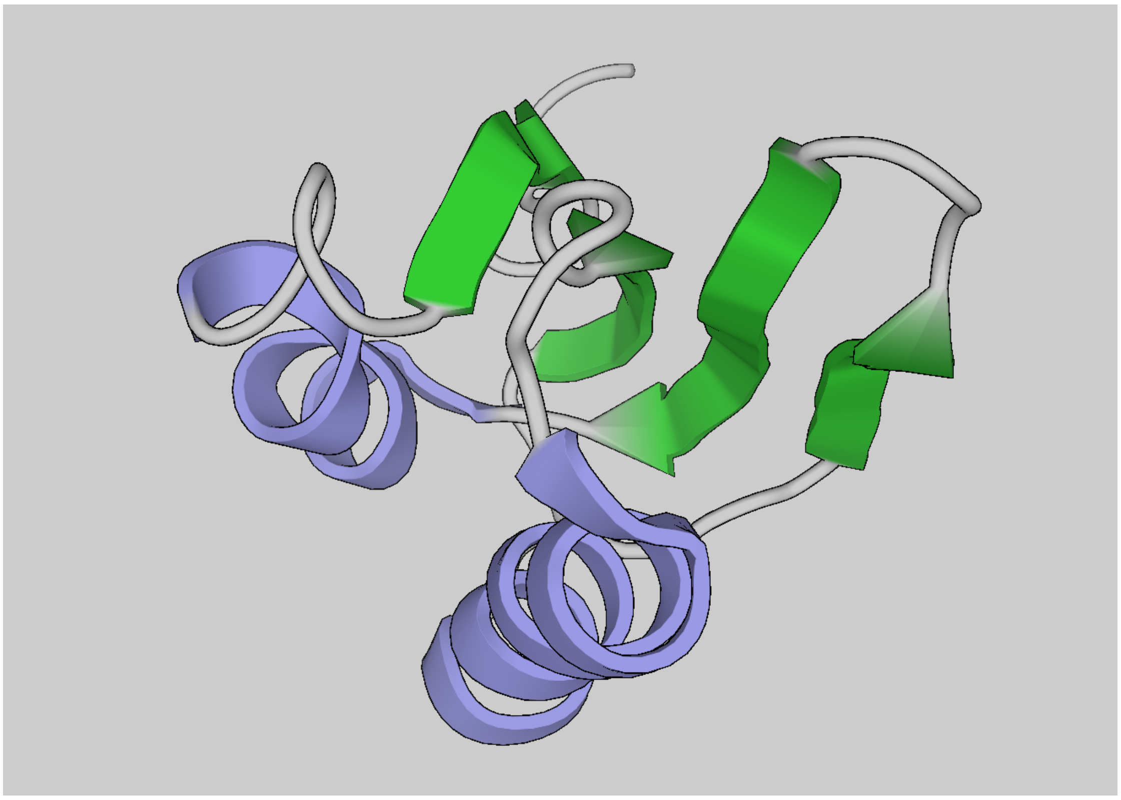

Figure 1.4: Protein 1ITP chain A, taken from the image produced by PV program. This protein is an example of class d.

The effort to apply bioinformatics and machine learning to improve the accuracy of SCOP classification of proteins is a well-researched field. I have reviewed many interesting and novel approaches to this problem which I present to you to consider. My supervisor and I have also thought through the problem and have tried a number of ideas and ultimately devised a method that uses image analysis of three dimensional visualizations of a protein's secondary structure, like the images in Figures 1-4, to classify them.

The process of visualizing the protein and capturing images leveraged an open source JavaScript application to visualize proteins called PV [16]. I modified this application to rotate a protein by 90 degrees in several directions, which creates six orthogonal images, like viewing the six sides of a die. These six images were then used to train a convolutional neural network using PyTorch that attained overall accuracies of 78-87%, which potentially improves upon other methods reported in the literature that range from 80% to to 85% (Table 2.2).

The proteins used in the training of the convolution neural network are from the 25PDB dataset which contains 1673 proteins which have a 25% or less similarity with each other. Six images were generated for each protein. From these images, 6008 images where used for training, 752 where used for validation, and another 752 where used for testing. The images used to build, validate, and test the convolutional neural network were randomly selected from the generated data.

1.1 Problem Background

At the time of writing this thesis The Protein Data Bank [18] houses 140,604 proteins, all of which share varying degrees of similarity in structure, evolution and function. One useful way of classifying a protein is by using the SCOP classification system, which is based on a protein's secondary structure composition. This thesis introduces a novel approach using image analysis of a protein to build a machine learning model that can automate the process of SCOP classification. Given a protein's visualization, we utilize a convolutional neural network to classify proteins according to the four SCOP classes.

1.2 Objectives and Proposed Methodology

1.3 Contributions

The main contributions in this dissertation are:

- Modification of an existing protein visualization program PV to produce orthogonal visualizations of a protein
- Implementation of a convolutional neural network trained on protein images to predict their SCOP classification.

1.4 Dissertation Outline

This dissertation is organized as follows:

- Chapter 2 presents background on the structured classification of proteins, as well as previous work in classifications.
- Chapter 3 discusses the implementations of image generation and convolutional neural network classification.
- Chapter 4 analyzes the accuracy of the implementation agains the reference datasets.
- Chapter 5 presents the conclusions from the above and suggests future work.

Chapter 2

Related Work

2.1 SCOP Classification

The Structural Classification of Proteins (SCOP) classification system comes from the SCOP database, which is a database of proteins curated by four different methodologies, one of which being by secondary structure distribution. The paper 'SCOP: a Structural Classification of Proteins database' [14] outlines the classification as

- Family, based on the amino acid sequence with similarity greater then 30% or with 15% similarity but with similar functions and structures.
- Superfamily, low sequence similarity but with similar structures and/or functional features.
- Common fold, when proteins 'have the same major secondary structures in the same arrangement and with the same topological connections'
- Class, relating to the secondary structure composition of the protein into one of four groups [14].
  - All-alpha, 'those whose structure is essentially formed by alpha-helices'
  - All-beta, 'those whose structure is essentially formed by beta-sheets'
  - Alpha/Beta, 'those with alpha-helices and beta-strands'
  - Alpha+Beta, 'those in which alpha-helices and beta-strands are largely segregated'

'The classification of proteins in SCOP has been constructed by visual inspection and comparison of structures.'[5] Research in computer science has worked on automating the classification classification process using bioinformatics and machine learning.

Dataset	All-Alpha	All-Beta	Alpha/Beta	Alpha + Beta	Total Proteins
25PDB	443	443	346	441	1673
FC699	130	269	377	82	858
D1184	251	258	199	477	1185
D8244	1744	1929	2357	2214	8244

Table 2.1: Shows the distribution SCOP classes and total number of proteins.

2.2 Benchmark Protein Datasets

Four datasets were used in this paper, the 25PDB dataset [10] was used to train and test with, while the FC699, D1184 and D8244 were used exclusively for testing the resulting trained model. 25PDB is an ideal dataset to train with as the proteins contained in the dataset have at most a 25% pairwise pairwise similarity. The FC699, D1185, and D8244 [23] are composed of proteins with 40% or less pairwise sequence similarity. Proteins with less similarity provide a better training set for a machine learning with the resulting model not being overtrained or overfitted.

Overfitting may occur when the training data does not have enough variability. Low variability in the training data may result in high accuracies with that training data, as the number of unique data samples would be very low. When trying to predict a new sample with an overtrain model that has not been seen yet an overfitted prediction model should perform poorly versus a model that has been trained with a diverse sample set.

Table 2.1 describes the distribution of proteins by SCOP class in each database that we used.

2.3 Results of using predicted secondary structures to predict SCOP Class

From the 2014 paper 'Novel structure-driven features for accurate prediction of protein structural class' [8] several known accuracies to methods that predict secondary structure are given. These results can be used as a benchmark for my results on the 25PDB dataset.

These methods do not use visualization of the protein to predict secondary structure, but rather they use the secondary structure sequence of the protein to devise a method to predict the SCOP classification.

Method	Class A	Class B	Class C	Class D	Overall
SCPRED [23]	93%	80%	74%	71%	80%
MODAS [13]	92%	84%	81%	68%	81%
RKS-PPSC [24]	93%	83%	86%	70%	83%
Kong and Zhang [8]	94%	87%	84%	74%	85%

Table 2.2: State-of-the-art prediction accuracies by SCOP class on 25PDB.

2.4 SCPRED

The SCPRED [9] method uses the predicted secondary structure to create statistical features to predict class. The 25PDB file has fields for protein name, amino acid sequence, predicted secondary structure, and SCOP class. For example, the protein 1A56 in the 25PDB has the amino acid chain:

- DADLAKKNNCIACHQVETKVVGPALKDIAAKYADKDDAATYLAGKIKG GSSGVWGQIPMPPNVNVSDADAKALADWILTLK

with a predicted secondary structure:

- CHHHHHHCCCCCCCCCCCCCCCCHHHHHHHCCCCCCHHHHHHHHHC CCCCCCCCCCCCCCCCCCHHHHHHHHHHHHHCC

and SCOP class all-alpha. In this notation, amino acid residues participating in an alpha helix structure are represented by H's, coil is represented by C's, and beta-sheet by E's. From the predicted secondary structure string, SCPRED creates features that encompass some significant aspects of the protein's structure. Nine features are then fed into a support vector machine (SVM) to train a model that will predict a protein's SCOP classification.

The 9 features used are the following:

- CV(L–G), a count of the number of LxxxG motifs found in the amino acid sequence, Leucine and Glycine are the amino acids respectively.
- NCount H6, normalized number of alpha helix segments built from at least 6 amino acids
- NCount H8, normalized number of alpha helix segments built from at least 8 amino acids
- CMV H1, composition moment vector
- NAvgSegH, normalized average length of alpha helix segments

- NCount E5, normalized number of beta-sheet segments built from at least 5 amino acids
- CV-E, percentage of beta-sheets (E characters) in a sequences of characters representing predicted secondary structure
- MaxSegE, length of the longest segment of beta-sheets
- NAvgSegE, normalized average length of beta-sheet segments

Note that the definitions above are paraphrased from Kurgan et al. [9]. The Last 8 are based on the predicted secondary structure. CV(L–G) is based on the amino acid sequence.

2.5 MODAS

Modas, or 'Modular prediction of protein structural classes from sequences of twilight-zone identity with predicted sequence' is a 2009 paper [13], where 'twilight-zone' references 'to over 95% of protein chains characterized by low, 20-25%, pairwise identity' [13] in the dataset. This research trained an SVM based on features from five different categories.

The categories of features used in MODAS [13] are:

1. Features derived from the amino acid sequence.
   - 12 features were computed
2. Features derived from the position specific scoring matrix (PSSM), which is a matrix built from the program PSI-PRED. PSI-PRED predicts the secondary structure from the amino acid sequence by using the Basic Local Alignment Search Tool (BLAST) algorithm to find similar sequences with known secondary structures. The PSSM [6] is a matrix that compares one amino acid chain against others and indicates how well conserved two chains are at any given position. This scoring is done via a substitution matrix that scores the tendency of one amino acid to be replaced by another.
   - From the PSSM, 65 features were computed
3. Features based both on the PSSM and the predicted secondary structure.
   - 179 features were computed

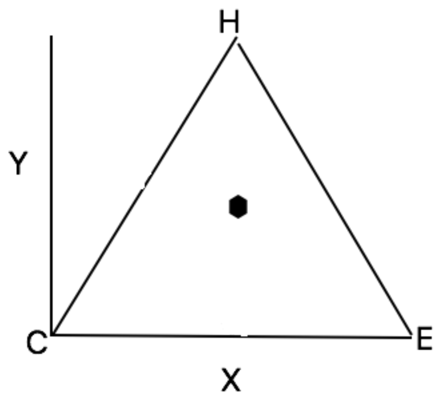

Figure 2.1: Diagram of chaos game.

4. Reused the features from SCPRED, which were are derived from the predicted secondary structure.
    - 73 features were computed
5. 'Novel features based on the predicted secondary structure which describe collocation of helical and strand segments' [13]
    - 50 features were computed

The feature set is too large to detail here, but an SVM was built for each SCOP classification using a subset of these features based on their performance. The result is a probability from each classifier that a protein matches the class of the SVM.

2.6 RKS-PPSC

The RKS-PPSC [24] method derived 24 features using three different methods. The features were the used as input to a Fisher's discriminant algorithm[12] to project a line to the predicted class, there method therefore does not you SVM. The features are derived using three methods after using a chaos game to transform the secondary structure sequence into a time series. The Chaos Game plots the three secondary structures from a given predicted secondary structure onto a three sided triangle inside a x-y graph.

Figure 2.1 of the Chaos game, where

> 'For each letter of the given secondary structure sequence, we then plot a point inside the triangle as follows. The first point is placed halfway between

the centre of the triangle and the vertex of the triangle corresponding to the first letter of the secondary structure sequence, and the i-th point is then placed halfway between the (i - 1)-th point and the vertex corresponding to the i-th letter. The obtained plot is called the Chaos Game Representation of the secondary structure sequence.' [24]

The rules from Figure 2.1 allows us to plot the secondary structure on the graph. The graph is then used to create two time series based on the x and y values which are used as input for driving features using the following methods.

1. Recurrence quantification analysis was used to derive 8 features, but how it was done was omitted from the paper.
2. K-string based information entropy was used to calculate p(H) and p(E) which represent the probability that either H or E will be present after a K length string of secondary structures.
3. Segment-based analysis was used, which compresses down a given secondary string to continuous strands of E's or H's, ignoring C's. Thus, for example, CCEEHH would become EH. After the compression phase, they calculate the probability of two consecutive Hs, and the probability Pt = 1 - (probability of two consecutive Hs) - (probability of two consecutive Es).

2.7 Distance Based Features

In the paper, 'Novel structure-driven features for accurate prediction of protein structural class.' [8], Kong and Zhang derive their features from the distance relationships between the predicted secondary structures. This method hopes to extract information based on the spatial, three-dimensional nature of proteins and their secondary structures. Some 27 different features are extracted and used to train an SVM. Please refer to the paper itself to see the details of the features and how they are derived. As shown in Table 2.2, the accuracy they achieved with their feature-based approach is 85% overall, which is a little better than previous attempts of others.

2.8 Using Euler Angles to Rotate A Three-Dimensional Body

The book 'Advanced Animation and Rendering Techniques—Theory and Practice' [22] describes the Euler angle as the 'most popular parametrization of orientation space'. The text [22] also states a ‘general rotation is a sequence of rotations round three mutually orthogonal coordinate axes fixed in space.’

$$R_x(\psi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\psi & -sin\psi \\ 0 & sin\psi & cos\psi \end{bmatrix} \tag{2.1}$$

$$R_y(\psi) = \begin{bmatrix} cos\psi & 0 & sin\psi \\ 0 & 1 & 0 \\ -sin\psi & 0 & cos\psi \end{bmatrix} \tag{2.2}$$

$$R_z(\psi) = \begin{bmatrix} cos\psi & sin\psi & 0 \\ sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

Equations (2.1) (2.2) (2.3) from [21] represents the three matrices to compute the axis of rotation in the x, y and z plane.

By applying a matrix calculation on a three-dimensional body’s orientation by any or all of the matrices of Figure 6 one can apply a rotation of a given angle. For example, rotating a body 90 degrees in the x axes would require the calculation of Rx and multiplying the Rx matrix with the current orientation matrix, then updating the orientation of the body with the new orientation that is a result of applying the 90 degree x rotation. One can rotate the body in all three x, y, z directions by applying the Rx, Ry and Rz matrices to the current orientation.

2.9 Motivation

During my research on this topic, I carried out a number of experiments where I used features derived from the known or predicted secondary structure chain. Ultimately, I concluded that the linear chain of predicted or observed secondary structure could not accurately capture the relationships that differentiate class c from class d in the SCOP classification. I hypothesized that I would have to take into account how a protein’s secondary structure is folded in to a three-dimensional molecule.

2.10 Training SVM with 25PDB Features

The principle data used for this thesis is derived from the 25PDB dataset that can be downloaded from the Biomine Kurgan Lab’s website [20]. The 25PDB.csv file contains the protein name, amino acid sequence, predicted secondary structure sequence and class. Also available for download is the 25PDB.arff file which contains 9 features described in Chapter 2 of this paper in the SCPRED section. Each protein found in the csv file has a corresponding line in the arff file with the features data and SCOP class. This data can then be used to create a supervised machine learning algorithm.

With the arff file it is easy to recreate the experiment to see how the features on their own perform using a Support Vector Machine (SVM). I wrote a matlab program and used the Weka GUI to test the accuracy of the SVM model based on the features, and produced an accuracy between 74% and 80%. The results are as follows.

SVM accuracy: 80%
Kernel function: Linear
Multiclass method: One-vs-One
5-fold cross validation

a	b	c	d	classified as
408	2	13	20	a = a
8	360	15	60	b = b
14	6	250	76	c = c
29	47	47	318	d = d

Table 2.3: Results with Matlab using SVM one-vs-one.

SVM accuracy: 74%
Kernel function: Linear
Multiclass method: One-vs-All
5-fold cross validation

a	b	c	d	classified as
414	2	13	14	a = a
10	396	20	17	b = b
25	18	258	45	c = c
46	131	87	177	d = d

Table 2.4: Results with Matlab using SVM one-vs-all.

SVM accuracy: 78%
Kernel function: ploy
Multiclass method: One-vs-One
5-fold cross validation

a	b	c	d	classified as
411	2	9	21	a = a
8	364	11	60	b = b
18	3	229	96	c = c
34	50	51	306	d = d

Table 2.5: Results with Weka using SVM one-vs-one.

2.11 Brute Force

After looking at the problem for a while and understanding the nature of the SCOP classification, the problem seemed trivial and that a brute force approach should be possible. Given a string of secondary structures composed of zero or more coil, C, alpha-helix H, and beta-sheets E, a simple calculation should determine the classification. I

Classification	A	B	C	D
A	406	5	30	5
B	6	356	53	25
C	38	7	83	218
D	29	85	223	104

Table 2.6: Confusion matrix of results from brute force prediction of secondary structure.

implemented the following simple algorithm,

- If the string contains only characters C and H, then it belongs to the all-alpha (a).
- If it contained only C and E, then it is all-beta-sheet (b).
- If the string contains C, H and E characters, and the H regions and E regions oscillate then it is a mixed class: a/b (c).
- If the string contains all three C, H and E regions, but the E and H regions to not oscillate meaning either there is a number of E regions followed by H regions or H regions followed by E regions then the class is segregated a+b (d).

The results of this approach were not satisfying, and in the strictest implementation the accuracy was only 48%. Including rules such as considering a minimum length of E and H regions to be approximately greater than 20% of the length of the overall string, meaning if the string was 10 characters long then only regions of 2 or more Es, or two or more Hs would be considered, increased the accuracy to 57%. The confusion matrix for the brute force result are captured in Table 2.6.

The motivation behind this was that there may be small fragments of secondary structure that were insignificant to determining class. The results matched up with the literature that acknowledges the difficulty in distinguishing between c and d classes.

The poor results showed that there was some pattern to the secondary structure that was difficult for the strict rules of brute force algorithm to account for, where in some cases a protein should be classified as a mixed alpha+beta, but is instead classified as a segregated alpha/beta.

2.12 Machine Learning with Visualization

The problem of determining rules that are not obvious to a human's intuition makes for a good fit for a machine learning approach, training machine learning algorithms with exemplars of the classes allowing the model to learn the appropriate relationship rules to produce the correct prediction. The question then involved what data to train with? While the feature approach had been tried many times, I wanted to work directly with the data and see if a model could be built from the source material, either the secondary string itself, or from a visualization of the protein.

The visualization of a protein as the data for training was appealing because the visualization encompasses the three-dimensional locations of the secondary structures. It isn't apparent from the linear chain of C, E, and H where one particular H strand would be located in the actual protein. A secondary structure string which appears to be segregated, with some Hs followed by some Es, may actually have some regions that fold into itself and result in a mix of H structures and E structures.

To capture images a number of protein visualization programs were tried: JMol, JSMol, and ultimately the choice of the PV application was introduced to capture an image of the protein. PV is built using JavaScript, with which I am familiar, and gives a web frontend which I thought could be leveraged to host the eventual result of the classification on the web.

Initially, the image capturing was implemented in such a way as to extract the best orientation of the protein. The best visualization of the protein involved counting the structures that were visible given the current rotation of the protein and attempting to maximize the number of different structures. The two structures alpha-helix and beta-sheets are each assigned a colour, and for each image the number of pixels corresponding to each type of structure was counted and multiplied together. The heuristic measure of goodness of a visualization that I implemented was:

- Count the alpha red pixels, countRed
- Count the beta blue pixels, countBlue
- Measure = max(countRed, countBlue, countRed * countBlue)

With the measure heuristic, it was then possible to build a hill climbing algorithm that would search the possible orientations of the protein. By maximizing the measure I was able to find the orientation that displays the greatest amount of secondary structures.

This best-image approach was later abandoned in favour of a brute force method. Realizing that using several orientations for visualizations of each protein might be better than using a single 'best' orientation which might not be ideal anyway, I instead generated a number of images from each protein, letting the learning algorithm extract meaning from the images. We chose to generate six images per protein, which I detail in the implementation data generation section of this paper.

2.13 Image Prediction With Machine Learning

This thesis employs image analysis to learn how to classify an image of a protein. Convolutional neural networks (CNN) are an ideal choice, as they are designed to accept the large volume of pixel data contained in an image. CNN's take a three-dimensional vector, or tensor, consisting of the resolution of the image, currently I use a 512 x 512 image, by the colour resolution RGB values as the depth. The resulting tensor has 512 x 512 x 3 = 786,432 values, which would be very large as the input for a neural network.

The CNN will reduce the size of the neural network down to a more reasonable input to a neural network, during each layering. The deeper the network the more higher level features like secondary structures may be learned [11]. The basic algorithm of a CNN as described in [2] is to utilize a sliding filter along the three-dimensional tensor, one colour dimension at a time. This filter is a two dimensional array, typically 3 x 3 which pans across the image. Each filter is then a small fragment of the image, and they are collectively returned from the convolutional layer. The result from the convolutional layer is then then passes to a pooling layer which will downsample the values in the filter and transform the 3 x 3 into a 2 x 2 filter, which would reduce the overall size of the tensor. The tensor will be processed by a number of CNN layers until eventually the output is fed into a linear layer of a neural network to output the classification.

Chapter 3

Experimental Design

In this chapter, I describe my approach to solving the problem, along with potential weaknesses of my approach.

The implementation to classify proteins based on the SCOP classification entails three main software processes.

- Generation of data, in the form of images of proteins
- Processing the generated data to prepare it for training
- Training a convolutional neural network to classify the png images of proteins

3.1 Generation of Data

The generation of data is performed with the utilization of a JavaScript/Python protein visualization program called PV [16], which has freely available source code. PV is capable of downloading a PDB file associated with a given protein, from the Protein Data Bank website [18] and visualizing it in a web browser. I augmented the JavaScript-based web browser client to enable the extraction of images and the rotation of the visualization.

3.1.1 Client Implementation

The client side is responsible for rendering a protein, rotating it 90 degrees six times exposing the proteins orthogonal views. I wrote code to capture image data corresponding to the perspectives of each die's face and to send the data to the server to save it as proteinName1...6.png.

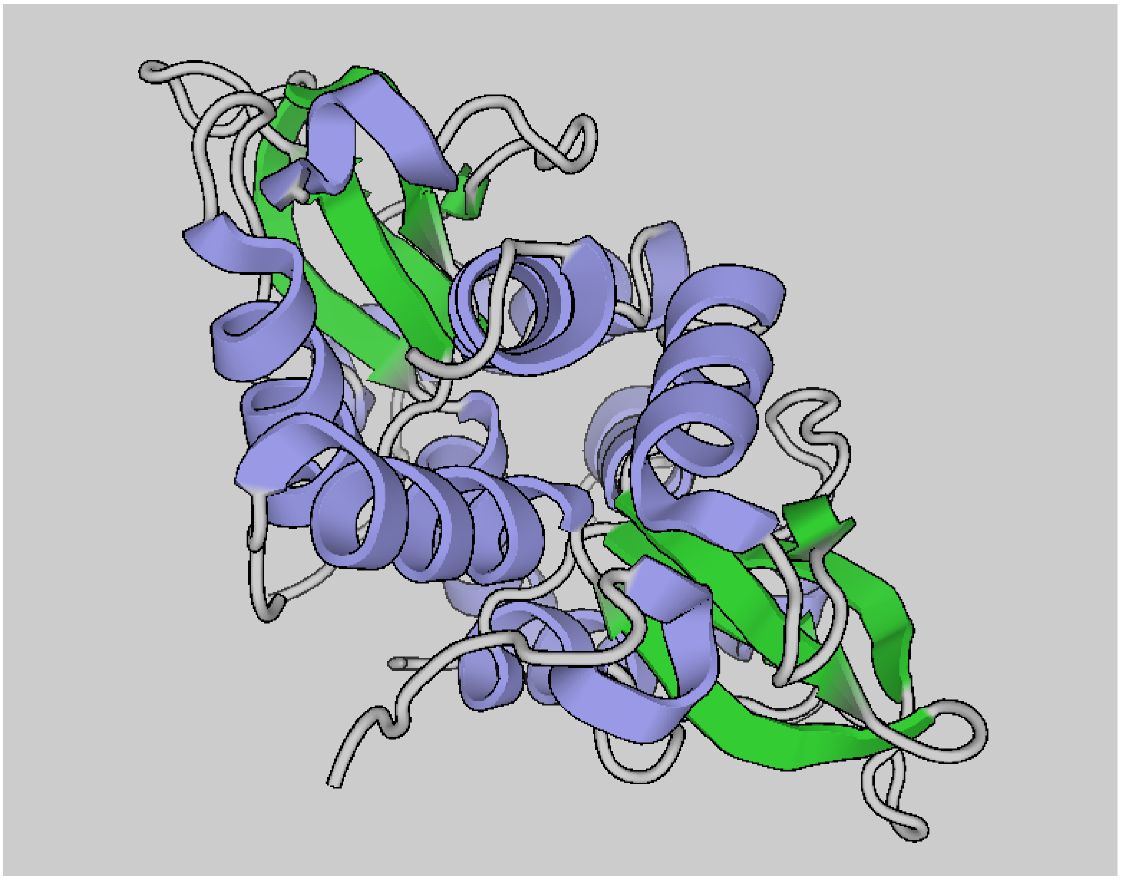

Figure 3.1: Protein 1O50 chain A range 1-1451, with an orientation of [0, 0, 0].

Figures 3.1 to 3.6 are examples of the output of the image generation phase, taken of protein ‘1O50’.

The client then requests the next protein to process and the loop repeats until no new protein is available.

The identifiers of the proteins listed in the dataset files have three components: the protein name as a four character string, an optional selected chain, and an optional index indicating a selected range into the chain. Examples of proteins found in the 25PDB file are 1A6M_ and 1AIPH:3-53.

- Protein 1A6M_ will be rendered by the PV program without modification since the underscore ‘_’ after the four letter identifier means ‘use the full protein’, through the load commands:

var go = viewer.cartoon(’structure’, s, //load the cartoon structure of the PDB file after removing unwanted elements described by the optional chain name and indexes

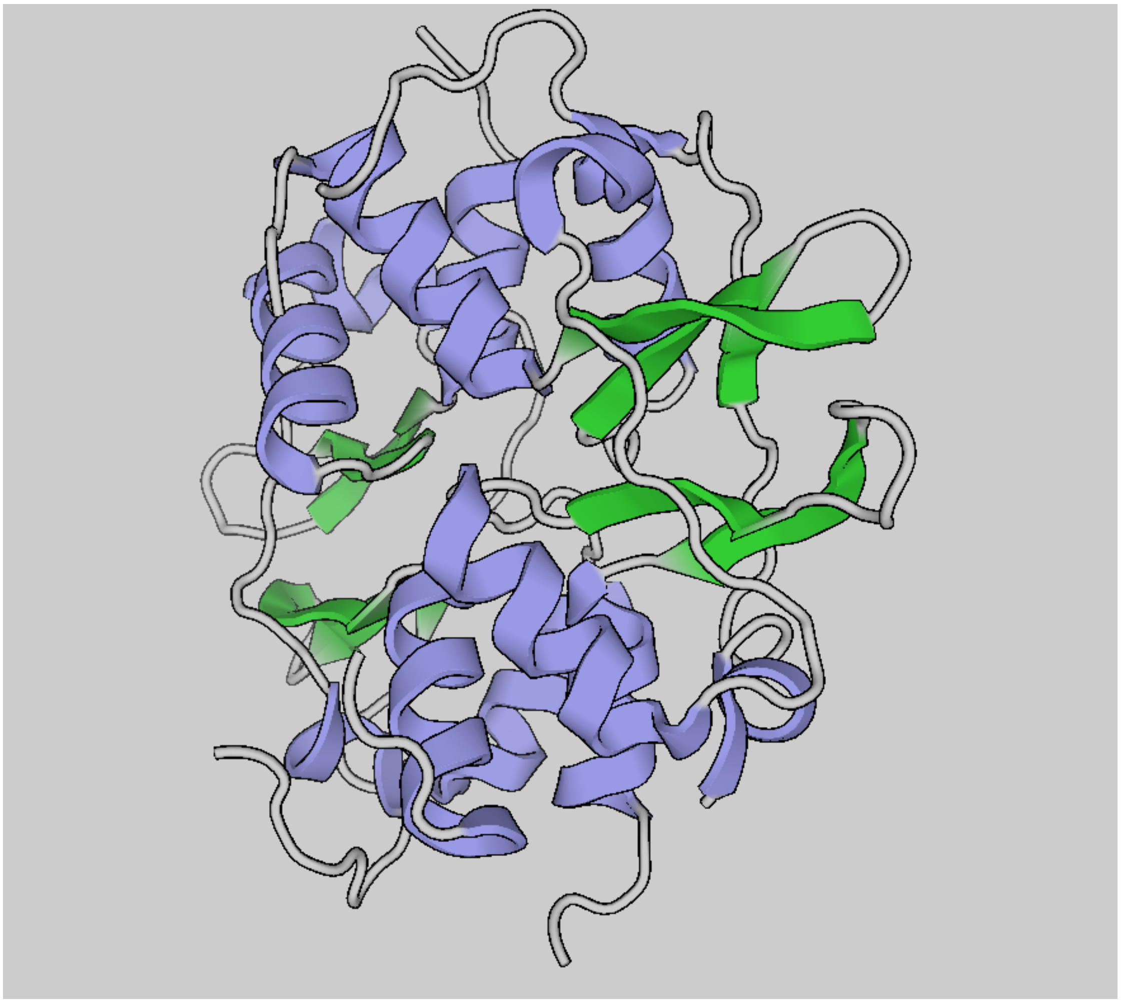

Figure 3.2: Protein 1O50 chain A range 1-1451, with an orientation of [90, 0, 0].

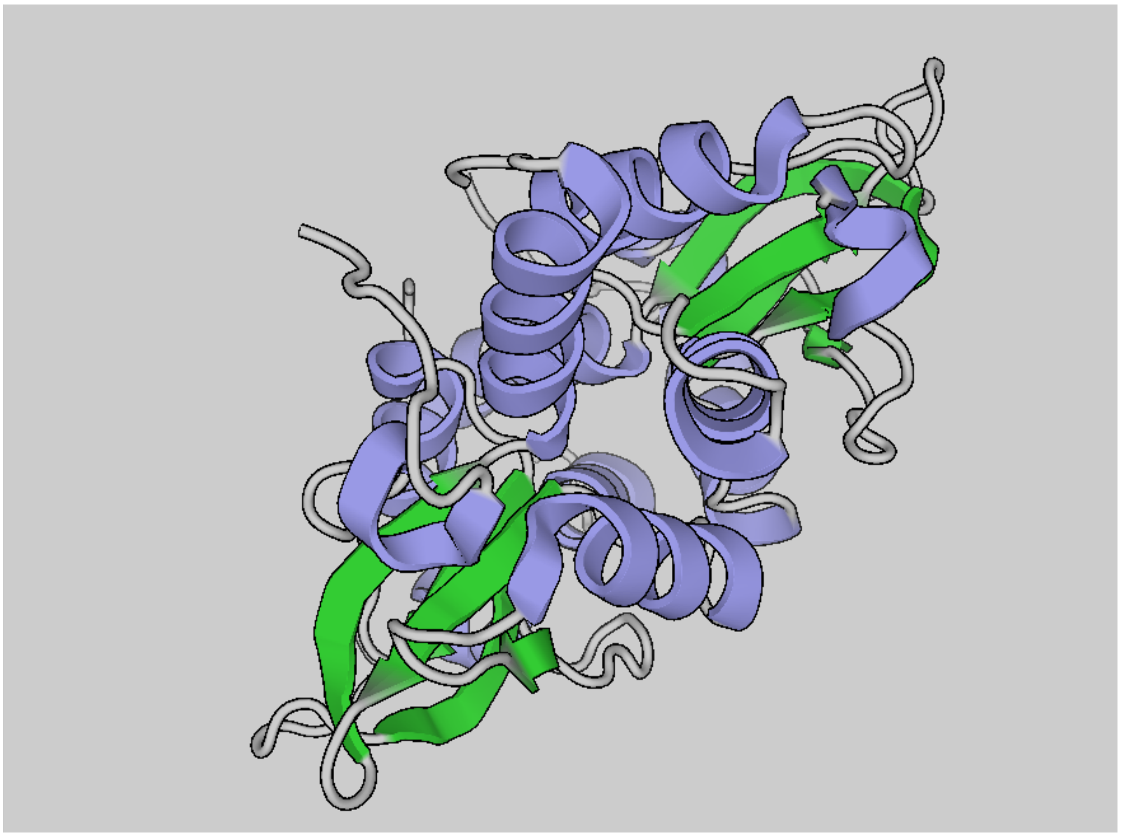

Figure 3.3: Protein 1O50 chain A range 1-1451, with an orientation of [180, 0, 0].

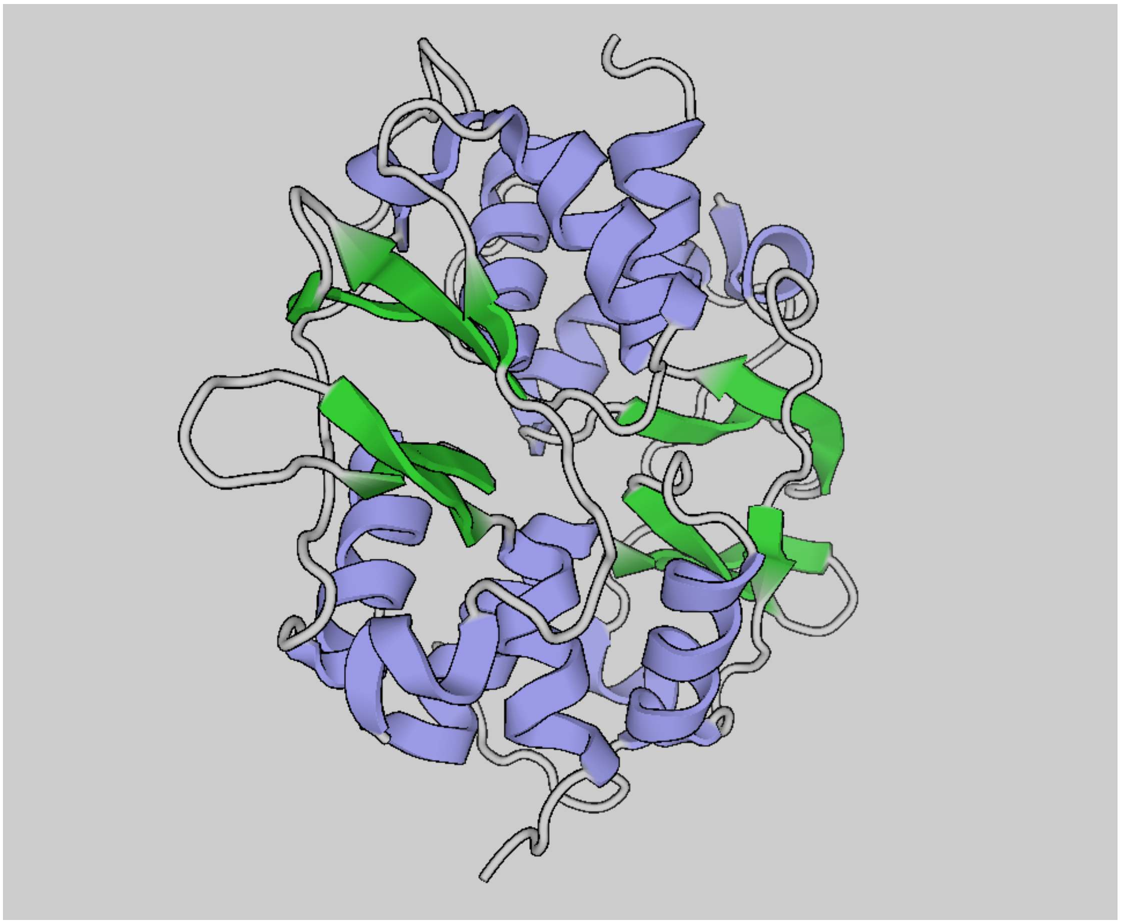

Figure 3.4: Protein 1O50 chain A range 1-1451, with an orientation of [270, 0, 0].

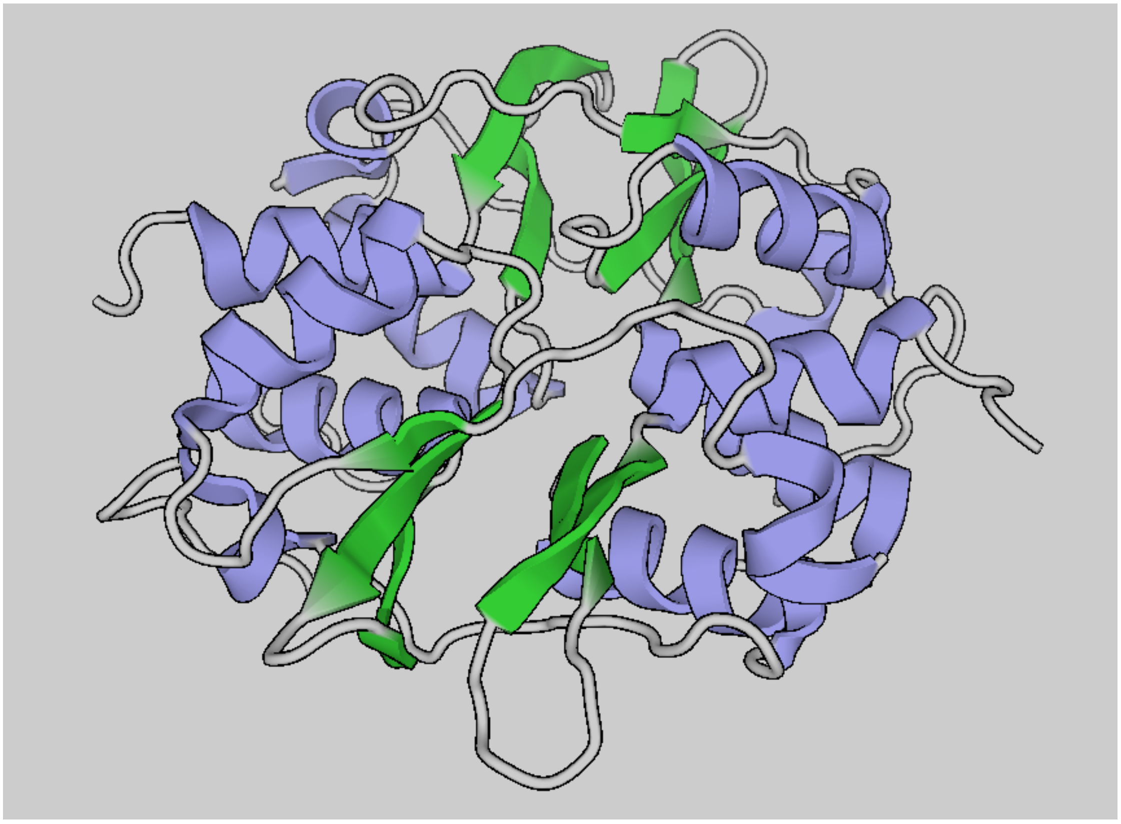

Figure 3.5: Protein 1O50 chain A range 1-1451, with an orientation of [0, 90, 0].

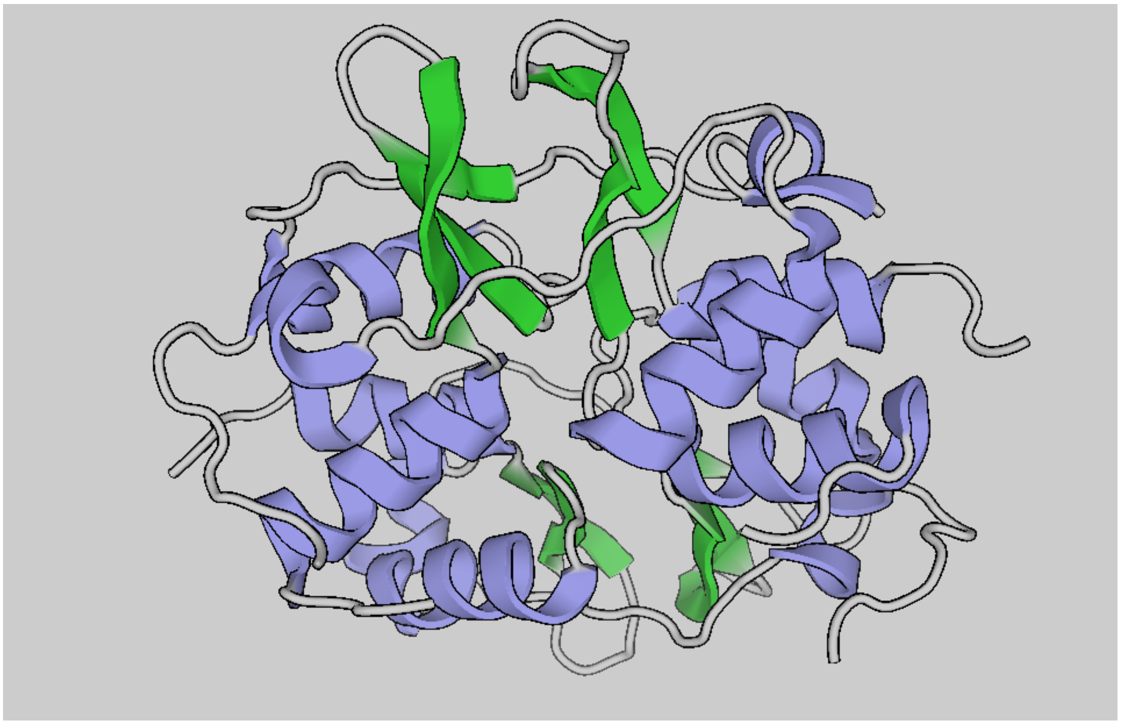

Figure 3.6: Protein 1O50 chain A range 1-1451, with an orientation of [0, 270, 0].

color : pv.color.ssSuccession(), showRelated : '1',); // set the orientation and colour of the different secondary structures var rotation = pv.viewpoint.principalAxes(go); viewer.setRotation(rotation); viewer.forEach(function(go) go.colorBy(pv.color.bySS());); viewer.autoZoom();

- The effect of the above commands will load a given protein as a cartoon rendering by the PV's viewer class. Where cartoon displays the secondary structures depicted in Figures 1-4 and 7-12.

- Protein 1AIPH:3-53 needs modification before it can be visualized, since after the first four letters identifying the protein, there is an 'H' meaning 'use only the H-chain', and then a numerical range, meaning 'use only residues 3 to 53'. Thus, 1AIPH:3-53 represents a small fragment of the 1AIP protein. Programatically all chains other then the H chain are removed from the structure, and also the indexes that don't fall within the range 3-53 are removed. After that, the structure can be loaded and rendered by the PV viewer class in cartoon style that isolates the region of the protein to be viewed.

The PV program was then modified to perform rotation of the protein in 90 degree increments using Euler angles with the following six orientations: [0,0,0], [90,0,0], [180,0,0], [270,0,0], [0,90,0], [0, 270, 0] JavaScript has a single thread design, which requires my program to toggle between the UI code and the image rotation code. I use the setInterval function for this which will call a designated function periodically. This interval execution of a function is used to perform state changes to the visual model, while interleaving image capture work.

```
loop() {
   if (this.index == this.nextIndex) {
      //increment the next index
      ++this.nextIndex;
      //record the image
      this.model.sendToServer(this.protein);
   } else if (this.index < this.moves.length) {
      //calculate next move
      var currentMove = this.moves[this.index];
      //move the model to the next orientation
      this.model.rotate(currentMove[0], currentMove, currentMove[2]);
      //increment next move
      this.index = this.nextIndex
   } else {
      //protein images captured end loop and get next protein
      clearInterval(this.loopId);
      this.callBackFunction();
   }
}
```

The set interval will call the loop function 13 times:

- 6 times when the index variable is equal to the nextIndex – in this phase the protein has been rotated and an image is sent to the server.
- 6 times when the index is less then the moves array size – in this phase the protein is rotated to the next move,
- Finally, when all rotations of the protein 3D model have been made, the function loops by calling helper function to get the next protein.

3.1.2 Server Implementation

The backend server is written in python and is responsible for parsing the protein dataset to determine which proteins have not been generated, in png format. The program does so by comparing the set of proteins to be rendered in a dataset, after it has been parsed, to the proteins already persisted, as png found in the designated folder location by the proteins predicted class. The python server then accepts POST requests by the client to save a protein's image as a png into the appropriate class folder (a, b, c or d).

The dataset files (25PDB.csv, D1185.xls, D8244.xls, and FC699.cvs) list each protein and which SCOP classification it represents. The client requests the next protein, and if not loaded, the python code loads all names of all the proteins already captured as images. Along with the known image file, the dataset file is parsed, with a file format of: proteinName, amino acid sequence, secondary structure sequence, class [a, b, c, d]. I am only interested in the proteinName and the class, so index 0 and 3 of the comma delimited list. To derive the next protein simply loop over the dataset proteins, and if it is not in the known protein list and return it to the client.

3.2 Processing the Data

The protein images have been generated from any one of the datasets [25PDB, D1185, D8244, and FC699] and are now organized by class in folders named

- allAlpha
- allBeta
- mixedAlphaBeta
- segregatedAlphaBeta

In order to start building the training phase three activities must take place.

1. Some of the proteins listed in the dataset files do not have a corresponding PDB entry and result in an empty image which is manually removed from the folder.
2. The datasets do not have equal numbers of proteins per class and to prevent over-sampling, the lowest number of samples in one of the four classes is used. For the 25PDB dataset there are 1673 proteins, of which there are 443 'a' classified proteins, 443 'b' classified proteins, 347 'c' classified proteins, and 440 'd' classified proteins. This means that at most only 347 proteins can be used, and after step 1 of pruning bad images, only 312 proteins from each class could be used.

3. The last phase consists of sorting the iamge files into training, validating and testing buckets. I use 80% of the images to train, 10% to validate, and 10% to test. The end result will be a folder structure in the form of proteinImages

- Training
  - allAlpha
  - allBeta
  - mixedAlphaBeta
  - segregatedAlphaBeta
- Validating
  - allAlpha
  - allBeta
  - mixedAlphaBeta
  - segregatedAlphaBeta
- Testing
  - allAlpha
  - allBeta
  - mixedAlphaBeta
  - segregatedAlphaBeta

3.3 Training the CNN

Classification of proteins involves training a convolutional neural network (CNN) to predict the SCOP classification. The CNN is implemented in python using the PyTorch [15] framework for machine learning, which will build a prediction model from the known image examples for each SCOP classification to both train and test.

I used two designs of the CNN: one that has two layers of convolution and one that uses five, each with a pooling and rectifier phase. I didn't find that the accuracy improved with more CNN layers, but many papers [7] have stated the case that deeper CNNs perform better. After the convolution the outputs are passed into a 2-layer neural network which outputs the predicted class label.

Training starts with dataloaders, one for each dataset: training, validation and testing datasets. I took 80% of the original data to train with, and 10% for validation and 10% for testing. By holding back 20% of data, I can confirm that the CNN has not been overfitted to the training data. During training the images are fed into the forward function to derive the outputs, which are then fed into a loss function with the labels to be used to perform the backward pass that updates the parameters.

To prepare for the backward pass an optimization step is performed using an Adam[7] algorithm for stochastic optimization that reevaluates the model and returns the loss. The loss is calculated using a CrossEntropyLoss function which is computed by the function (3.1) from [17]

$$loss(x, class) = -log\left(\frac{exp(x[class])}{\sum_j exp(x[j])}\right) = -x[class] + log(\sum_j exp(x[j])) \tag{3.1}$$

where the inputs to the loss function are variables x and class,

- The variable x is a tensor and the result of the CNN predicting the class of a set of image files, in my case the size of the set or batch is 32. The variable x contains the image's probability for each classification, where the values range from 0 to 1
- The variable class is the known classification for each of the images, which is used to compare the predicted with the actual values.

The loss is also averaged over the number of images predicted in this batch run, in my case 32 instances.

The 'backward' function is then called to compute the gradients which minimized the loss function in weight space [19] and then the 'step' function is used to update the CNN with the next parameters. The basic implementation of training a neural network is as follows:

```
//set gradiants to zero
optimizer.zero_grad()
//forward pass compute the output using the CNN net
outputs = net(inputs)
//calculate the loss
loss_size = loss(outputs, labels)
//calculate the gradients
loss_size.backward()
```

```
            //update the CNN with the new gradients
            optimizer.step()
```

The above implementation will result in loading the images for training a CNN through a dataset image folder data loader:

```
train_data = torchvision.datasets.ImageFolder(root=TRAN_DATA_PATH,
                    transform=TRANSFORM_IMG)
train_data_loader = data.DataLoader(train_data, batch_size=32,
                    shuffle=True,  num_workers=4)
```

The image folder TRAN_DATA_PATH takes the path to images, which will result discovery of classes that will be used, as this method expects the root folder to contain subfolders for each class. During loading a transformation is applied to the images in the folder, which converts the image to a tensor, I also resize the image to a uniform 512 X 512 resolution and the images are randomly selected when shuffle is true.

The work to predict the classification occurs in the forward function which performs the following tasks:

1. The 2D convolution layers apply a convolution over the tensor, the images are converted into a 3 * 512 * 512 tensor, the depth 3 represents. The possible colours red, green and blue as 0-255 values.
    - The convolutional pass is configured with a kernel size of 3 and a stride of 1 and a padding of 1
        - The three layers of 512X 512 values are iterated over using a kernel with a 3 X 3 window that slides over the 2D tensor moving with a stride of 1 pixel.
2. Each convolution pass has the RELU and pooling function applied to the outputs
    - maxPool2d(F.relu(Conv2d (tensor(image)))
        - The F.relu applies a rectified linear unit function element-wise
        - maxPool2d applies a 2D max pooling over the input with a 2 * 2 kernel and a stride of 2
3. The result of the 2 Convolutional Neural Network layers is converted from a 64* 128 * 128 tensor into a 1 * 1048576 array which can then be passed to the first neural network layer that converts the inputs into 64 outputs that are finally passed into the last neural network that outputs to 4 values one for each class label.

3.4 Running the application

To generate the images the modified PV application was run on an IMac, after which on the same machine the images were processed. The convolutional neural network was then trained and tested in three different environments, an IMac, a Linux based laptop and on Google's Colaboratory environment which is an online execution environment.

Chapter 4

Results

4.1 Best Image

The first approach I took I call ‘best images’ which used the canvas to extract the pixels and determine which orientation of the protein has the highest contrast. Contrast was calculated by multiplying the beta-sheet coloured pixels by the alpha helix coloured pixels, or 1 if either are 0. This method of image generation resulted in 1100 training images, 275 image samples for each class.

I built a 2-layer convolutional neural network with a 2-layer neural network, which had the following accuracies:

- Training: 97% with 1100 images
- Validation: 76% with 160 images
- Testing: 76% with 160 images

4.2 Multiple Image

In the next experiment, using the ‘multiple orientation approach’, I trained the CNN based on the same 25PDB dataset but instead of a best image I generated 6 images per protein. These images are generated from 90 degree rotation algorithm using 3-dimensional rendering of a protein. This method does not have any opinion on the quality of the image and instead emphasized the importance of generating images that exemplify the SCOP class. Again, I used 80% of the data from 25PDB to train, 10% to test, and 10% to validate, with the same 2-layer CNN. This approach produced the results shown in Table 4.1.

Dataset	Number of Images	Accuracy of A%	Accuracy of B%	Accuracy of C%	Accuracy of D%	Overall Accuracy %
25PDB training	5972	100	100	100	100	100
25PDB validating	760	88	88	82	67	81
25PDB testing	748	81	81	78	69	78
FC699 testing	5001	48	64	91	58	74
D1185 testing	6861	78	58	78	22	51
D8244 testing	48493	82	78	37	73	66

Table 4.1: Results of model with 2 convolutional layers.

In the next experiment, I made a 5-layer convolutional neural network by adding 3 additional convolutional layers to the 2-layer CNN from above. This CNN has the same two fully-connected layers after the convolutional layers used in the first experiment, and it was trained and tested with the same data from the 25PDB dataset as the previous experiment.

The accuracies in Table 4.1 and 4.2 represent one run from the training and testing of a machine learning CNN. A run involves retraining a CNN with the 25PDB dataset and then using the resulting model to predict the different dataset images to produce an overall accuracy. Each run takes approximately eight hours to complete. These results varied by 2-3% from the shown values, and each 2-layer and 5-layer convolutional neural network was run about 10 times each. Most tests were made purely with the 25PDB dataset, as I only found the other three datasets later in the development process. The D8244 dataset was always tested on its own model from 25PDB training, and the FC699 and D1185 datasets where tested on the same model from 25PDB training, mainly for memory and timing as D8244 has a high number of images to classify.

Dataset	Total Images	Accuracy of A%	Accuracy of B%	Accuracy of C%	Accuracy of D%	Overall Accuracy %
25PDB training	5972	99	93	94	89	94
25PDB validating	760	95	84	85	84	87
25PDB testing	748	88	83	78	74	80
FC699 testing	5001	53	56	91	69	73
D1185 testing	6861	83	54	79	27	53
D8244 testing	48493	86	65	61	61	67

Table 4.2: Results of model with 5 convolutional layers.

Chapter 5

Conclusion

5.1 Contributions

Overall the work described in this thesis has demonstrated a novel approach using protein images to address the well-researched problem of predicting SCOP classification. The results are comparable with the best known methods (see Table 2.2). Results from my approach might be improved by overtraining on the C and D classes, that is by rebalancing the training data to favour those images, which are typically the harder to classify.

The application on the whole was designed to be interactive. Since the application is written in JavaScript, we could host a site where a user can enter a protein's name, see the image, view it being rotated and then get feedback on its predicted SCOP classification. This may be of interest to the general public as well as the research community.

In general this work allowed me to explore a number of research areas required to solve a real world problem: (1) generating data, (2) understanding and building a JavaScript programming model, (2) updating an existing program, (3) learning PyTorch and (4) understanding how to build a machine learning model with CNN, using all its different properties, which can be more fully explored.

5.1.1 Availability of my code

The code for this project has been checked into the following git repository [3]

- The file ssCNN5Layer512.py is of main interest and contains
  - The definition of two python CNN classes CNN5Layer and CNN where the layers of the convolution neural network is defined and a forward propagation

function that connects the layers together.

    * Note, I switch between the two depending on the run I am executing.

  - The computeAccuracy function is where I calculate the accuracy and output total accuracy, accuracy per class and the confusion matrix per class, how many were right and what incorrect class was selected.
  - The function trainNet is the main loop where the CNN learns to predict the correct classes.

The git repository [4] hosts the modified/forked PV application.

My three contributions that modified the PV application to capture protein images were, 1) to add a class to parse protein dataset files, 2) to render the protein described by the dataset file, and 3) to produce orthogonal images of the protein.

The parser was implemented Python and can be found in the file proteinPicker.py, which is in the ProteinClassification folder. The proteinPicker.py file also stores the relationship between which protein belongs to which class, which is used to save an image in its appropriate classification folder.

The p3Serve2, also implemented in Python is the main method for the PV application which serves the static HTML and JavaScript files to the client. The p3Serve2 program was further modified to provide a POST method for the client to call to save an image give the image data and protein name. Also the p3Serve2 program has a GET method that the client uses to fetch the next protein that has not been image captured so far.

The two next contributions to the PV application were implemented in JavaScript and the additions where made in the 'js' folder, adding files ImageGenerator.js, FiniteIterateSearch.js and PVModel.js.

The new JavaScript files allow the client to load a given proteins PDB file from the Protein Databank website [18], processing the file to match the scope of the protein eliminating chains and isolating to indexed section of a chain specified by the dataset. The FiniteIterateSearch.js file will then rotate the visualized program and send back to the server the orthogonal images to be persisted on the server.

5.1.2 General Remarks

The results of the CNN approach performed well against the 25PDB dataset with a best accuracy of 87%, but the model performed poorly on the two benchmark datasets D1185 and D8244 with accuracies ranging from 51% to 67%. Including images from these datasets into the training phase would likely produce an overall improvement to their specific scores. I did not include them, because the proteins in D1185 and D8244 have a higher similarity of 40% as oppose to the lower similarity of 25% in 25PDB. Including more similar proteins may result in overtraining on those similar proteins with a resulting loss of accuracy when trying to predict the class of a new protein that is not in any of the four datasets. Nevertheless, this should be tried to verify my assumption.

5.1.3 Future Work

An improvement on the current system would be in developing a classification model that is capable of continuously learning as new proteins are discovered. Continuous learning would mean that an interface could be provided to expert users to spot check the model with new proteins. This would allow a user to add interesting proteins to the overall training, first seeing if the model was correct and if not, there could be a means of indicating the true class. I trained both a 2-layer and 5-layer CNN, but trying a range of different CNN compositions could lead to better overall model accuracy.

References

[1] J.M. Berg, J.L. Tymoczko, and L. Stryer. *Biochemistry, Fifth Edition*. W.H. Freeman, 2002. ISBN: 9780716730514. URL: `https://books.google.ca/books?id=uDFqAAAAMAAJ`.

[2] *CS231n Convolutional Neural Networks for Visual Recognition*. `http://cs231n.github.io/convolutional-networks/`. (Accessed: 2019-05-01).

[3] D. Franklin. `https://github.com/dfranklinsmail/thesis-imageClassification`. (Accessed: 2019-05-01).

[4] D. Franklin. `https://github.com/dfranklinsmail/pv`. (Accessed: 2019-05-01).

[5] T.J.P. Hubbard, B. Ailey, S.E. Brenner, A.G. Murzin, and C. Chothia. "SCOP, Structural Classification of Proteins Database: Applications to Evaluation of the Effectiveness of Sequence Alignment Methods and Statistics of Protein Structural Data". In: *Acta Crystallographica Section D* 54.6 Part 1 (Nov. 1998), pp. 1147–1154. DOI: `10.1107/S0907444998009172`. URL: `https://doi.org/10.1107/S0907444998009172`.

[6] D.T. Jones. "Protein secondary structure prediction based on position-specific scoring matrices." eng. In: *J Mol Biol* 292.2 (Sept. 1999), pp. 195–202. ISSN: 0022-2836 (Print); 0022-2836 (Linking). DOI: `10.1006/jmbi.1999.3091`.

[7] D.P. Kingma and J. Ba. "Adam: A Method for Stochastic Optimization". In: *CoRR* abs/1412.6980 (2015).

[8] L. Kong and L. Zhang. "Novel structure-driven features for accurate prediction of protein structural class". In: *Genomics* 103.4 (2014), pp. 292–297. ISSN: 0888-7543. DOI: `https://doi.org/10.1016/j.ygeno.2014.04.002`. URL: `http://www.sciencedirect.com/science/article/pii/S0888754314000627`.

[9] L. Kurgan, K. Cios, and K. Chen. "SCPRED: Accurate prediction of protein structural class for sequences of twilight-zone similarity with predicting sequences". In: *BMC Bioinformatics* 9.1 (May 2008), p. 226. ISSN: 1471-2105. DOI: 10.1186/1471-2105-9-226. URL: https://doi.org/10.1186/1471-2105-9-226.

[10] L. Kurgan and L. Homaeian. "Prediction of structural classes for protein sequences and domains—Impact of prediction algorithms, sequence representation and homology, and test procedures on accuracy". In: *Pattern Recognition* 39 (Dec. 2006), pp. 2323–2343. DOI: 10.1016/j.patcog.2006.02.014.

[11] S. Liu and W. Deng. "Very deep convolutional neural network based image classification using small training sample size". In: *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR).* (Nov 2015), pp. 730–734. DOI: 10.1109/ACPR.2015.7486599.

[12] G.J. McLachlan, John Wiley & Sons., and Wiley InterScience (Online Service). *Discriminant analysis and statistical pattern recognition / G.J. McLachlan.* English. Wiley-Interscience Hoboken, N.J, 2004, xv, 526 p. : ISBN: 0471615315 0471691151 0471725293. URL: http://ezproxy.lib.monash.edu.au/login?url=http://dx.doi.org/10.1002/0471725293.

[13] M.J. Mizianty and L. Kurgan. "Modular prediction of protein structural classes from sequences of twilight-zone identity with predicting sequences". In: *BMC Bioinformatics* 10.1 (Dec. 2009), p. 414. ISSN: 1471-2105. DOI: 10.1186/1471-2105-10-414. URL: https://doi.org/10.1186/1471-2105-10-414.

[14] A.G. Murzin, S.E. Brenner, T. Hubbard, and C. Chothia. "SCOP: a structural classification of proteins database for the investigation of sequences and structures." eng. In: *J Mol Biol* 247.4 (Apr. 1995), pp. 536–540. ISSN: 0022-2836 (Print); 0022-2836 (Linking). DOI: 10.1006/jmbi.1995.0159.

[15] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. "Automatic differentiation in PyTorch". In: *NIPS 2017 Workshop Autodiff Submission* (2017).

[16] *PV - JavaScript Protein Viewer.* https://biasmv.github.io/pv. (Accessed: 2019-05-01).

[17] *pytorch reference api.* https://pytorch.org/docs/stable/nn.html. (Accessed: 2019-05-01).

[18] *RCSB Protein Data Bank.* https://www.rcsb.org/. (Accessed: 2019-05-01).

[19] R. Rojas. *Neural Networks: A Systematic Introduction.* Berlin, Heidelberg: Springer-Verlag, 1996. ISBN: 3-540-60505-3.

[20] *SCPRED method for in-silico prediction of protein structural classes.* `http://biomine.cs.vcu.edu/datasets/SCPRED/SCPRED.html`. (Accessed: 2019-05-01).

[21] G. Slabaugh. *Computing Euler angles from a rotation matrix.* `www.gregslabaugh.net/publications/euler.pdf`. Jan. 1999 (Accessed: 2019-05-01).

[22] A. Watt and M. Watt. *Advanced Animation and Rendering Techniques.* New York, NY, USA: ACM, 1991. ISBN: 0-201-54412-1.

[23] X. Xia, M. Ge, Z. Wang, and X. Pan. "Accurate Prediction of Protein Structural Class". In: *PLOS ONE* 7.6 (June 2012), pp. 1–8. DOI: `10.1371/journal.pone.0037653`. URL: `https://doi.org/10.1371/journal.pone.0037653`.

[24] J. Yang, Z. Peng, and X. Chen. "Prediction of protein structural classes for low-homology sequences based on predicted secondary structure". In: *BMC bioinformatics* 11 Suppl 1 (Jan. 2010), S9. DOI: `10.1186/1471-2105-11-S1-S9`.