ABERYSTWYTH UNIVERSITY

Deep neural networks for pattern recognition

CS39760 Major Project

Author: Jiaying Li (jil20@aber.ac.uk)

Supervisor: Chuan Lu (cul@aber.ac.uk)

2017/4/21
Version 1.0 (Release)

This report was submitted as partial fulfilment of a BSc degree in Degree Scheme: G400 Computer Science

Department of Computer Science
Aberystwyth University
Aberystwyth, Ceredigion, SY23 3DB
Wales, UK

Declaration of originality

I confirm that:

- This submission is my own work, except where clearly indicated.
- I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
- I have read the regulations on Unacceptable Academic Practice from the University's Academic Quality and Records Office (AQRO) and the relevant sections of the current Student Handbook of the Department of Computer Science.
- In submitting this work I understand and agree to abide by the University's regulations governing these issues.

Name.........Jiaying Li..

Date............07/05/2017..

Consent to share this work

By including my name below, I hereby agree to this dissertation being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Name.............. Jiaying Li...

Date.................. 07/05/2017..

Acknowledgements

First I want to thank my parents who support me to go abroad and study in the UK, not only financially but also mentally. You always keep contact with me and guide me. I am so grateful to both of you.

Aberystwyth is a small, quiet town which is the opposite of the city I lived before. It helps me to focus on what I should do and learn things like English and living skills. The people in this town are really kind and easy to get alone with. I would never forget the days I lived in here.

I would like to thank my supervisor Chuan Lu. You are so patient to guide me and point out the insufficiency. You advise me of ideas which help me to get rid of obstacles. I really appreciate it.

Thanks to the PHD guys in the lab. You invite me to the activities and play sports together. I enjoy the time with you guys.

Wish you all the best!

Jiaying Li
07/05/2017

Abstract

Convolution neural network has become a considerable choice for extracting features from multidimensional data, e.g., image and video [1]. A CNN is composed of one or more convolutional layers with fully connected layers on top. It also uses tied weights and pooling layers [2]. This architecture endows CNNs with the ability to extract dense features from sparse input data. Therefore, in this project, we are trying to apply CNN on a single dimension dataset to learn the patterns hidden in the amino acid sequences and then predict its corresponding secondary structure in a protein. We reached around 80% accuracy using 4 layers of CNN and 2 layers of fully connected network on a few independent bench mark test sets. Compared to the state-of-the-art method, we still have rooms for improvement. The use of Theano [3] based Keras library helps me to start the network building quickly and focus on the architecture design instead of low-level coding [4].

CONTENTS

List of figures

List of Tables

Chapter 1

INTRODUCTION

In this chapter, we will discuss the motivation of our project, what we are aiming for and the deep neural networks we used in our project

1.1. Background

With the advance and development of the protein research technologies, there are more and more protein sequences that are determined in the last decade. UniProt (Universal protein) is one of the most famous and abundant databases which records and classifies proteins. It contains approximately 84 million proteins so far and the number is still increasing dramatically every day, with only 550 thousand of them have been functional annotated [5]. Why there is only a small portion of them been reviewed?

Research shows that the function of a protein usually depends on its three-dimensional structure. But the methods to determine three-dimensional structures like X-ray crystallography, NMR spectroscopy are very difficult, expensive and can be only applied to small molecule proteins. Thus, the process of functional determination remains sluggish.

In order to accelerate the process, we have to find a way to analyse the protein sequence information and predict the 3D structure more efficiently rather than manually experiment. Some scientists suggest that Computational Biology which is also called Bioinformatics is a rational solution.

1.2. Protein Structure Prediction

Proteins are series of amino acids concatenated together by peptide bonds. It is the conformational changes such as rotation or folding shape the three-dimensional structure of a protein. There are overall twenty amino acids and four structures within the protein (Primary, Secondary, Tertiary, Quaternary structure) whereas the secondary structure could be considered as the most important factor to determine the protein structure. If we can correctly predict the secondary protein structure, we can then derive the further structure like its tertiary structure or even the overall structure.

The prediction of protein secondary structure is usually evaluated by Q3 or Q8 accuracy, which means that its accuracy is judged by the percentage of residues that are been correctly predicted into 3-state and 8-state secondary structure. Early methods are applied to a single sequence and predict based on the previous known structure with statistic probability analysis, resulting in around 60-65% accuracy. The introduction of modern machine learning methods such as neural networks has raised the accuracy up to 80%.

Protein structure prediction has become one of the most important goals pursued by bioinformatics and theoretical chemistry because it benefits to industries e.g. drug design and the design of novel enzymes. Every two years, the performance of current methods is assessed in the CASP experiment (Critical Assessment of Techniques for Protein Structure Prediction) [6]. However, the current best Q3 accuracy is still around 80% obtained by PSIPRED and other state-of –the-art methods such as JPRED. We need models that can recognize structural information and extract features from sparsely distributed data to further improve the performance.

1.3. Deep Learning

Deep learning has become one of the most popular branches in Machine Learning field. With the ability to learn abstract representations of data, it shows a great potential on many complex domains such as image recognition, natural language processing and artificial intelligence. The hidden layers between input and output can derive higher level features from lower level feature maps to form a hierarchical representation.

Convolutional Deep Neural networks are first introduced from modelling animal visual perception, to solve visual recognition problems. Convolutional neural networks consist of multiple layers of receptive fields. These fields are working as brain cells to receive a portion of your visual world using convolutional kernels. Each portion will finally tile so that their input regions overlap, eventually result in a higher-resolution representation of the original image. These features endow CNNs with the ability to extract local context from the long amino acid sequences.

1.4. Project Analysis

This project is aiming at predicting the structure of the proteins, particularly the secondary structure using convolution neural networks and comparing the results with the current algorithms that are also working on predicting secondary structures; see if our implementation has any advance. Therefore, our objectives can be divided into the following points:

1) To understand the background and general approaches that is used to predict protein secondary structure.

2) To study the previous state-of-the-art methods and see how they process data or what is their parameters configuration.

3) To study the architecture of convolution neural networks on the other applications and develop our own approach for protein secondary structure prediction.

4) To implement my convolution neural networks and try to improve the accuracy by parameter tuning using either grid search or random search.

5) To compare the results with the other state-of-the-art methods and discuss the pros and cons.

The challenge of this project might be the pre-process of the protein data. We need to ensure that proteins in the dataset share less than a rational number of percentage similarity to improve the efficiency of our network. The data type is another problem to be considered. The different input format chosen will result in different network architecture and data representation.

1.5. Research Method

Projects on Machine Learning field do not require a lot of coding. It is more focusing on algorithm design rather than application developing. And you do not have a specific requirement as well; the only goal is to improve the accuracy. It is hard to apply classic development methods such as Waterfall or XP, Scrum. Thus, I would prefer to use some of the agile features and principles during my project process:

a) Develop small, incremental releases and iterate

b) Quick responses to change and continuous development

c) Weekly meet with supervisor to review status

1.6. Development Method

The overall project has been pushed to GitHub for improving the traceability of the code. GitHub is a powerful tool to track the version of our code. We can simply manage the project by committing changes or reverse it.

The entire project is publicly available at: https://github.com/JiayingLi/Protein-Secondary
-Structure-Prediction-using-Convolution-Neural-Network

Chapter 2

Experiment Method

This chapter will introduce the theoretical principles of PSSP (Protein Secondary structure Prediction) and the general concept of convolution neural networks.

2.1. Principles of Protein Secondary Structure Prediction

2.1.1. Characteristics of protein structure

The protein structures we discussed here refer to the spatial 3D structures of protein molecules. All proteins are built from 20 different L-α-amino acids showed in Figure 2.1. They are called residues after each amino acid is linked in a protein chain.

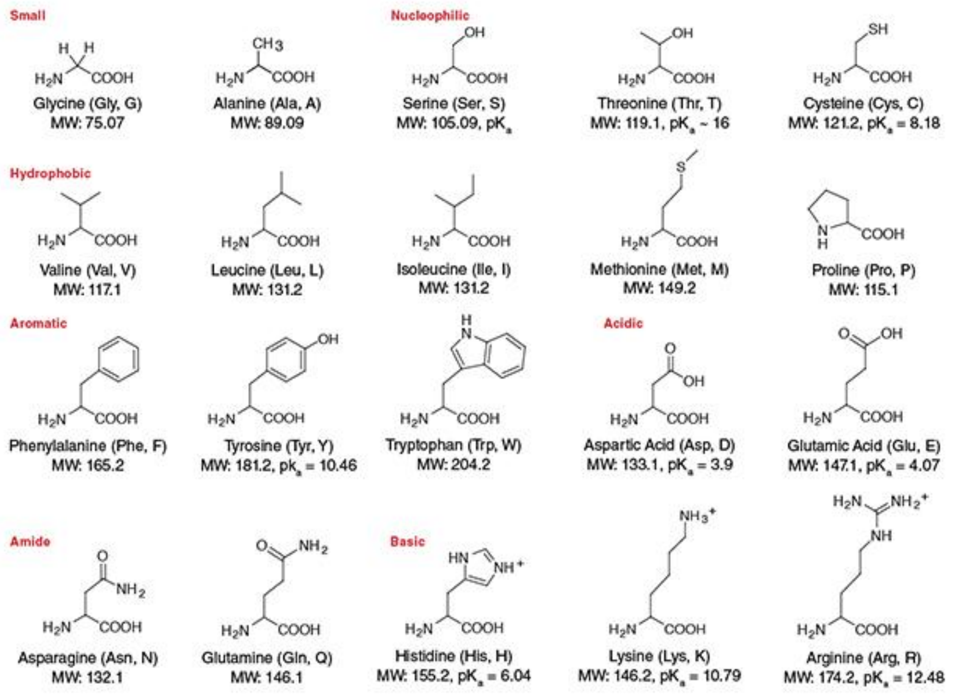

Small
Glycine (Gly, G) MW: 75.07
Alanine (Ala, A) MW: 89.09
Nucleophilic
Serine (Ser, S) MW: 105.09, pK$_a$
Threonine (Thr, T) MW: 119.1, pK$_a$ ~ 16
Cysteine (Cys, C) MW: 121.2, pK$_a$ = 8.18
Hydrophobic
Valine (Val, V) MW: 117.1
Leucine (Leu, L) MW: 131.2
Isoleucine (Ile, I) MW: 131.2
Methionine (Met, M) MW: 149.2
Proline (Pro, P) MW: 115.1
Aromatic
Phenylalanine (Phe, F) MW: 165.2
Tyrosine (Tyr, Y) MW: 181.2, pk$_a$ = 10.46
Tryptophan (Trp, W) MW: 204.2
Acidic
Aspartic Acid (Asp, D) MW: 133.1, pK$_a$ = 3.9
Glutamic Acid (Glu, E) MW: 147.1, pK$_a$ = 4.07
Amide
Asparagine (Asn, N) MW: 132.1
Glutamine (Gln, Q) MW: 146.1
Basic
Histidine (His, H) MW: 155.2, pK$_a$ = 6.04
Lysine (Lys, K) MW: 146.2, pK$_a$ = 10.79
Arginine (Arg, R) MW: 174.2, pK$_a$ = 12.48

Figure 1: Twenty common Amino Acids in protein

Source: https://www.neb.com/tools-and-resources/usage-guidelines/amino-acid-structures

Protein synthesis is the process of amino acids connection. Two or more amino acids form a peptide chain through dehydration condensation, namely the original protein fragments, and then form specific protein structures through folding and spiralling.

There are four structures that can be composed of amino acids: Primary, Secondary, Tertiary and Quaternary structure. Primary structures are simply chains of amino acids. Secondary structures start to have some local spatial features that can represent a region in the same protein molecule. Tertiary structures are basically the overall shape of a single protein molecule i.e. the relationship between the secondary structures inside a protein. Quaternary structures are the structures that formed by a few protein molecules. It is hard to directly predict a tertiary or quaternary structure by the known amino acid sequences, but it is relatively simpler to derived secondary structure from the sequence motif (the pattern of the amino acid sequence), then to predict the function of a protein.

2.1.2. Protein Homology

Protein homology is defined in terms of shared ancestry. Homologous proteins are likely to share similar structures and functions. Sequence alignment is the method that used to indicate which regions of sequence are homologous [7]. The rows of the alignment matrix are the protein sequences while the columns represent the matching status.

Profile alignment is one of the sequence alignment methods. They were first used by Gribskov [9]. It produces the scores based on the amino acid probability distribution in each column. The most famous one should be PSI-BLAST. It obtains a position-specific scoring matrix (PSSM) or profile features from the multiple sequence alignment of sequences detected above a given score threshold using protein–protein BLAST [8]. Therefore, PSI-BLAST provides a means of detecting distant relationships between proteins. PSIPRED which was prediction using PSI-BLAST profile obtained around 80% accuracy on Q3 classification. Hence, we are going to use profile alignment in our project to process the protein sequences data.

2.1.3 Protein Structure Prediction

Scientist started to research on predicting protein structures with amino acid sequences since Anfinsen [26] showed that the information necessary for protein folding residues completely within the primary structure.

The appearance of rapid methods of DNA sequencing and the translation of the genetic code into protein sequences has boosted the need for automated methods of interpreting these linear sequences into terms of two or three-dimensional structure [27].

With the development of computing resources, especially the use of GPU, there is no doubt that manually experiment on protein prediction such as X-ray crystallography or NMR techniques would be gradually abandoned. Instead, neural network or deep neural network will be the primary choice to predict protein structure.

Typically, deep neural networks have the ability to capture distantly relationship which makes it capable of recognising the homology among proteins and predict structures accordingly.

2.2. Convolution Neural Network

Convolution neural networks are variations of Multilayer Perceptrons model inspired by biology process, introduced by Hubel and Wiesel in the 1960s through the research to cat and monkey visual cortex neurons. Provided the eyes are not moving, the region of visual space within which visual stimuli affect the firing of a single neuron is known as its receptive fields [10].

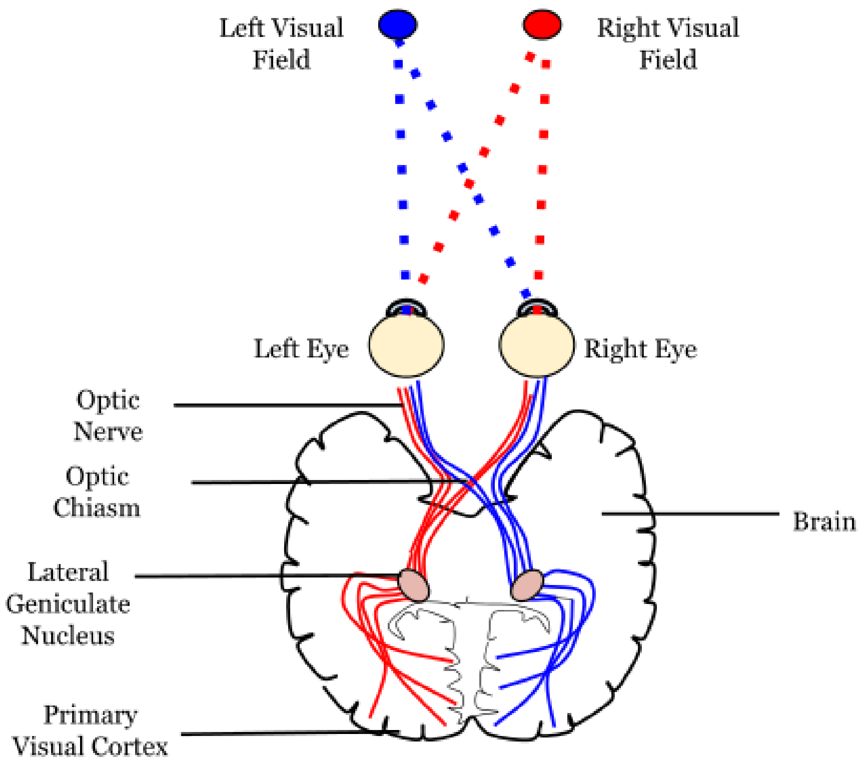


Figure 2: Neural pathway diagram

Source: https://commons.wikimedia.org/wiki/File:Neural_pathway_diagram.svg

The cells that in the same cortex have similar and overlapping receptive field, which cooperate together bring out a complete image from the target space. There are two types of cells in the visual field: Small cells are mainly for responding to the straight edges within their field; while the complex cells have larger receptive field but consequently not that sensitive in detecting edges compared to small cells.

The animals’ visual cortex is the most powerful image processing system that, many models in the visual recognition field are inspired and constructed from the visual neurons such as Neocognitron, LeNet-5, Shift-invariant neural network and etc.

Sparse Connectivity and Shared Weights are two essential features of convolution neural network. It is these features significantly reduce the parameters required in the traditional neural networks and make the training of multiple hidden layers become feasible.

2.2.1. Sparse Connectivity

Convolution neural networks explore the spatially local features by strengthening the Local Connectivity Pattern between the neurons of adjacent layers. As shown in figure 3, the input of layer 2 is the local subset of all the nodes from the last layer. Assuming that each of the neurons in layer 2 can receive a receptive field of size three, then they will only connect to three adjacent nodes from layer 3. The connections for the next layer are the same. Therefore, we can see from the figure that, though the output layer only responds to three neurons in the last layer, the overall receptive field has become massive when it comes to the input layer.

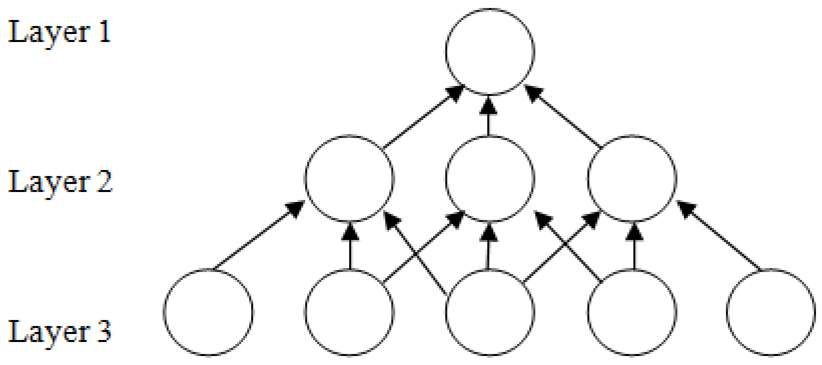


Figure 3: CNNs sparse connectivity

This architecture allows the neural networks to extract the local feature from a large scale of input data or identify the homologies from distant regions. Additionally, the local connections between small regions of data can significantly decrease the number of parameters in the layers. For a large deep neural network, the reduction of parameters is essential.

2.2.2. Shared Weights

Another trait of CNN is shared weights. During training, CNNs have units called filters sliding across the width and height of the input volume. Those units share the same weights and bias and form a feature map. The feature detection is regardless of the feature position when the neurons are replicated in this way.

Moreover, the number of being learnt parameters is decreased which relatively increase the learning efficiency and urge the CNN to be a more powerful model on image processing field.

2.2.3. CNN Architecture

A convolution neural network usually consists of three main distinct types of layers: Convolution layer, down-sampling layer and fully-connected layer. Figure 4 is a typical convolution neural network.

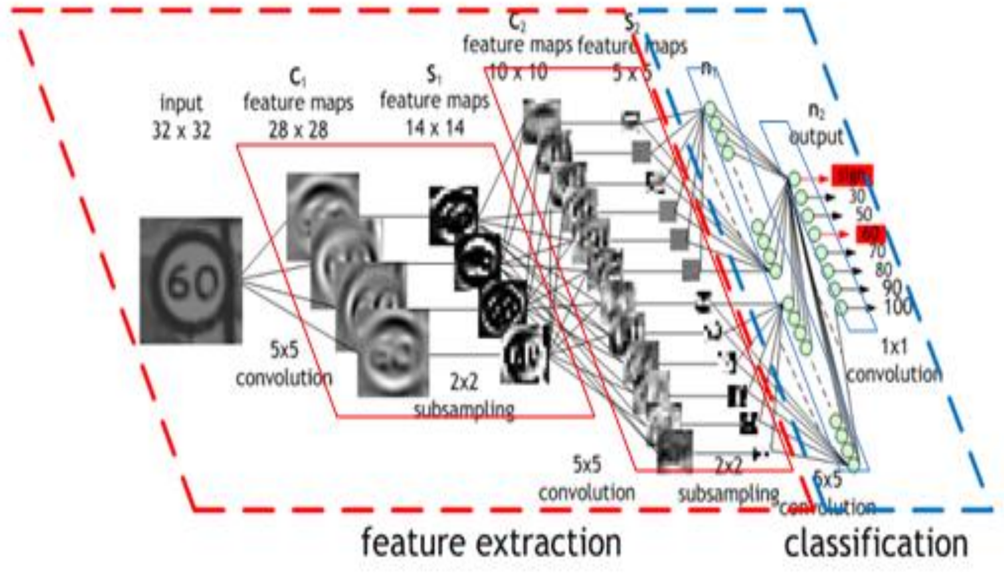


Figure 4: A typical convolution neural network

Source: https://devblogs.nvidia.com/parallelforall/deeplearning-nutshell-core-concepts/

Convolution Layer

Convolution layer is the core building block of CNN that plays the most important role in feature extraction. It is composed of a set of learnable kernels (filters). Every kernel is small spatially, but extends through the entire visual field. During the forward process, we slide the kernels across the width and height of the input volume and compute dot products between the entries of the filter and the input at any position. This will result in a 2-dimension activation map that gives the reflection on every position of the input field. After a few layers of convolving, the network will eventually learn to react to some types of features such as edges of some orientation or corners that in different colours.

Figure 5 is an example of convolution. Each neuron in the convolutional layer is connected only to a local region in the input volume spatially, but to the full depth. Three hyperparameters control the output volume of the convolutional layer: depth, stride and zero-padding. The depth in the figure is 5. The design of depth usually depends on the number of classes you have in the input, e.g. the convolution network for colour classification will have a depth of 3 because colour has 3 channels. The stride will specify the size of each step when the kernel sliding through the input. The zero-padding which pad zeroes around the border allow us to control the size of output volume.

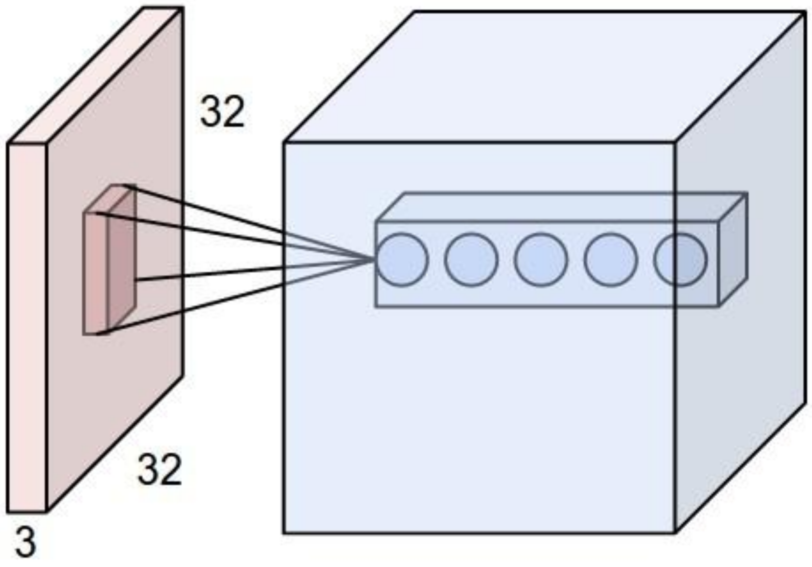


Figure 5: A convolutional process

Source: http://cs231n.github.io/convolutional-networks

Down-Sampling Layer

Down-sampling layer is also known as pooling layer. There are several non-linear pooling layers while the max-pooling layer is the most commonly used one. It reduces the spatial size of the representation by doing max operations in a set of non-overlapping sub-regions (the size is generally 2x2). The depth dimension remains unchanged. The use of down-sampling layer not only reduces the number of parameters and amount of computational work, but also controls the occurrence of overfitting.

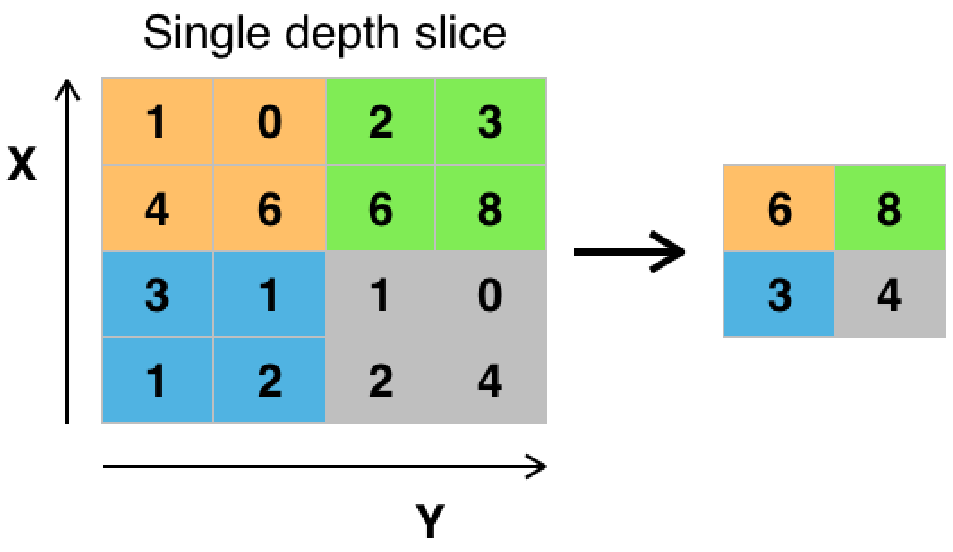


Figure 6: Max pooling with a 2x2 filter and stride = 2

Source: https://en.wikipedia.org/wiki/Convolutional_neural_network#Pooling_layer

Fully-Connected Layer

Fully-connected layer is usually used as the final layer and will output the result that been processed by the network. It is composed of neurons that have full connections to all activations in the previous layer.

Activation Function

ReLU is the most common used activation function for convolution neural networks. Its full name is Rectified Linear Unit. Here is the formula of ReLU:

$$f(\mathrm{x}) = \max\,(0, \mathrm{x})$$

There are also other activation functions, but the reason ReLU is preferred is that it increase the non-linearity in a more efficient way [11]. Using ReLU can reduce training time without making a noticeable difference to the generalisation accuracy.

Softmax is the commonly used activation function on classification problems as the output layer. The equation is as following:

$$\sigma(\mathrm{Z})_{\mathrm{j}} = \frac{e^{z_j}}{\sum_{k=1}^{k} e^{z_k}}$$

The sum of the Softmax output will add up to 1. Each output element could be considered as the predicted possibility for that class.

Dropouts

Dropout is one of the regularization techniques, which aims to prevent complex co-adaptations on training data with the goal to avoid overfitting. It will randomly drop units from the neural network during training. Only the nodes left are trained on the data at the epoch. The removed node will then reinsert into the network with their original weights. This prevents units from co-adapting too much [20].

0.5, 0.3, 0 is the numbers that frequently used in neural network dropouts. You can design you own dropout values according to your own architecture and dataset as well.

2.2.4 Feature Embedding

For a better interpreted data, we can use two types of method to encode the amino acid dataset: One-Hot vector encoding and Multiple Sequence Alignment.

One-Hot vector encoding

One-hot is a series of bits that with only a single high bit (1) while all the others are low (0). In our case, there are 20 amino acids and 3 structures to be represented. Therefore, the one-hot vector will look like the following:

	A	R	N	D	C	E	Q	G	H	I	L	K	M	F	P	S	T	W	Y	V
Ala:	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

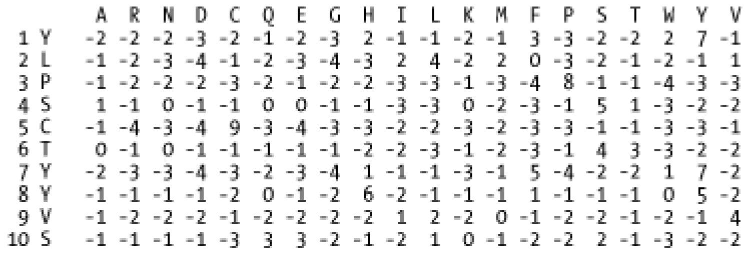
	H	E	C
Helix:	1	0	0

According to this, the size of a protein sequence will be n $\times$ 20 after encoding, where n is the number of residues in the protein sequence.

Multiple Sequence Alignment

Multiple sequence alignment (MSA) is a sequence alignment of three or more biological sequences, commonly used on protein, DNA, RNA. It was first introduced to exploit the similarity or the evolutional relationship between sequences. After the use of MSA in PHD neural network, researchers found that it could bring considerable improvement to the prediction [12].

Position-Specific Iterative Basic Local Alignment Search Tool (PSI-BLAST) provides a function to construct profiles from given protein sequences. By producing a scoring matrix based on the likelihood of the ith amino acid replacing the current residue, PSI-BLAST is able to isolate the motifs (patterns) among multiple sequences.



		A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
1	Y	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	-1
2	L	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4	-2	2	0	-3	-2	-1	-2	-1	1
3	P	-1	-2	-2	-2	-3	-2	-1	-2	-2	-3	-3	-1	-3	-4	8	-1	-1	-4	-3	-3
4	S	1	-1	0	-1	-1	0	0	-1	-1	-3	-3	0	-2	-3	-1	5	1	-3	-2	-2
5	C	-1	-4	-3	-4	9	-3	-4	-3	-3	-2	-2	-3	-2	-3	-3	-1	-1	-3	-3	-1
6	T	0	-1	0	-1	-1	-1	-1	-1	-2	-2	-3	-1	-2	-3	-1	4	3	-3	-2	-2
7	Y	-2	-3	-3	-4	-3	-2	-3	-4	1	-1	-1	-3	-1	5	-4	-2	-2	1	7	-2
8	Y	-1	-1	-1	-1	-2	0	-1	-2	6	-2	-1	-1	-1	1	-1	-1	-1	0	5	-2
9	V	-1	-2	-2	-2	-1	-2	-2	-2	-2	1	2	-2	0	-1	-2	-2	-1	-2	-1	4
10	S	-1	-1	-1	-1	-3	3	3	-2	-1	-2	1	0	-1	-2	-2	2	-1	-3	-2	-2

Figure 7: PSSM for the first 10 amino acids of the coelacanth HoxA11 protein

Source:

http://etutorials.org/Misc/blast/Part+V+BLAST+Reference/Chapter+13.+NCBI-BLAST+Reference/13.8+blastpgp +Parameters+PSI-BLAST+and+PHI-BLAST/

2.2.5. Reduction of predicting state

The most successful approaches for secondary structure prediction apply machine learning algorithms to maximize the sequence relationship between proteins' primary sequences and their assigned secondary structure as defined by the program DSSP [28] . They defined eight secondary structure classes which are H (α-helix), G(310-helix), I(π-helix), E(β-strand), B(isolated β-bridge), T(turn), S(bend), and - (coil). Furthermore, we can classify them into three classes due to the similarity among them. We can merely predict the function of a protein just by those three classes. They are:

DSSP class:	Prediction Classes:
H, G	(H) α-helix
E	(E) β-Strand
B, I, S, T, E, G, H	(C) Coil

2.2.6. Parameter Tuning

Parameter tuning is also called hyperparameter optimizer which is literally to justify a set of parameters that optimizes the performance in terms of the current network architecture. There are two common tuning method used in Convolution Neural network.

The classic way to optimize the hyperparameters is Grid Search, which is basically an exhaustive searching within all the possible choice of parameter setting. But since the Neural Networks are growing massively as well as the parameters, sometimes it is impossible to search exhaustively through all the possibilities in polynomial time. Therefore, Random Search is introduced. Random

Search is a empirical approach which will test over a set of fixed numbers that has been proved to be efficient. This is more effective than grid search because in most case, parameters of best performance are similar [32].

2.3. Evaluation Method of Prediction Accuracy

Though we have training accuracy and training loss throughout the process, we would still like to discover more methods and measures to further analyse our neural network performance.

2.3.1. Confusion Matrix

The size of a confusion matrix is usually decided by the number of classes we needed for prediction. Therefore, in this task, we would produce a matrix of size 3x3 where each rows and columns represented by the three states H, E, C respectively.

	H_{pred}	E_{pred}	C_{pred}
H	100	30	40
E	40	60	10
C	100	50	20

Table 1: An example of confusion matrix

The sum over the columns of matrix gives the number of residues n_j which are predicted to be in structure type j. The sum over the rows is the number of test labels. Generally speaking, each A_{ij} in a grid is the number of structure type i predicted to be type j, and the diagonal elements of the matrix are the correct predictions while the off-diagonal elements can be seemed as the wrong predictions [18].

2.3.2. Percentage of Correctly Predicted Residues

Q3 accuracy is the main measurement of the accuracy predicted in protein secondary structure. It is computed by the following equation which estimates the percentage of all correctly predicted residues within the three-state (H, E, C) classes:

$$Q_3 = \frac{\sum_{i=1}^{3} A_{ii}}{N} x100$$

Where N is the total number of residues. It can be also represented by this:

$$N = \sum_{i=1}^{3} N_i$$

Furthermore, the correctly predicted accuracy for each residue can be written as

$$Q_i = \frac{A_{ii}}{N_i} x100, (i = H, E, C):$$

2.3.3. Matthews Correlation Coefficient (MCC)

The Matthews correlation coefficient is originally used in machine learning as a measure of the relationship between two variables, introduced by biochemist Brian W. Matthews in 1975 [30]. It aims to solve the problem that there is only algorithm for counting the percentage of true positive but no penalty when data is predicted to be wrong classes. The MCC can be obtained directly from confusion matrix by this formula:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

There is also an improved version for multiclass classification problem:

$$C_i = \frac{P_i \times N_i - U_i \times O_i}{\sqrt{(P_i + U_i)(P_i + O_i)(N_i + U_i)(O_i + N_i)}}$$

Where P_i is the number of correctly predicted class; N_i the number of those correctly rejected; U_i the number of false negatives, and O_i that of over-predictions (False positive).

Chapter 3

Experiment details and Hyperparameters configuration

In this chapter, we will analyse the experiment detail and discuss the hyperparameters configurations.

3.1 Datasets

Rost and Sander defined non-redundancy to mean that no two proteins in the set share more than 25% sequence identity over a length of more than 80 residues [13]. They sorted out 126 proteins set with which to train and test secondary structure prediction algorithms. Many well-known algorithms and programs like PHD, PSI-PRED and JPRED have been trained on the Rost and Sander set of 126 proteins.

In structural classification of proteins (SCOP), protein structure superfamilies are defined from careful analysis of structure, evolution and function. The SCOP superfamilies contain protein domains that have the same fold and are likely to have evolved from a common ancestor [31]. The ASTRAL90 dataset was derived from SCOP database where it filtered sequence with less than 30% identity to each other. It contains 3,344 proteins with secondary structure labels.

Researches show that larger dataset can improve the performance of protein secondary structure prediction [15], and it is also essential for CNN to be trained on large dataset in order to generate features. Therefore, we obtained another ASTRAL dataset that filtered at 95% sequence identity which consists of 15,201 protein sequences originally from Protein Data Bank for training. It was used for our primary training set.

Critical Assessment of protein Structure Prediction (CASP) is a worldwide conference of protein structure prediction held every two years since 1994 [6]. During the process, more than 100 groups from all over the world will evaluate their prediction methods on the given dataset. Our CASP dataset was derived from target proteins of the 9th CASP competition.

CB513 [25] is a benchmark dataset that used for test only. It was originally created with the non-redundant protein sequence sets from 3Dee database. This representative set of 554 domain sequences was called CB554. Some scientists want to combine CB554 with RS126. To ensure that they do not share similar sequence, 158 homologies were removed and this left 396 sequences as

CB396. Also, there were 9 proteins showing similarity to at least one other within the RS126 set that needs to be removed. Finally, the combination of the two sets results in the CB513 datasets. It is such widely used that it almost turns up at every single paper about Protein Structure Prediction.

3.2 PSI-BLAST

PSI-BLAST is the tool to generate Position Specific Matrix which is basically kind of feature profile that is used in prediction process. It produced sequences profile with 20 channels which are the same as the number of protein category. PSSM rescaled through a logistic function and it was calculated by BLAST+ (A local programme for PSI-BLAST which can input multiple sequences at once provided by NCBI) against the UniRef90 database with E-value threshold 0.001 and 3 iterations.

3.3 Implementation Platform

My code is implemented on Keras library [4], a publicly available deep learning library which is built on the basis of Python and Theano packages [3]. Keras focuses on present a comprehensive, efficient and extensible way to experiment with deep neural network. The construction of each deep network layer can be simply represented by a line of code.

3.4 Architecture Design

The overall structure of my network is shown in figure 3.1. The data will be preprocessed through two methods respectively (One-hot encoding and PSI-BLAST) before input into the convolution layers. Weights are randomly initiated by Keras for our neural net. SGD is selected as optimizer for our task because of its flexibility. The output between layers is regularized with dropout (0.3, 0.5) to avoid overfitting.

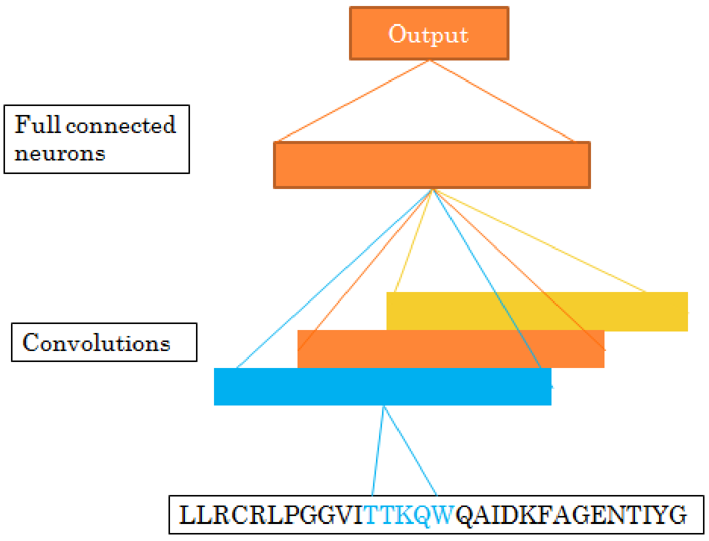


Figure 8: Network architecture

3.4.1. Convolution Layers

The number of convolution layers is one of the important hyperparameters as well as the convolutional kernel size and the number of kernels. For the number of layers, we will use exhaustive search which is to simply specify the hyperparameter manually to find a solution. The number of kernels and its size will be set to commonly used figures and change accordingly.

Model	Q3 Accuracy
Two Dense layers without CNN layer	48.40%
Two Dense layers with 1 CNN layer	61.32%
Two Dense layers with 2 CNN layer	65.50%
Two Dense layers with 3 CNN layer	66.51%
Two Dense layers with 4 CNN layer	66.78%

Table 2: Q3 accuracy under influence of CNN layers

From Table 1, we can see that adding CNN layer can bring a significant improvement to the neural network, over 13% compared to the general neural network that without CNN layer. As the layer increase, the performance keeps impressively growing till the third CNN layer is added. It seems that the impact of CNN layers has been saturated. Moreover, the adding of layer 5 caused exponential growth of training time, which makes the training became impracticable. So our CNN architecture was designed to have 4 CNN layers.

The size of the filter kernels is initially set as 3/5/7/9 which is a quite common configuration in

CNN. But I changed to 3/7/11/20 after reading a few design of Protein Structure Prediction methods which brings a considerable improvement to the CNN (up to 70.77% Q3 accuracy).

3.4.2. Number of hidden units

I randomly initialized the number of hidden units as 32/64/128 which basically 2 to the power of n with hierarchy. But after I studied CNN architecture in a website from Stanford [22], I started to interpret the hidden units as the number of features that we want to learn. Each filter will learn one of the features from the input data and higher level filters will integrate them together as an abstract relationship. It is not saying that increase the number of filters will result in a better accuracy because the number of learnable features is limited. Additionally, the more filters you add, the training time will increase exponentially due to the hierarchical architecture in CNN. Thus, we need to find a balance between the most features learned and less time expenditure.

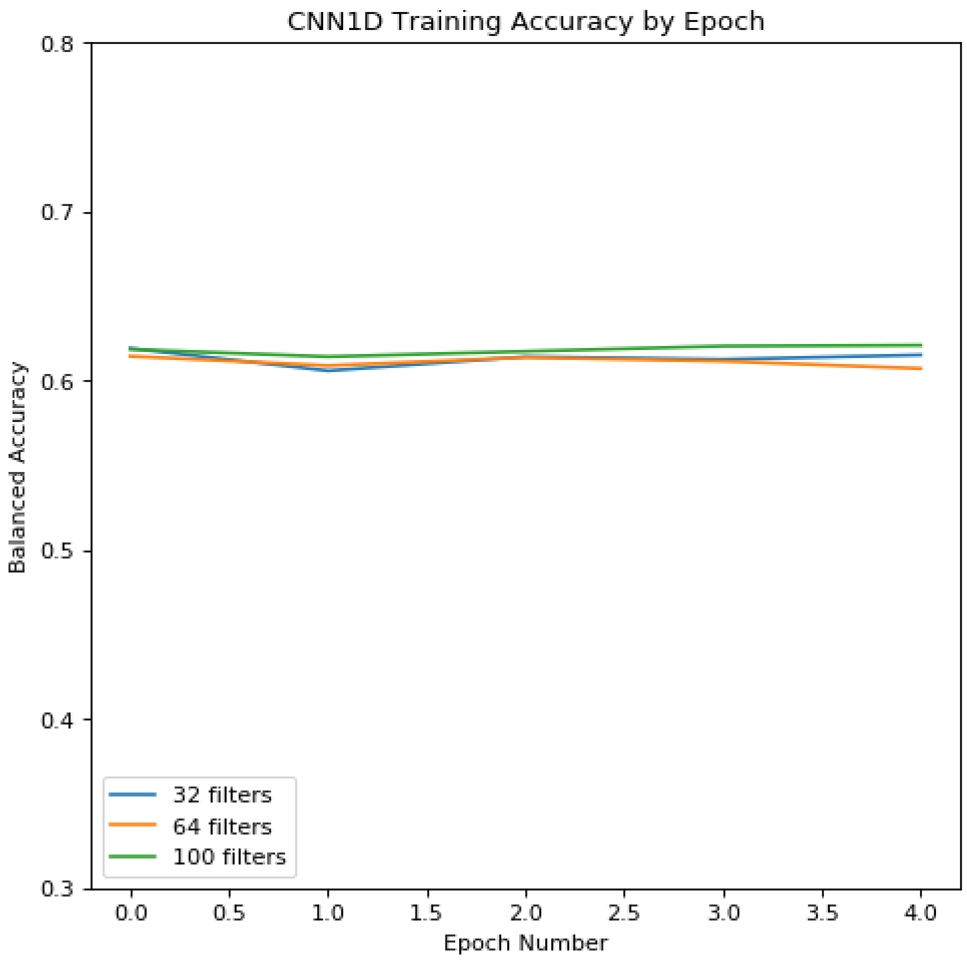


Figure 9: Training on different filters

However, from the graph, we can observe that the number of filters does not improve the training result that much. This may due to two reasons:

1. The training dataset I am using is quite large (over 15,000 proteins) that it can stimulate

the network to fully learn the features inside the dataset regardless of the number of filters used.

2. Another possibility is that the features have been saturated before 32 filters.

Above all, I managed to use 32 filters for my first input CNN layer.

3.5. Choice for loss function

Loss functions are used to represent the divergence between observed distribution and predicted distribution. In our task, we are performing a multiclass classification on one-hot label data, where ‘categorical_crossentropy’ is used in most of the case.

3.6. Choice for optimizer

There are a few optimizers to choose such as Adam, Adadelta, SGD and etc. If your input data is sparse, then you likely achieve the best results using one of the adaptive learning-rate methods (Adam, Adadelta and RMSprop). An additional benefit is that you will not need to tune the learning rate but likely achieve the best results with the default value.

SGD is called Stochastic Gradient Descent which is used for optimize the parameter of the network such as weights. The parameters of SGD such as learning rate and momentum are manually configured. It is the upgraded version of Gradient Descent. Instead of updating the parameters once per batch, it frequently updates with high variance, which enables it to jump to new and potentially better local minima [23]. On the other hand, research shows that SGD still converges the same as normal Gradient Descent when we decrease the learning rate slowly during the training process.

3.7. Model Callbacks

After training a model with datasets, we would seek for some methods to store our model and its weight in order to reuse it; otherwise we have to train it again which would cost a few hours or even days. Therefore, I chose to save my fit results in pickle file and keep weights in hdf5 files. It enables us to reload the pre-trained data and perform further test conveniently.

3.8. Fine-tuning

In this task, we have large datasets and small datasets. Pre-train our model in terms of large and sufficient dataset, and then use the small and typical dataset to adjust it; i.e. fine-tuning. It can bring improvement to the network accuracy because the model has more possibility of capturing features in a given large dataset, and generate the specific abstraction from the features obtained while training on small datasets.

Chapter 4

Results and Conclusion

This chapter will present the performance on several architectures that we described above, compare them to each other and make conclusion.

4.1. Results

During architecture design, we have discussed how we would set up our hyperparameters. And eventually, I chose to have 4 convolution layers, with 3/7/11/13 filter size and 32/64/128/256 filters for each CNN layer.

	Accuracy	Frequency
Q3	.676	1.0
H	.814	.37
E	.443	.23
C	.685	.39

Table 3: Q_3 and Q_c accuracy in 4 CNN layer model with one-hot vector input evaluated on CB513

From the table, we can see that the accuracy of each structure differs. The accuracy of structure type E is irregularly low while type C is closed to the overall performance and type H is performing better than the others. This is not supposed to be the case that the network architecture goes wrong because the CNN layer treat the data equivalently. I assume that one of the possibilities is the pattern of lower frequency have a lower chance to provide sufficient features to be learnt. The ability to learn features of CNN is not only based on large dataset but also balanced classes. And after evaluated the other different architecture, it is still the case, where Q_H is the best and Q_C has poor performance, which proved my assumption. More training data is the way to solve this especially the data of type C.

4.1.1 Effect of Multiple Sequence Alignment

The use of MSA data has dramatically improved the prediction since the matrix can better reflect the relationship between residues and proteins. We can see the improvement after applying PSSM to our input from table 4:

Accuracy	Without PSSM	With PSSM
Q_3	.676	.777
Q_H	.814	.884
Q_E	.443	.596
Q_C	.685	.782

Table 4: Comparison between training with and without PSSM

4.1.2. Effect of Fine-Tuning

Fine-tuning is commonly applied when we have large dataset. Pre-trained on large dataset for a few epochs can obtain more general features through the massive data. After that, we continue the training on small and specific dataset, in order to generate useful feature. This procedure can not only save training time but also improves the performance. Fine-tuning increased our model with approximately 1-2% accuracy.

4.1.3. Training on ASTRAL90 and Testing over a few test sets

Our model trained on ASTRAL90 which contains more than 15,000 proteins and filtered at 95% sequence identity achieves around 80% Q3 accuracy. In order to compare to the other protein structure prediction method, we evaluated our method on RS126, CB513 and ASTRAL30 with different types of input feature embedding.

Method	CB513	RS126	ASTRALL30
CNN with One-hot	70.23%	70.79%	69.73%
CNN with PSSM	79.88%	79.67%	80.30%

Table 5: Q_3 accuracy of different benchmark test sets.

The precision over the benchmark test sets remains relatively stable which means our model did not overfit the training dataset. Another point is that feature extraction performs better when using PSSM to encode input features.

4.1.4. Performance of Blind Test

As we have introduced before, CASP is the Critical Assessment of Techniques for Protein Structure Prediction held every two years. It target on identifying the state-of-the-art method in protein structure prediction among those participants and releasing new annotated proteins. In the competition during the conference, each group will try to predict structures with their own defined method on the given sequences.

The reason why it is called blind test is that, the target of casp datasets are the sequences of latest released. They are never been used in any of the other training dataset before. Moreover, the sequences in casp datasets are manually classified by the CASP organizers with no homologies of known sequence structure.

The composition of casp9 dataset is a series of 203 protein sequences. It contains 37.2% α-Helix, 23.7% β-Sheet and 39% Coil.

	Q_3(%)	Q_H(%)	Q_E(%)	Q_C(%)
Casp9	80.2	87.9	60.2	79.5

Table 6: Blind test on casp9 dataset

Our model shows a high performance on the blind test which means the model has produced sufficient general features to adapt to the new input data which have not seen before.

4.1.5. Comparison to the other Prediction Methods

Method	Q_3(%)	Q_H(%)	Q_E(%)	Q_C(%)
PSI-PRED	81	82	69	86
JPRED	81.6	\	\	\
DeepCNF	84.4	\	\	\
DCRNN	87.8	\	\	\
Our method	80.3	88.4	59.5	79.6

Table 7: Q_3 and Q_r Accuracy of state-of-the-art methods on casp9 dataset

We can observe from table 6 that our method is comparable to the previous state-of-the-art methods. The recent method such as DCRNN has considerable improvement in the prediction. The main reason they have such result is that they applied a technique called feature concatenation. I did not apply it due to time and resources limitations. However, my method still achieves 80% accuracy without feature concatenation and has a much simpler model, which means that there is potential space for further improvement by applying some up-to-date method that remains to be explored based on our model.

4.2. Conclusion

The accuracy of protein structure prediction achieved by this project is fairly high since I am using quite a simple Convolution Neural Network architecture, with only 4 CNN layers, 2 FC layers and around 100 hidden units per layer due to some limitations such as the lack of computing resources and my poor Machine Learning basis. My CNN model performs an average of 80% Q3 accuracy over a few benchmark datasets which is closed to the state-of-the-art method.

The application of Position Specific Scoring Matrix is one big step to our success. This effective implementation has brought 10% additional accuracy over our original feature representation method (One-hot vector).This is a dramatic improvement for our prediction method.

The others are some small details e.g. choice of filters, layers, optimizers, fine-tuning etc. They are all working together, little by little, contribute to our model training and result in this achievement.

Chapter 5

Critical Evaluation

The research goal of this work is to study the previous method of protein structure prediction, learn the principles behind the academic background such as Bioinformatics and Machine Learning and implement my own solution on this task. And I did learn a lot from this task. I guess I would do much better and quicker in the next time developing a deep learning application with more concrete knowledge of CNN structures and parameters.

I choose to use Keras library to implement my design. There are some other libraries such as TensorFlow or Caffe. But I believe that Keras is quite suitable for me not only because I am Machine Learning fresher and I need a quick start from it, but also it enables user to keep code simple and organized without losing the extensibility and efficiency.

I am not expecting my work can be a state-of-the-art method. Whereas, it shows the potential to be the best solution since I have not applied a few techniques on it due to time and resources limitation. I feel satisfied with this result.

There are also some places that I fall short on. Despite I am using a literally large dataset for training, the performance is not as good as I expected especially the accuracy of Beta-Sheet structure. The main reason might be the unbalance in the dataset. In the other hand, the lack of computation resource forced me to compromise by training one epoch only. This might be solved by using another big training set such as CullPDB [29] dataset where sequences with larger than 25% identity with the CB513 dataset was removed.

Appendices

Appendix A

Software Libraries

The software environment is based on Window 10(Linux seems to be better). The libraries I used are developed on the basis of python 2.7 and open-sourced on Github. I used Anaconda which includes all of the libraries I used in this project. They are listed below:

Anaconda	*4.3.1*
Python	*2.7.13*
Numpy	*1.12.1*
SciPy	*0.18.1*
Scikit-Learn	*0.18.1*
Keras	*1.2.2*
Theano	*0.9.0*
Pickle	*1.2.3*

Hardware environment

The capability of training on GPU is one of the important features for deep learning. Moreover, it is especially useful for convolution neural network which has many parallel computations. All the training and testing are performed on GPU NVIDIA GT755m of my own laptop.

Appendix B

Source code

```
# -*- coding: utf-8 -*-
from __future__ import print_function
import numpy as np
import theano
#import theano.tensor as T
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential, Model
from keras.layers import Dense, Dropout, Activation,Flatten
#from keras.layers import Embedding
from keras.layers import Convolution1D, MaxPooling1D, Input, ZeroPadding1D,
GlobalMaxPooling1D
from keras.layers.embeddings import Embedding
from keras.layers.recurrent import LSTM,SimpleRNN
from keras.layers.wrappers import TimeDistributed
from keras.optimizers import Adam, SGD
import pickle
import sys

nb_filter = 64
filter_length = 7
input_dim = 20
embedding_size = 20
input_length = None
padding = filter_length // 2

AAs = ['X','I','L','V','F','M','C','A','G','P','T','S','Y','W','Q','N','H','E','D','K','R']
AAIndexes = {AAs[i] : i for i in range(len(AAs))}

def piecewise_scaling_func(x):
    if x < -5:
        y = 0.0
    elif -5 <= x <= 5:
        y = 0.5 + 0.1*x
    else:
        y = 1.0
    return y
```

```
class ReadData(object):

    @staticmethod
    def encodeAA(residue) :
    # Encode an amino acid sequence

        return [1 if residue == amino_acid else 0
            for amino_acid in ('A', 'R', 'N', 'D', 'C', 'E', 'Q', 'G', 'H', 'I',
                                        'L', 'K', 'M', 'F', 'P', 'S', 'T', 'W', 'Y', 'V')]
        if x not in AAIndexes :
            return 0
        return AAIndexes[x]


    @staticmethod
    def encode_dssp(dssp):
        return [1 if dssp == hec else 0
                for hec in ('H', 'E', 'C')]
        y = 0
        for hec in ('H', 'E', 'C'):
            if dssp == hec:
                break;
            else:
                y += 1
            if y >= 2:
                break;
        return y

    @staticmethod
    def load(filename):
        print('... loading data ("%s")' % filename)
        n=1
        index = 0
        with open(filename, 'r') as f:
            line = f.read().strip().split('\n')
            num_proteins = len(line) // 2
            X = [None] * num_proteins
            Y = [None] * num_proteins
            for line_num in range(num_proteins):
                sequence                =                line[line_num*2]
#Get the amino acid sequence on line line_num*2
                structure        =        line[line_num*2        +        1]
#Get the corresponding secondary structure from line_num*2 +1
                if len(sequence)!=len(structure):
                    print(line_num*2)
#Check if there is a protein has different number of AA and structure
                #if len(sequence)<maxlen:
                #       for i in range(len(sequence),maxlen):
                #               sequence += '0'
                #               structure += '0'

                X[line_num] = [ReadData.encodeAA(residue) for residue in sequence]
                Y[line_num] = np.array([ReadData.encode_dssp(dssp) for dssp in structure])
                index += len(sequence)
            #X = np.array(X)
            #print (X.shape)
            #open('X.txt','w').write(str(X))

        return X,Y,num_proteins,index

    @staticmethod
    def load_pssm(filename, scale=piecewise_scaling_func):
        print('... loading pssm ("%s")' % filename)

        index = 0
        with open(filename, 'r') as f:
            num_proteins = int(f.readline().strip())
            X = [None] * num_proteins
            Y = [None] * num_proteins
            index = 0 #[None] * num_proteins
            for protein_num in range(num_proteins):
                m = int(f.readline().strip())
                sequences = [None] * m
                for line_num in range(m):
                    line = f.readline()
                    sequences[line_num]=[]
                    for i in range(20):
                        s = ''.join(line[i*3: i*3+3]).strip()
                        sequences[line_num].append(scale(float(s)))

                #double_end = ([0.]*20) * (window_size//2)
                #sequences = double_end + sequences + double_end
                X[protein_num] = sequences

                structure = f.readline().strip()
                Y[protein_num] = [ReadData.encode_dssp(dssp) for dssp in structure]

                index += m
        #X[0] = np.array(X[0])
        #print(X[0].shape)
        return X, Y, num_proteins,index

def get_batch(data_in, data_out):
    while True:
        for i in range(len(data_in)):
            input_data = np.array([data_in[i]])
            output_data = np.array([data_out[i]])
            #input_data = np.squeeze(input_data)
            #output_data = np.squeeze(output_data)
            yield (input_data, output_data)

    lengths = list(set(map(len, data_in)))
    data_by_length = {}
    for i, l in enumerate(map(len, data_in)):
        if l not in data_by_length:
            data_by_length[l] = []
        data_by_length[l].append(i)

    while True: # a new epoch
        np.random.shuffle(lengths)
        for length in lengths:
            indexes = data_by_length[length]
            np.random.shuffle(indexes)
            input_data = np.array([data_in[i] for i in indexes])
            output_data = np.array([data_out[i] for i in indexes])
            yield (input_data, output_data)

def get_Xbatch(data_in):
    lengths = list(set(map(len, data_in)))
    data_by_length = {}
    for i, l in enumerate(map(len, data_in)):
        if l not in data_by_length:
            data_by_length[l] = []
        data_by_length[l].append(i)

    while True: # a new epoch
        np.random.shuffle(lengths)
        for length in lengths:
            indexes = data_by_length[length]
            np.random.shuffle(indexes)
            input_data = np.array([data_in[i] for i in indexes])
            yield (input_data)

def get_3LayerModel():
    model = Sequential()

    #We start with a embedding layer which convert our sequence to dense vectors
    #model.add(Embedding(2,embedding_size))

    #model.add(ZeroPadding1D(padding))
    # we add a Convolution1D, which will learn nb_filter
    # word group filters of size filter_length:

    model.add(Convolution1D(nb_filter=32,
                            filter_length=11,
                            input_shape=(None,20),
                            border_mode='same',
                            activation='relu',
                            subsample_length = 1))
    model.add(Dropout(0.3))

    model.add(Convolution1D(64, 7, border_mode='same', activation='relu'))
    model.add(Dropout(0.3))

    model.add(Convolution1D(128, 3, border_mode='same', activation='relu'))
    model.add(Dropout(0.3))

    #model.add(GlobalMaxPooling1D())

    model.add(Dense(128))
    model.add(Activation('relu'))
    model.add(Dropout(0.5))

    model.add(Dense(3))
    model.add(Activation('softmax'))
    return model

def get_4LayerModel():
    model = Sequential()

    #We start with a embedding layer which convert our sequence to dense vectors
    #model.add(Embedding(2,embedding_size))

    #model.add(ZeroPadding1D(padding))
    # we add a Convolution1D, which will learn nb_filter
```

```
        # word group filters of size filter_length:

        model.add(Convolution1D(nb_filter=32,
                                filter_length=20,
                                input_shape=(None,20),
                                border_mode='same',
                                activation='relu',
                                subsample_length = 1))
        model.add(Dropout(0.3))

        model.add(Convolution1D(64, 11, border_mode='same', activation='relu'))
        model.add(Dropout(0.3))

        model.add(Convolution1D(128, 7, border_mode='same', activation='relu'))
        model.add(Dropout(0.3))

        model.add(Convolution1D(200, 3, border_mode='same', activation='relu'))
        model.add(Dropout(0.3))

        #model.add(GlobalMaxPooling1D())

        model.add(Dense(200))
        model.add(Activation('relu'))
        model.add(Dropout(0.5))

        #model.add(Dense(400))
        #model.add(Activation('relu'))
        #model.add(Dropout(0.5))

        model.add(Dense(3))
        model.add(Activation('softmax'))
        return model
    def train_OneHotLabels():

        X_train, Y_train, num_proteins, index_train = ReadData.load('casp9.data')
        X_valid, Y_valid, num_validproteins, index_valid = ReadData.load('RS126.data')
        X_eval, Y_eval, num_evalproteins, index_eval = ReadData.load('RS126.data')

        model = get_4LayerModel()
        model.compile(loss='categorical_crossentropy', optimizer='adadelta', metrics=['accuracy'])
        #SGD(lr=1e-3, momentum=0.9)Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-08)
        #model.summary()
        #nb_samples = X_train[0].shape
        #test_size = int(0.1 * num_proteins)
        #X_test, Y_test =    X_train[-test_size:], Y_train[-test_size:]
        #print(Y_train[0])
        #print('\n\nModel  with  input  size  {},  output  size  {},  {}  conv  filters  of  length
    {}'.format(model.input_shape, model.output_shape, nb_filter, filter_length))

        fit_results = model.fit_generator(get_batch(X_train, Y_train),
                                          samples_per_epoch=index_train,
                                          validation_data=get_batch(X_valid, Y_valid),
                                          nb_val_samples=index_valid,
                                          nb_epoch=5)

        #input = np.array(X_eval[0])
        #input = np.expand_dims(input, axis=0)
        #y_predict = model.predict(input)

        #eval = model.evaluate_generator(get_batch(X_eval,Y_eval),
        #                                index_eval)
        #print(eval)
        print('Saving model....')
        sys.setrecursionlimit(10000)
        pickle.dump(fit_results, open('files/nodestest100_fit_results.pkl', 'w'))
        while False:


            model.save_weights('files/CNN1D2Dense_model_weights.hdf5')
            json_string  =  model.to_json()                                  #Equal  to:
    json_string = model.get_config()
            open('files/CNN1D2Dense_architecture.json','w').write(json_string)

        #Loading model and weights
        #model = model_from_json(open('my_model_architecture.json').read())
        #model.load_weights('CNN1D_model_weights.hdf5')

    def get_2D4LayerModel():
        model = Sequential()
        #model.add(Embedding())
        #model.add(ZeroPadding2D((1,1)))
        model.add(Convolution2D(32,          3,          20,          border_mode='same',
    input_shape=(None,None,20) ,activation='relu'))
        model.add(Dropout(0.5))
        #model.add(MaxPooling2D())
        model.add(Convolution2D(64, 3, 11, border_mode='same' ,activation='relu'))
        model.add(Dropout(0.3))
```

```
	model.add(Convolution2D(128, 3, 7, border_mode='same' ,activation='relu'))
	model.add(Dropout(0.3))
	model.add(Convolution2D(128, 3, 3, border_mode='same' ,activation='relu'))
	model.add(Dropout(0.3))
	#model.add(Flatten())
	#model.add(Dense(32,input_shape=(20,),activation='relu'))
	#model.add(Dropout(0.5))

	model.add(Dense(128,activation='relu'))
	model.add(Dropout(0.5))
	model.add(Dense(3))
	model.add(Activation('sigmoid'))
	return model

def get_label(prediction):
	Y_label = []

	for residues in prediction:
		j = 0
		for n in residues:
			if n==1:
				break
			if j==2:
				break
			j += 1
		Y_label.append(j)

	return Y_label

def get_max(prediction):
	Y = []
	for residues in prediction:
		max_value = max(residues)
		max_index = residues.index(max_value)
		l = []
		for n in residues:
			if n==max_value: l.append(1)
			else: l.append(0)
		Y.append(l)
	return Y
def print_matrix(X, Y, model):
	y_predict = []
	y_label = []
```

```
    for i in range(num_proteins):
        X_input = np.expand_dims(X[i], axis=0)
        X_input = np.expand_dims(X_input, axis=0)
        y_out = model.predict(X_input)
        y_out = np.around(y_out)
        y_out = y_out.astype(int)
        y_predict += np.squeeze(y_out).tolist()
        y_label += np.squeeze(Y[i]).tolist()
    y_predict = get_max(y_predict)
    y_predict = get_label(y_predict)

    y_label = get_label(y_label)
    print(confusion_matrix(y_label,y_predict))
def model_predict()
   print('Loading CNN model....')
   model = model_from_json(open('files/CNN2D4Layers_architecture.json').read())
   print('Loading model weights....')
   model.load_weights('files/CNN2D4Layers_model_weights.hdf5')
   model.compile(loss='categorical_crossentropy', optimizer='adadelta', metrics=['accuracy'])
   X_test, Y_test, num_proteins,index = CNN2D.ReadData.load_pssm('casp9.pssm')
   #X_valid, Y_valid, num_validproteins,index = CNN2D.ReadData.load_pssm('CB513.pssm')
   #model.fit_generator(CNN2D.get_batch(X_valid,Y_valid),index,
   #                         nb_epoch=3)
   #model.save_weights('files/CNN2D4Layers_model_weights.hdf5')
   #save model weights

   #eval = model.evaluate_generator(CNN2D.get_batch(X_test,Y_test),index)
   #print(eval)
   print_matrix(X_test, Y_test, model)

def get_2D2LayerModel():
    model = Sequential()
    #model.add(ZeroPadding2D((1,1)))
    model.add(Convolution2D(32,          3,          7,          border_mode='same',
input_shape=(None,None,20) ,activation='relu'))
    model.add(Dropout(0.5))
    #model.add(MaxPooling2D())
    model.add(Convolution2D(64, 3, 3, border_mode='same' ,activation='relu'))
    model.add(Dropout(0.3))

    #model.add(Flatten())
    #model.add(Dense(32,input_shape=(20,),activation='relu'))
    #model.add(Dropout(0.5))
    model.add(Dense(128,activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(3))
    model.add(Activation('sigmoid'))
    return model

def get_2D3LayerModel():
    model = Sequential()
    #model.add(ZeroPadding2D((1,1)))
    model.add(Convolution2D(32,          3,          11,          border_mode='same',
input_shape=(None,None,20) ,activation='relu'))
    model.add(Dropout(0.5))

    model.add(Convolution2D(64, 3, 7, border_mode='same' ,activation='relu'))
    model.add(Dropout(0.3))

    model.add(Convolution2D(128, 3, 3, border_mode='same' ,activation='relu'))
    model.add(Dropout(0.3))
    #model.add(MaxPooling2D())

    model.add(Dense(128,activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(3))
    model.add(Activation('sigmoid'))
    return model
def split_pssmfile():
    f = open('valid.pssm','r')
    result = list()
    num_train = 1000
    num_test = 100
    result1 = list()
    num_proteins = int(f.readline())
    result.append(str(num_train)+'\n')
    result1.append(str(num_test)+'\n')
    print('Spliting' , num_train,'training and', num_test ,'testing proteins from ', num_proteins,
'samples')
    for n in range(num_train):
        m = int(f.readline())
        result.append(str(m)+'\n')
        for i in range(m):
            line = f.readline()
            result.append(line)
        line = f.readline()
        result.append(line)
        for n in range(num_test):
                m = int(f.readline())
                result1.append(str(m)+'\n')
                for i in range(m):
                        line = f.readline()
                        result1.append(line)
                line = f.readline()
                result1.append(line)
        #print result
        f.close()
        open('train1000.pssm', 'w').write('%s' % ''.join(result))
        open('testing100.pssm', 'w').write('%s' % ''.join(result1))

def extract_pssm(pssm_file):
        for line in pssm_file:
                l.append(line[:-1])
        l.pop()
def extract_dssp(dssp_file):
        return dssp_file.read()
       l = list()
def extract_script(dssp_file):
       with open('list', 'r') as name_list:
               for line in name_list:
                       name = line[:-1]
                       with open('dssp/'+name+'.dssp', 'r') as dssp_file:
                               dssp = extract_dssp(dssp_file)

                       l.append(str(len(dssp)))
                       #print len(dssp)-1

                       with open('pssm/'+name+'.data', 'r') as pssm_file:
                               extract_pssm(pssm_file)

                       l.append(dssp)
               open('CB513.pssm','w').write('%s' % '\n'.join(l))
```

Bibliography

[1] Michael A. Nielsen, “Neural Networks and Deep Learning” 2015. [Online]. Available: http://neuralnetworksanddeeplearning.com/index.html .

[2] Sheng Wang, Jian Peng, Jianzhu Ma & Jinbo Xu, “Protein Secondary Structure Prediction Using Deep Convolutional Neural Fields” 2016. [Online]. Available: http://www.nature.com/articles/srep18962.

[3] LISA lab, “Theno Docs” 2008. [Online]. Available: http://deeplearning.net/software/theano/.

[4] Francois Chollet. Keras: Theano-based deep learning library. https://github.com/ fchollet/keras, 2015.

[5] Breda A, Valadares NF, Norberto de Souza O, et al. Protein Structure, Modelling and Applications. 2006 May 1 [Updated 2007 Sep 14]. In: Gruber A, Durham AM, Huynh C, et al., editors. Bioinformatics in Tropical Disease Research: A Practical and Case-Study Approach [Internet]. Bethesda (MD): National Center for Biotechnology Information (US); 2008. Chapter A06. Available from: https://www.ncbi.nlm.nih.gov/books/NBK6824/

[6] J. Moult, K. Fidelis, A. Kryshtafovych, and A. Tramontano, “Critical assessment of methods of protein structure prediction (CASP)–round IX.” Proteins, vol.79 Suppl 10, pp. 1–5, 2011.

[7] Koonin EV (2005). "Orthologs, paralogs, and evolutionary genomics". Annual Review of Genetics. 39: 309–38. doi:10.1146/annurev.genet.39.073003.114725. PMID 16285863.

[8] Bhagwat M, Aravind L. PSI-BLAST Tutorial. In: Bergman NH, editor. Comparative Genomics: Volumes 1 and 2. Totowa (NJ): Humana Press; 2007. Chapter 10. Available from: https://www.ncbi.nlm.nih.gov/books/NBK2590/

[9] Gribskov, M., Mclachlan, A. D. and Eisenberg, D. (1987). Profile Analysis Detection Of Distantly Related Proteins. Proceedings of the National Academic of 212 Science. USA. 84:4355-4358

[10] Hubel, D. H.; Wiesel, T. N. (1968-03-01). "Receptive fields and functional architecture of monkey striate cortex". The Journal of Physiology. 195 (1): 215–243. doi:10.1113/jphysiol.1968.sp008455. ISSN 0022-3751. PMC 1557912 Freely accessible. PMID 4966457.

[11] Krizhevsky, A.; Sutskever, I.; Hinton, G. E. (2012). "Imagenet classification with deep convolutional neural networks" (PDF). Advances in Neural Information Processing Systems. 1: 1097–1105.

[12]J. A. Cuff and G. J. Barton, "Application of multiple sequence alignment profiles to improve protein secondary structure prediction." Proteins, vol. 40, no. 3, pp. 502–511, Aug. 2000.

[13][Rost and Sander, 1993] Burkhard Rost and Chris Sander. Prediction of protein secondary structure at better than 70% accuracy. Journal of molecular biology, 232(2):584– 599, 1993.

[14]A. Murzin, S. E. Brenner, T. Hubbard, and C. Chothia. Scop: A structural classification of proteins database and the investigation of sequences and structures. J. Mol. Biol., 247:536-540, 1995.

[15]J. M. Chandonia and M. Karplus, "The importance of larger data sets for protein secondary structure prediction with neural networks." Protein science : a publication of the Protein Society, vol. 5, no. 4, pp. 768–774, Apr. 1996.

[16]"BLAST+" [Online]. Available: ftp://ftp.ncbi.nlm.nih.gov/blast/executables/ .

[17]Ehsaneddin Asgari and Mohammad RK Mofrad. Protvec: A continuous distributed representation of biological sequences. arXiv preprint arXiv:1503.05140, 2015.

[18]Kloczkowski, A., Ting, K. L., Jernigan, R. L. and Garnier, J. (2002). Combining The GOR V Algorithm With Evolutionary Information For Protein Secondary Structure Prediction From Amino Aid Sequence. Proteins: Structure, Function, and Genetics, Supplement. 49: 154-166

[19]Matthews, B. B. (1975). Comparison of The Predicted and Observed Secondary Structure Of T4 Phage Lysozyme. Biochima et Biophysica Acta. 405(2): 442- 451.

[20]Srivastava N, Hinton G E, Krizhevsky A, et al. Dropout: a simple way to prevent neural networks from overfitting[J]. Journal of Machine Learning Research, 2014, 15(1): 1929-1958.

[21]Rosasco, L.; De Vito, E. D.; Caponnetto, A.; Piana, M.; Verri, A. (2004). "Are Loss Functions All the Same?" (PDF). Neural Computation. 16 (5): 1063–1076. doi:10.1162/089976604773135104. PMID 15070510.

[22] "CS231n Convolutional Neural Networks for Visual Recognition"[Online]. Available: http://cs231n.github.io/convolutional-networks/.

[23]Sebastian, R. An overview of gradient descent optimization algorithms. arXiv:1609.04747 [cs.LG]

[24]Pheox. Secondary protein structure prediction based on cellular automata [Online]. Available: https://github.com/Pheox/ca-protein-predictor

[25][Cuff and Barton, 1999] James A Cuff and Geoffrey J Barton. Evaluation and improvement

of multiple sequence methods for protein secondary structure prediction. Proteins: Structure, Function, and Bioinformatics, 34(4):508– 519, 1999.

[26]Anfinsen, C. B. (1973). Principles That Govern The Folding Of Protein Chains. Science. 181: 223-230.

[27]Stephen, R. Holbrook, Steven, M., Muskal and Sung-Hou Kim. (1990). Predicting Protein Structural Features With Artificial Neural Networks. in: Lawrence Hunter ed. Artificial Intelligence and Molecular Biology. UK.

[28]Kabsch W, Sander C (1983) Dictionary of protein secondary structure: pattern recognition of hydrogen-bonded and geometrical features. Biopolymers 22: 2577–2637.

[29]Zhou, J., and Troyanskaya, O. G. 2014. Deep supervised and convolutional generative stochastic network for protein secondary structure prediction. arXiv preprint arXiv:1403.1347.

[30]Matthews, B. W. (1975). "Comparison of the predicted and observed secondary structure of T4 phage lysozyme". Biochimica et Biophysica Acta (BBA) - Protein Structure. 405 (2): 442–451. doi:10.1016/0005-2795(75)90109-9.

[31]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. [PDF].

[32]Bergstra, James; Bengio, Yoshua (2012). "Random Search for Hyper-Parameter Optimization" (PDF). J. Machine Learning Research. 13: 281–305.