UNIVERSITY OF SOUTHAMPTON
Faculty of Physical Sciences and Engineering
School of Electronics and Computer Science

A project report submitted for the award of
BSc Computer Science

Supervisor: Professor Mahesan Niranjan
Examiner: Dr Julian Rathke

Knowledge distilation and Model compression on CNN for Protein Secondary Structure Prediction

by Ioan Ieremie

April, 2018

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF PHYSICAL SCIENCES AND ENGINEERING

A project report submitted for the award of BSc Computer Science

by Ioan Ieremie

Predicting the secondary structure of the protein is one of the most challenging problem in Bioinformatics. Experimentally determining the protein structure is costly and inefficient, therefore multiple machine learning algorithms have been published that predict the secondary structure from the amino-acid residues chain.
The purpose of the project will be to train different secondary structure predictors and to understand if there exists significant asymmetry in the inputs that explain the predictions. The aim of this work is to observe if systematic trends appear on large datasets - especially on Prokaryotes and Eukaryotes where the protein synthesis mechanism is different.

– MODIFY THIS ONCE THE REPORT IS DONE AND THE MAIN IDEA OF IT IS UNDERSTOOD

Contents

1 Introduction **1**

2 Background of the problem **3**

2.1 Dictionary of protein secondary structure . . . 3

2.2 Generative Stochastic Network . . . 3

2.3 Deep Convolutional Neural Fields . . . 4

2.4 Cascaded Convolutional and Recurrent Neural Networks . . . 5

2.5 Saliency Map on CNNS . . . 6

2.6 CNN arhitecture overview . . . 7

3 Final approach **11**

3.1 Libraries and tools . . . 11

3.2 Dataset . . . 11

3.2.1 PSSM - position-specific scoring matrix . . . 12

3.3 Evaluation methods . . . 13

4 Implementation, experiments and results **17**

4.1 Data prepossessing and input discussion . . . 17

4.2 CNN depth exploration . . . 18

4.3 K-fold accuracy . . . 20

4.4 Improving the architecture . . . 23

4.4.1 Accuracy on CB513 . . . 23

4.5 Model Assembly . . . 25

4.6 Comparison with state-of-the-art methods . . . 27

Bibliography **31**

List of Figures

2.1 Comparison between a regular neural network and CNN 8
2.2 1D convolution of a AA chain 9

3.1 Dataset distribution 14

4.1 Q8 accuracy of models with different numbers of layers 19
4.2 Q8 accuracy of models with the same numbers of parameters 20
4.3 Q8 accuracy of the 5 layer network with various number of filters 21
4.4 Accuracy variation on 10 fold cross-validation 21
4.5 Example of a window of size 19 split into two windows of size 9 22
4.6 Accuracy variation on 10 fold cross-validation - window of size 9 22
4.7 Convolution architecture 24
4.8 Example of a AA chain that contains padding 24
4.9 Q8 Accuracy variation of 20 models on CB513 25
4.10 Model assembly performance on the test set CB513 26
4.11 Confusion matrix on the Q8 classification problem 27

List of Tables

4.1 Q8 accuracy on CB513 using different padding sequences 25
4.2 Our CNN vs current published models using the CB513 benchmark 28
4.3 Comparison of precision and recall values 29
4.4 comparison of precision and recall values - assembly of networks 29

Listings

Chapter 1

Introduction

hehehe

Chapter 2

Background of the problem

2.1 Dictionary of protein secondary structure

Protein secondary structure is in essence a way of classifying each amino-acid present in the sequence as belonging to a particular class. The work of Wolfgang Kabsch and Christian Sander (1983) suggests that for a successful classification of the amino-acid sequence there must be a physical meaningful description of the secondary structure. In their paper, the problem of classification is seen as a pattern-recognition process on hydrogen-bonds and geometrical features that have been previously extracted from x-ray coordinates.
Prior to their work, other researchers have published different structure assignments. The assignments of the secondary structure appear in the Brookhaven Protein Data Bank [1] however, some of the results are incomplete and can be considered subjective. The final work proposes a set of patterns which are simple but powerful enough to discriminate among different types of secondary structures. The accuracy of the dictionary revolves around the resolution of each structure, therefore the process of assigning labels is determined by the accuracy (resolution) of the measurements taken from the x-ray. The most valuable information found in this paper is in regards to the idea that there are 21 different types of amino-acids and each one can be assigned one of 8 classes: 3 types of helices (H, G, I), 2 types of β-sheets (B, E), 3 types of coils (T, S, L). [2]

2.2 Generative Stochastic Network

The work of Jian Zhou and Olga G. Troyanskaya (2014) presents a supervised generative stochastic network which is fed by a 1D Convolutional Neural Network layer (1 protein sequence as input) and learns a Markov chain. The main advantage over a unsupervised

GSN is that it focuses the resources on dependencies that have higher importance on the desired output/prediction. An important aspect of this approach has to do with the hierarchical representation which enables the architecture to deal with long sequences of amino-acids - there are two channels considered as input, one for the features (X) and one for the labels (Y) and only the last one is being corrupted (injecting noise to input and intermediate computation).

The dataset used for training and testing the model was created using the PISCES Cull PDB server and the final result consisted of 6128 proteins (5600 Training, 256 Validation, 272 Testing). All the solved amino-acid sequences have a resolution better than 2.5Å and the length varies between 50 and 700. Apart from this, the PSSM values ($n \times b$ matrices, with n representing the sequence length and b the number of amino-acid types) have been computed using PSI-BLAST against UniRef90 database and the results have been normalised to a 0-1 range using the sigmoid function.

The Q8 accuracy on the test set was 72.1 ± 0.6% with high scores when it comes to predicting the states (H, E, L, T) and low scores on the states (S, G, B) - this is due to an imbalanced dataset (the state 'I' was not present in the test set so the accuracy was 0). Performing validation on the public benchmark dataset CB513 led to an accuracy of 0.664 which outperformed the current best. [3]

2.3 Deep Convolutional Neural Fields

Considerable improvement compared to GSN on the Q8 and Q3 accuracy has been made by Sheng Wang et al. (2016). They developed a new machine learning method called Deep Convolutional Neural Fields. DeepCNF introduces a combined version of conditional neural fields and Deep Convolutional Neural Networks which manages to capture relations between amino-acids at adjacent positions. The method finds complex relations between the input features and the output classes - this is due to the fact that a convolutional deep neural network will capture more information coming from longer amino-acid sequences than a normal deep neural network when using the same window size.

A few key differences from Jian Zhou and Olga G. Troyanskaya (2014) can be observed, starting with the fact that instead of having two channels as input (features and labels), DeepCNF only offers the secondary structure features to the visible layer. Apart from this, the DeepCNF model is explicitly learning the amino-acid interdependency while Sheng Wang et al. (2016) does not. This is due to the fact that each amino-acid label probability is calculated directly compared to the sampling method of Zhou's work.

When it comes to accuracy, there is a clear improvement to the previous work especially on the Q3 classification. DeepCNF outperforms other state-of-the-art methods, but it also does a better job at labeling structure types that are known to be hard to predict such as (S, T, L).
The dataset used for training is similar to the one used by Zhou and it is publicly available (Cull PDB). The dataset is split differently for this work - there is a similarity less than 25% between training data (≈ 5600) and test data (≈ 500). Besides this dataset, a number of four other publicly available datasets have been used for testing purposes and for comparing the accuracy with other models: CB513(513 proteins), CASP10(123 proteins), CASP11(105 proteins) and CAMEO test proteins (175 after filtering) from the last 6 months (2014/12/05 - 2015/05/29). An important note here is that every dataset mentioned above is generated using the dictionary of protein secondary structure (Wolfgang Kabsch and Christian Sander). An interesting addition to the Q3 accuracy measurement is brought by the SOV score (Segment of overlap) which assigns a lower score to a prediction which had wrong labels at the middle of the sequence compared to one that made incorrect classification of the amino-acids at the terminals.

The accuracy on the Q8 classification of DeepCNF on CullPDB results in a value of 75.2% which represents an increase of 3.1% compared to ICML2014, while the Q3 accuracy yields a value of 85.4%. In terms of SOV score, the model has an accuracy of 86.7% - that is, DeepCNF makes more meaningful predictions on the protein sequences and this is due to the use of deep neural networks.

Further work has been done on observing the differences in accuracy that appear when the models are tested on homologous information. Neff is a way of measuring how sparse the amino-acid sequence is - a small Neff value indicates that the test protein has a sparse sequence while a high Neff value indicates that the protein secondary structure contains a considerable amount of homologous information. The results of running the model on sparse sequences suggested that there is no significant improvement to other methods and that it is still complicated to predict secondary structure from the primary structure.[4]

2.4 Cascaded Convolutional and Recurrent Neural Networks

Zhen Li and Yizhou Yu (2016) introduced a method which also sets up a new benchmark in terms of accuracy for secondary structure prediction by proposing a combination of

CNN and RNN architectures - it is known that local contexts, in particular amino-acid neighbours, represent a good way to predict the labels and that CNNs have proven to be effective in picking up such patterns. RNNs are another type of neural networks that contain loop connections and in the case of secondary structure prediction are able to find dependencies that appear at a larger distance than the local contexts (global context) - this is particularly implemented using multiple bidirectional GRU (gate recurrent units) which represent the most important part of the model.
The input is represented by two channels, one containing the sequence features and one containing the profile features. Different from other approaches using 1 hot encoded values, here the first input is being transformed by the feature embedding layer - sparse sequences are mapped into denser feature vectors. The new input is then fed to the CNN structure which extracts the local features, then the concatenated local features become input to the three BGRU layers.

The data set used for training was CB6133 (PISCES CullPDB) which contains 6128 proteins, a number of which 256 represent the validation data and 272 the testing data. For testing and comparison reasons, the same datasets mentioned previously were used, one of which is the well known benchmark CB513. A filtered version of the CB6133 has been created in order to remove the proteins that shared more than 25% similarity with the test dataset. Training on the original CB6133 and testing on the 272 proteins gives a Q8 accuracy of 73.2 $\pm$ 0.6%, 1.1% more accurate than GSN. Using the filtered version of the dataset and testing on CB513 gives a Q8 accuracy of 69.4 $\pm$ 0.5%, 1.1% higher than the previous state of the art method, DeepCNF.[5]

2.5 Saliency Map on CNNS

Guillermo Romero Moreno et al. (2019) takes another approach to the well known Bioinformatics problem of secondary protein prediction. To this point, most of the efforts have been made to improve the accuracy of the predictions by introducing and training new deep neural architectures. In this paper, the main idea is to find a way of interpreting the decisions made by the architectures using Saliency map techniques. The model used in the paper has been trained using the dataset produced by Zhou and Troyanskaya and the Q8 accuracy yields a value of 69.23% - not far from the state-of-the-art methods (the model has been kept simple to keep saliency map calculations to low levels).

Every input to the model represents a amino-acid sequence where each element is represented by 42 features (21 features for representing the type of amino-acid in 1 hot encoded form and 21 values for the PSSM features). Furthermore, each position in the protein sequence generates a Saliency map that spans across the entire feature space and 9 positions to each side - this has to do with the chosen window size which is 19. Each possible output label will then have a Saliency map value, so the final size is $8 \times 42 \times 19$.

This paper highlights the importance of PSSM values on predicting the secondary structure using Saliency maps. This is done by separating each Saliency map into sequence features and profile features and the results are summed to obtain an overall score. These score suggests that PSSM values have 4 times or more importance in the majority of positions. This finding is backed up by training a model with only PSSM values and the final results show an improvement of 0.14% compared to the original Q8 accuracy - this suggest that the one-hot-encoded amino-acids are of no importance in the presence of PSSM values.

Another useful insight that Saliency maps offer is the study of spatial importance, in other words, where the machine learning model looks when it makes predictions. In the paper, for every input labeled correctly, the Saliency map is summed together on the feature space and that generates a vector of dimension 19. For every class, all the values are averaged together and the final results suggest what were the positions most relevant when predictions were made. In the case of the class 'H' there is some periodicity appearing at the right side and large asymmetry - this can suggest a strong dependency that appears at the protein chain creation. These findings have been validated by running another version of the model which ignores the input from the left positions and then from the right positions (setting convolutional filters parameters to zero). Using the left positions gives an accuracy of 80.81% while the right positions achieves 86.60% (on H and non-H classification). [6]

2.6 CNN arhitecture overview

With the introduction of the previous deep learning techniques, current efforts in secondary protein structure are focused on convolutional/recurent neural networks. – MAYBE WRITE HERE ABOUT SOME OF THE PAPERS AND CITE THEM–. This section is a technical overview of CNN based on the open course offered by the University of Stanford - CS231n: Convolutional Neural Networks for Visual Recognition.

CNN are similar to normal neural networks meaning that they contain neurons that have weights and biases that need to be learned. A neuron receives the input and performs a dot product with the current parameters and outputs the result that can be followed by an activation function. The entire network uses a differentiable score function along with a loss function that is used in the last fully connected layer (eg.Softmax).

A regular neural net consists of multiple fully connected layers. The input layer is followed be a series of "hidden layers" and the final output layer in a classification problem represents the class scores. Each layer is composed of a number of neurons that are fully connected to the previous layer and do not share any parameters with other neurons. The problem with this architectures is that the number of learnable parameters increases with the addition of new layers and soon this can become unmanageable - when working with images, an input image of resolution 200x200 pixels in the RGB colour space would require a neuron with a number of 200x200x3 = 120,000 weights. On the other hand, CNN consider the input to be an image and all the following layers have the neurons arranged in 3 dimensions - height, width and depth. This allows neurons to be connected only to a specific region of the previous layer. (Figure 2.1)

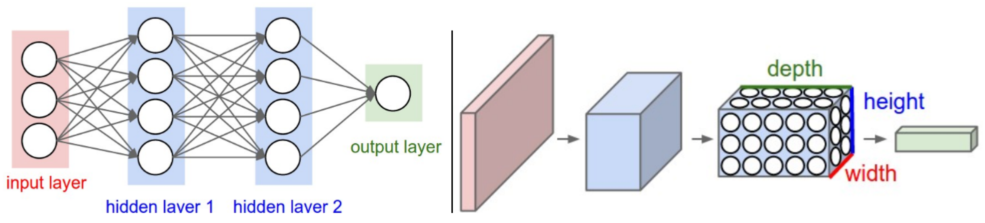


Figure 2.1: Comparison between a regular neural network and CNN, sourced from Stanford-University (CS231n). On the left side, each neuron is fully connected to all the neurons in the previous layer. On the right side, neurons in the activation map are only connected to a section of the previous layer.

One of the most important part of the CNN is the convolutional layer. It consists of a series of filters that represent a set of learnable parameters. Each filter is a spatial "patch" (width and height) that extends over the entire depth of the input - for example, the first layer of CNN that does image classification could contain a filter of size 5x5x3 that has depth 3 in order to accommodate the entire depth of the input(RGB colour space). During the forward pass in the architecture, a filter would slide across the width and height of the input and perform a dot product with the current weights(convolution) - this would create an 2D activation map that contains the response of the filter at each position of the input. The idea behind this process is that the network would look for the same pattern over the entire span of the input and therefore use the same parameters (parameter sharing). Increasing the number of filters per convolutional layer would add new activation maps and the stacked result would be the depth of the output.

Moving away from image classification, in protein secondary structure prediction, the data is represented by a chain of amino-acid residues that form the protein. The main key difference here is that for this particular problem 1D convolutional layers are used. In Figure 2.2, the filter(feature detector) spans across the entire representation of the amino-acid and is of size 3, meaning that it convolves with the feature vectors of 3 amino-acids.

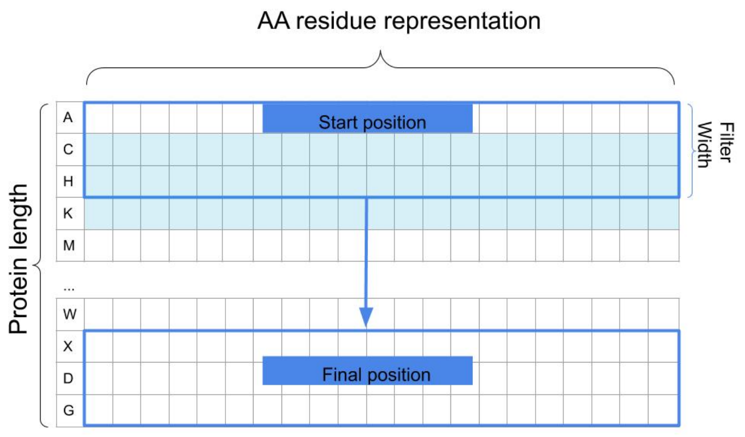


Figure 2.2: Convolution on a AA chain with a filter of size 3. Each AA is encoded as a 21 dimensional vector which can be 1 Hot encoded or containing the PSSM values. The filter slides across the sequence by moving with 1 position - it gathers information from 3 AAs. In this example padding is not used so the output volume will change.

Chapter 3

Final approach

3.1 Libraries and tools

This project has been build using Python and the *JupyterLab IDE*. The main libraries used for creating the model architectures and for training-testing the models are *Keras* [7] and *Tenserflow* [8]. From a hardware perspective, a combination of multiple machines and servers have been used to cope with the high number of experiments. At the beginning of the project, most of the training was done with the help of the Google Colab platform (Tesla K80 GPU - 2496 CUDA cores, 12GB GDDR5 VRAM). Two other computing resources used were offered by the University of Southampton and accessed through remote desktop connection (2 NVidia RTX2070 GPU cards with 2,304 CUDA cores each).

3.2 Dataset

Two main datasets are used for developing the architecture in this report - CULLPDB and CB513 that are made available by Jian Zhou and Olga G. Troyanskaya (2014). CULLPDB contains protein sequences that have been dowloaded from Protein data bank [9] using the PISCES Cull PDB server [10]. The final result consists of solved proteins structures that have a resolution better that 2.5 Å and share a similarity of less than 30%. The protein length has been limited to the range $[50, 700]$ and no discontinuous chains appear in the dataset.
CullPDB contains 6128 proteins and this has been further filtered to remove those chains that share more than 25% similarity with the test set CB513. The final configuration consists of 5534 proteins (1,183,318 AA) split into 5278 training and 256 validation similar to [11] [12].
CB513 is the test set that is usually used as benchmark for model comparison - it contains 513 protein chains but with 514 entries because the last protein is split in two.

The following structure shows the dimensionality of the filtered dataset.

$$5534\ \text{proteins(N)} \left\{ \overbrace{\begin{pmatrix} p_{11} & p_{12} & \cdot & \cdot & \cdot & p_{1,a-1} & p_{1,a} \\ p_{21} & p_{22} & \cdot & \cdot & \cdot & p_{2,a-1} & p_{2,a} \\ \cdot & & & & & & \\ \cdot & & & & & & \\ \cdot & & & & & & \\ p_{n,1} & p_{n,2} & \cdot & \cdot & \cdot & p_{n,a-1} & p_{n,a} \end{pmatrix}}^{700\ \text{amino} - \text{acids(A)}} \right. \text{(Dataset dimensions)}$$

where each p value in the matrix represents one amino-acid that has the following 57 features:

$[0:22)$: amino-acid residues that appear in one hot encoded form

$[22:31)$: secondary structure labels - (L,B,E,G,I,H,S,T)

$[31:33)$: N and C terminals - not used

$[33:35)$: relative and absolute solvent accessibility - not used

$[35:57)$: sequence profile (PSSM) - the values belong to the range $[0,1]$

The elements found at positions 21 and 30 mark the end of the protein sequence. Each protein has a sequence which ranges from 50 to 700 amino-acids and zero padding is used such that each protein has the same length.[3]

3.2.1 PSSM - position-specific scoring matrix

PSSMs are used in bioinformatics as representations of motifs that appear in biological sequences. Each row in the PSSM represents 1 type of amino acid (20 types) and each column is for each position present in the pattern (700 in CullPDB dataset). The first step in obtaining the PSSM is to create the position frequency matrix PFM that counts the occurrence of each AA at every location. Then, the position probability matrix PBM is computed by dividing the number of occurrences by the number of sequences such that the final values are normalised. Equation 3.1 offers the formal definition of the matrix for a set P of size N aligned sequences (proteins in our case) of length l (700).

$$M_{k,j} = \frac{1}{N}\sum_{i=1}^{N} T(P_{i,j} = k) \tag{3.1}$$

Where:

k:	is one 20 types of amino-acids
$M_{k,j}$:	is the value for amino-acid k at position j.
$P_{i,j}$:	is the amino-acid found at position j in protein P_i.
$T()$:	is a function that denotes the ground truth and returns 0 or 1

Finally, PSSM are obtained by transforming the raw probabilities into log likelihoods using a background model b that assumes how frequently each-amino acid appears in the set. The simplest form of the model is to assume that each AA appears equally, $b_k = \frac{1}{|k|}$.

$$M_{k,j} = \log_2 \frac{M_{k,j}}{b_k} \tag{3.2}$$

If the set of sequences is too small, values of 0 could appear in the matrix (no occurrences) and after applying the logarithm we are left with values of ∞ and $-\infty$ - this can be solved by using pseudocounts (applying a Dirichlet distribution to each column).
[13]

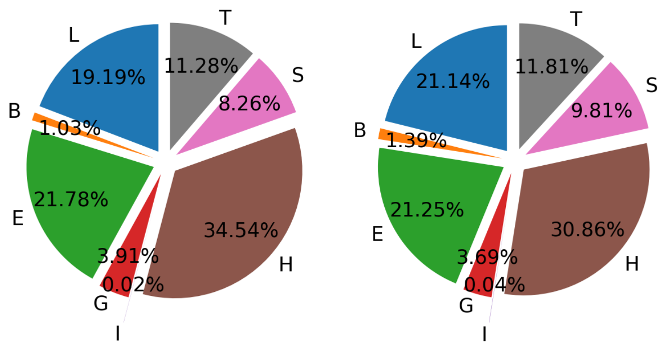
PSSM found in the CullPDB dataset has been generated using PSI-BLAST against UniRef90 database and the final results was transformed using the sigmoid function. One interesting point is that there are 21 entries for the amino-acids, 1 more than the known types - there is one extra encoding ”X” that denotes an unknown type.[3]

It is important to see how balanced the data is before moving on to training. In Figure 3.1(a) we can observe how heavily imbalanced the entire dataset is. The labels ’B’,’G’ appear in a low number compared to the others while the number of amino-acids labeled as ’I’ represents a really small fraction (212 labels).

3.3 Evaluation methods

Throughout the report, different models are evaluated and multiple types of performance metrics appear, so it is important to highlight how they are computed.

It seems a bit counter intuitive, but the main idea behind protein secondary structure prediction is not to label correctly the entire protein - for example, the test datset CB513 contains 513 proteins and the number of AAs is 84,765 (Figure 3.1(b)), therefore the accuracy found in Equation 3.3 uses the number of AAs and not the number of proteins



(a) CullPDB training data (b) CB513 test data

Figure 3.1: The distribution of the secondary structure labels. The training and test data have a similar distribution, with a low frequency of the π-helix(I)

in the dataset. That is, a model performs well if it manages to predict how the AAs are going to interact with each other and stabilise into a secondary state.

$$accuracy = \frac{\text{number of correct predictions}}{\text{total number of cases}} \tag{3.3}$$

Depending on the number of secondary structure classes each AA needs to be assigned we depict three types of accuracies: Q2,Q3,Q8

$$Q2 : AA \in \text{class } \{\text{Helix (H), non-Helix (G,I,B,E,T,S,L)}\} \tag{3.4}$$

$$Q3 : AA \in \text{class } \{\text{Helix (H,G,I), } \beta\text{-sheets (B,E), coils (T,S,L)}\} \tag{3.5}$$

$$Q8 : AA \in \text{class } \{\text{H, G, I, B, E, T, S, L}\} \tag{3.6}$$

Equation 3.3 is the default one used by the *Keras* library. Here we present in a detailed manner how the class scores returned by a model are labeled correct or incorrect.

As we mentioned in literature review, the last layer of a convolutional architecture is represented by a dense layer that has a number of neurons equal to the number of classes of the clasification problem. These nodes output the class scores followed by an activation function. Let the score of a input X_i be denoted with $s = f(X_i; W)$, where W represents the weights of the network. A *softmax function* can act as an activation function and takes these unnormalised log probabilities of the classes and normalizes them such that they add up to 1.(Equation 3.7)

$$P(Y = y_i \mid X = X_i) = \frac{e^{s_{y_i}}}{\sum_j e^{s_j}} \tag{3.7}$$

That is, for the Q8 classification, the final results represent a probability distribution over 8 classes, showing how 'confident' the network is. *Keras library* chooses the class with the highest probability as the final results. (Equation 3.8)

$$T(y_i) = \begin{cases} \text{correct}, & \text{if } argmax(y_i) = argmax(target) \\ \text{incorrect}, & \text{otherwise} \end{cases} \tag{3.8}$$

Where:

$target$: is the true class label in 1 hot encoding

Chapter 4

Implementation, experiments and results

4.1 Data prepossessing and input discussion

Thankfully, the work of Jian Zhou and Olga G. Troyanskaya (2014) on the CullPDB dataset has been made available to the public - this saved hours of work for filtering the AA chains and computational time for computing the PSSM values.

When it comes to proteins secondary structure, the neural networks have to label AA that have local and long-range dependencies. Many techniques have been published where the concept of a sliding window is used in order to capture local and intermediate interactions. Using CNN this can be implemented by applying a filter (window) over the input in the first convolutional layer - essentially, the protein is segmented in smaller parts. It is known that in order to improve the Q8 accuracy there is a need to introduce global information[3].Therefore, most of the recent approaches make use of deep CNN networks ([14], [15], [11], [5]) or a combination of local feature extraction using CNN and other deep learning techniques such as long short term memory networks([16], [17], [3], [12], [4],).

When using a convolutional layer to segment the original input, the window parameters are still subjective to change according to the gradients from the loss function. It can become unclear whether the performance comes from local or long-range interactions as further convolutions combine these results and can disregard parts of the AA chain while training. In this report we focus mainly on local and intermediate range interactions by reshaping the dataset such that each AA appears in the center of an odd length window. In work of Matt Spencer et al. [18] multiple sizes of windows are tested and

the peak accuracy is achieved at 19. Our final dataset contains windows of size 19 and the prediction label is coming from the AA in the center - for the terminal AA, further experiments have been made to decide the best type of padding and this will be discussed in a further section.
Finally, the following developed architectures all use input of size 19x21 (19 amino acids with 21 features each) - vanilla approach where in order to make a prediction we gather information from 9 adjacent positions from the left and from the right.

4.2 CNN depth exploration

Taking in consideration that the average length of an alpha helix is 5 [19] and that of β-strand (E) is 6 [20], our input can capture this information. The next step is to discover the best arhitecture, in particular at what the depth(number of convolutional layers) we obtain the best accuracy.

To keep the computational time low, we have decided to use the smallest filter size that can capture some context (3), and a number of 64 filters per layer - the stride is set to one, meaning that a filter moves by 1 position and padding (zeros) is used to preserve the dimensionality of the input. The loss function used is a combination of the *cross-entropy loss* that uses the softmax scores and an additional factor which is the regularization parameter.

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right) \tag{4.1}$$

$$R(W) = \sum_k \sum_l W_{k,l}^2 \tag{4.2}$$

$$Loss = \frac{1}{N} \sum_i L_i + \lambda R(W) \tag{4.3}$$

(4.1) is the *cross entropy loss* between the true distribution(p) and our predicted distribution(q) which can be written as $H(p,q) = -\sum_x p(x) \log q(x)$ - what this means is that this part of the loss function "forces" the network to distribute the probability mass such that it matches the 1 hot target. (4.2) is called the L2 norm and is added such that the model is kept as simple as possible by penalising large weights.

In the following experiments, we use a lambda value of $\lambda = 10^{-4}$. For optimising the weights, we use TensorFlow's ADAM optimizer [21] (default initialisation parameters) with a learning rate value of $LR = 5*10^{-4}$. The networks that we train all have different number of convolutional layers that are followed by Batch Normalisation [22] layers such that we overcome the problem of gradient vanishing; at the end of the networks we have 2 Dense layers of size 16 and 8 (Softmax layer).

In Figure 4.1 we can observe the performance of networks with different depth where we use the same number of filters per layer - note, the number of parameters is different: $\approx 24k, \approx 50k, \approx 75k, \approx 100k$ and $\approx 125k$. There is a clear difference in training and validation accuracy as the models with more depth perform better despite over-fitting the data. CB513 accuracy is close to other publications and it follows the same trend - it is 4% lower than the validation accuracy.

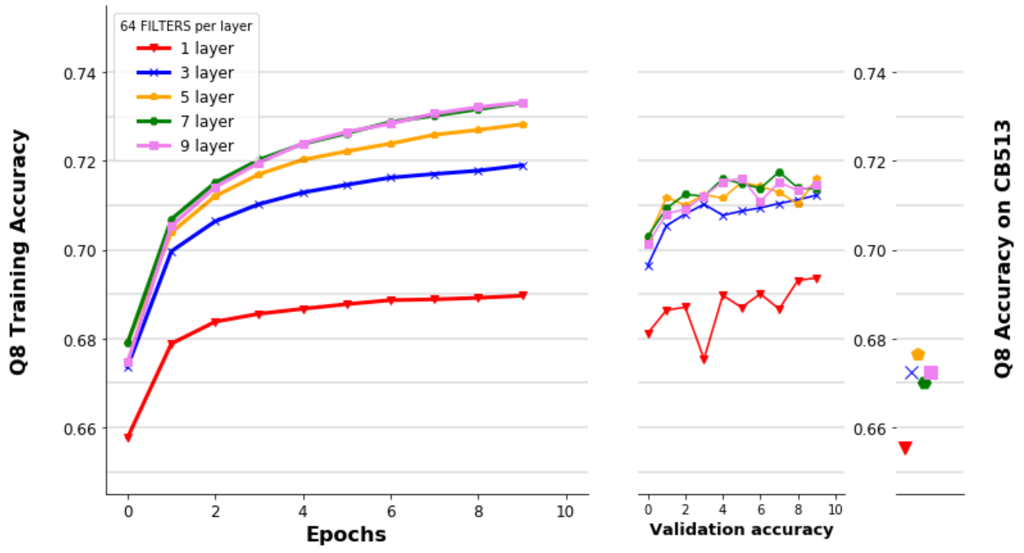


Figure 4.1: Q8 accuracy of models with different numbers of layers that use the same filter size. The model with only one layer performs relatively poor, while the networks with more layers start to overfit the data. The model with 5 layers has the highest accuracy on CB513 and it seems as the best trade-off between depth and performance.

We can observe that increasing the number of convolutional layers above 5 does not bring any noticeable contribution to the overall performance.

To see if the performance comes from the depth of the network and not from the increased number of parameters, we have retrained other architectures each with $\approx 125k$ parameters such that the only difference is the depth.

In Figure 4.2 we can see some improvement for the network with one layer, but the overall learning trend seems to stagnate. The other networks seem to be quite similar, with the same over-fitting problem as before. We believe that choosing the right regularization factor for each network would have made the performance difference clearer but using the information provided by the training accuracy, the network with 5 layers has the most descriptive power. The accuracy on the test set CB513 could suggest that a network with 3 layers has the same performance - taking into consideration that we intend to further improve the accuracy using other methods, we choose the network with 5 layers as it offers the best trade-off between performance and training time.

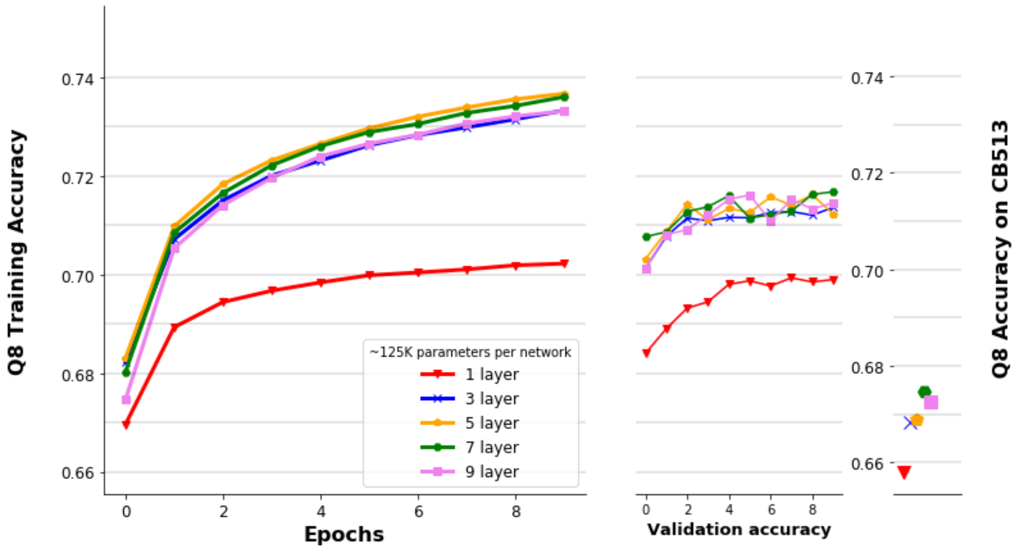


Figure 4.2: Q8 accuracy of models with the same numbers of parameters. Making the number of parameters equal across the networks makes it harder to differentiate between the deeper models. The model with 5 layers has the most descriptive power (training accuracy).

We further study the effect of the number of filters on the 5 layer network to see if it is possible to shrink the number of parameters to keep the model as simple as possible. In Figure 4.3, the networks with a decreased number of filters tend to generalise better on unseen data (validation) but still perform worse than the one with 64 filters that overfits. We can observe that the 5 layer architecture performs the best on the CB513 and it has enough descriptive power which means that there is no need to increase the number of parameters.

4.3 K-fold accuracy

In this section we perform 10 fold-cross-validation on the training data to see the variation of the accuracy on the test set - in particular, we divide the entire dataset in 10 and use a subset of 9 for training and the one left out for testing. Two types of input are used: windows of size 19 in PSSM and 1 Hot encoding. The following experiments are run with the same parameters on the the architecture with depth 5 from the previous section. An additional change to the dataset is made to reshape the target labels from Q8 to Q3 and Q2 in order to provide the following accuracies.

The accuracy variation highlighted Figure 4.4 is quite small across all problem types with a small number of outliers - this suggests that the network manages to perform well in general and not because of a ”lucky” initialisation. Another important idea

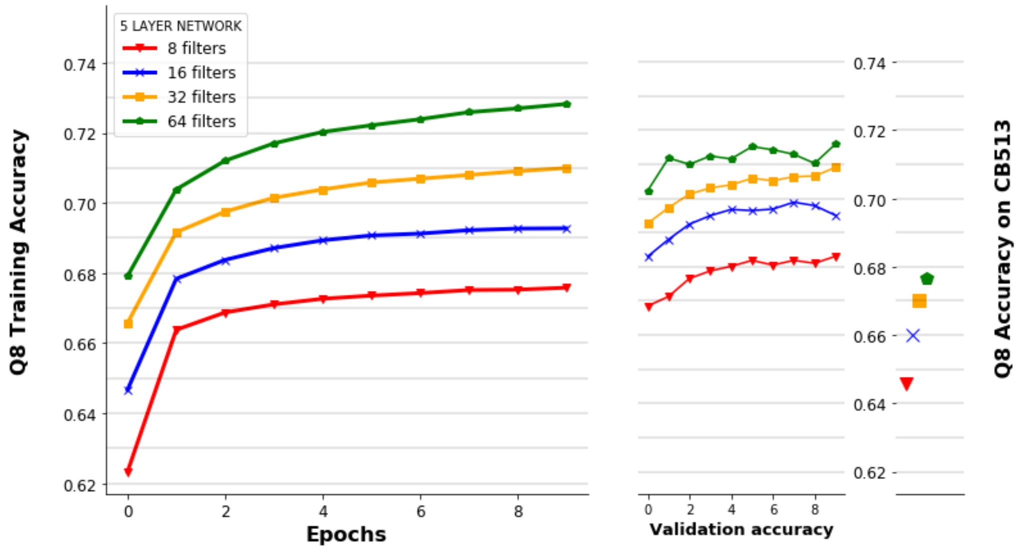
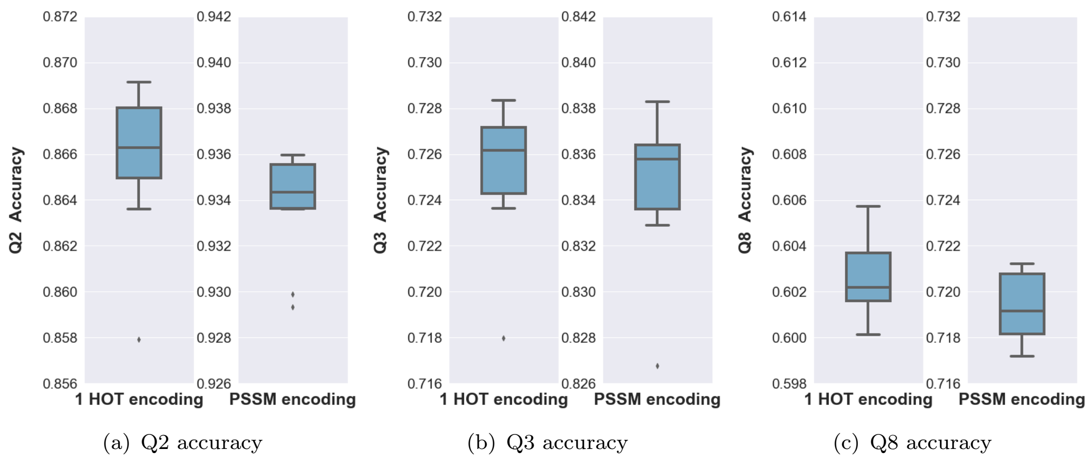


Figure 4.3: Q8 accuracy of the 5 layer network with various number of filters. Decreasing the number of filters comes with a decrease in the validation and test accuracy. The network with 64 filters starts to overfit the data, indicating that there are enough trainable parameters.



(a) Q2 accuracy (b) Q3 accuracy (c) Q8 accuracy

Figure 4.4: Accuracy variation on 10 fold cross-validation using a window of size 19 and predicting the label of the AA in the middle. Using the PSSM values as input, the models perform up to 10% better than using 1 Hot encoding. The variation in accuracy is also bigger for the second type of input, showing that the networks tend to learn different features for each random initialisation.

suggested by these results is that using PSSM values instead of 1 hot encoding format, we can boost the accuracy up to 10% - for the Q2 class this is only 5% as it is a simple classification problem where the label can be H or non-H and the PSSM values do not bring much contribution. We can observe that the Q8 classification is much harder than Q3(10% accuracy difference) - we believe that this is due to the network having problems predicting those AA that have long-interactions. That is, for each different run with random initialisation, the model has a slightly different way of understanding the Q8 classification problem.

To validate the findings of Guillermo Romero Moreno et al. (2019), where it is stated that the right side of the window holds more information, we have reshaped the dataset to windows of size 9 where the target class label belongs to the AA on the right or the left side of the window.(Figure 4.5)

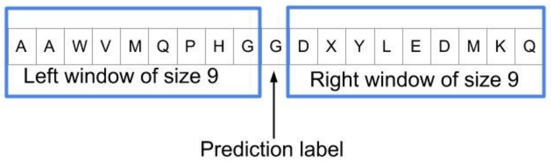


Figure 4.5: Example of a window of size 19 split into two windows of size 9. The network gets as input the right or the left window and it has to predict the label of the AA in the center

The idea that α-helices hold a large asymmetry can be seen in Figure 4.6 (a). It is indeed easier to predict the label using the information from the right side - as it is mentioned in the paper, this asymmetry can point to a strong dependency on the previous (posterior) AAs. In other words, the information found at the right side is more valuable because we can "infer" knowledge about previous amino-acids. What is more interesting is that we can see the same dependency when it comes to Q3 and Q8 classification.

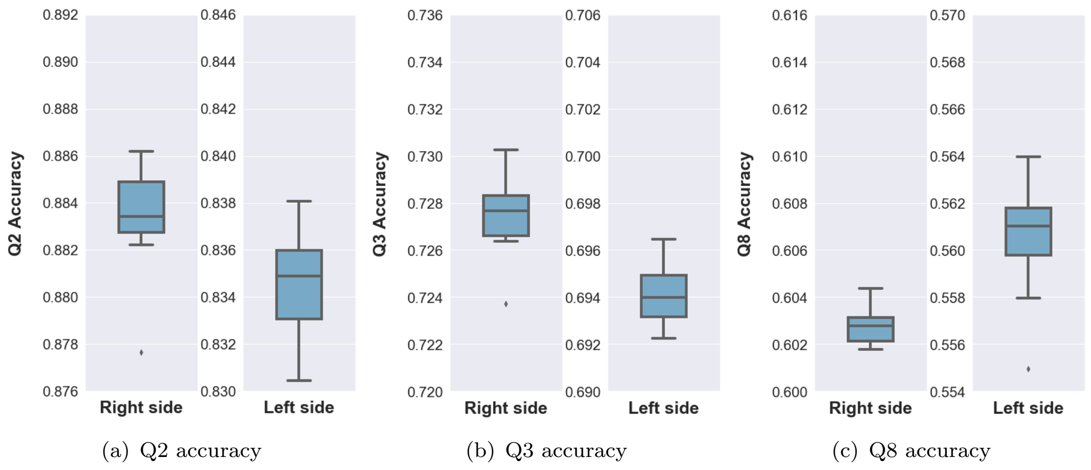


Figure 4.6: Accuracy variation on 10 fold cross-validation with a window of size 9. The right side holds more information and this is reflected in a higher accuracy. The variation in accuracy when using the left window is considerable higher - this is due to the low content of information that makes the networks to have a lower confidence over particular labels.

One advantage over the previous reported results is that here we are limiting the model to learn local interactions and we can be confident that the difference in accuracy does not come from information found further across the AA chain.

4.4 Improving the architecture

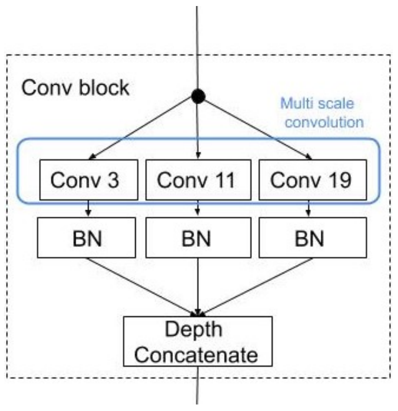
Motivated by the previous findings and the work of Akosua Busia and Navdeep Jaitly [11] that achieved the state-of-the-art accuracy on CB513 benchmark using the CullPDB trainig set, we have decided to further investigate possible improvements to our network. The CNN constructed by Akousa Busia et. al, uses some recent findings in the field of Deep Learning especially from Computer Vision (image classification) to boost the performance - towards the end of the paper, it is mentioned that the vast majority of the information used for protein secondary structure prediction comes from local interactions (different from what was believed in the past). In this section, we use some of these new techniques but what is different from the other published approaches is that we are trying to use the depth of the network only for local interactions that appear in the span of 19 AA. We are not going to go in depth of every new addition to the network as this goes beyond the purpose of this report, but a general overview will be provided.

The Inception network that is designed for image classification [23], introduces multi-scale convolutions that apply different filters on the input that compute different features at various scales. In Figure 4.7(a), our multiscale convolution uses three types of filters that slide on the same input. Here we decided to keep the previous window of 3 as it performed relatively well, and we introduce two new filters of size 11 and 19. The filter of size 19 might seem odd, with a size way larger than a typical kernel - considering that our input is only a fraction of the AA chain, the parameters for a filter of size 19 is still manageable (19x21 x 19 instead of 700x21 x 19). The intuition of using a filter size the same as the input is to "build" global context (entire AA chain) from a small window. Thinking about how a typical deep CNN manages to use information from the entire sequence, here we are extending the information of the window by "diluting" the ends with staked convolutions that use padding. In other words, we are trying to infer knowledge about the surrounding AAs that the network cannot "see" from a small context input.

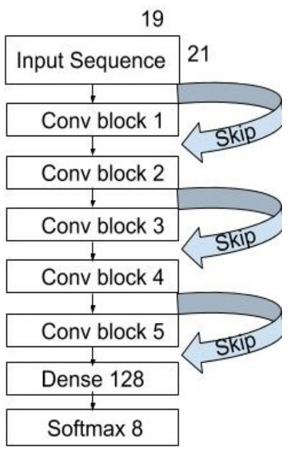
Huang, Liu, & Weinberger's DenseNet (2016 [24]) makes use of depth concatenation to improve the information flow - each layer gets as input the depth concatenation of the outputs of all the previous layers. In Figure 4.7(b), the skip connections represent this concatenation, meaning that the network is not limited to the results suggested by the last layer and that it can use all the information available (for example, Conv block 2 gets as input the results of Conv block 1 plus the Input sequence).

4.4.1 Accuracy on CB513

We train the previous mentioned network with the same parameters as before but instead of using L2 weight regularization to stop the network from over-fitting, we are



(a) Convolutional block



(b) Convolutional architecture

Figure 4.7: (a) A convolutional block that contains multi-scale convolutions with depth concatenation of the results. (b) Our final deep convolutional architecture that contains 5 conv-blocks with skip connections between each one and 1 fully conected layer along with the softmax output layer.

introducing a dropout layer after each convolution. Dropout is simple and efficient regularization technique introduced by Srivastava et al., (2014 [25]) that is implemented by deciding whether to keep a neuron active with a *probability* p(hyperparameter) or setting it to 0 otherwise. Here a dropout probability of 0.6 is used in order to keep the network with $\approx$ 6 milion parameters overfit the dataset containing $\approx$ 1 milion windows.

In section 4.1 we mentioned that for the AAs that are close to the ends of the AA chain, we need to add some kind of padding such that we are able to center them in a window of size 19 (Figure 4.8). In this section, we pad each protein primary structure in CullPDP and CB513 using different techniques - for each dataset version, we train the previous architecture for 45 epochs and we use the validation set for early stopping.

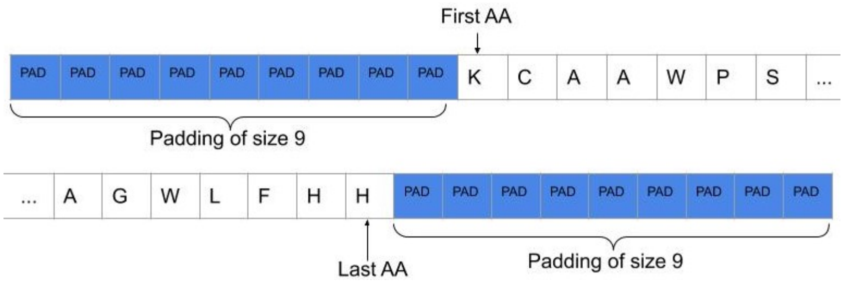

First AA
PAD PAD PAD PAD PAD PAD PAD PAD PAD K C A A W P S ...
Padding of size 9
... A G W L F H H PAD PAD PAD PAD PAD PAD PAD PAD PAD
Last AA
Padding of size 9

Figure 4.8: Example of a AA chain that contains padding in order to accommodate the terminal AAs - a window of size 19 will be able to slide outside the original formation to center each AA

Two of the most common padding methods are to repeat the last AA or to use 0, meaning that there is no amino-acid present at that position. Using random AAs from the chain to pad the sequence or to consider the chain continuous like in a loop (the end of the sequence is also the beginning) is not beneficial as the Q8 test accuracy is much

lower than padding with 0.
In Table 4.1, we can observe that again, the right side is more important - padding with 0 the right side and repeating the left AA has the highest accuracy. The idea behind this finding is that repeating the end AAs should perform better than padding with 0. The reason this is not reflected in the results is because altering the right side (holding the most information) brings the accuracy down. Therefore, the experiments presented in the latter sections all use this dataset configuration.

Random	Repeating both sides	0	Repeating only left-side	Repeating only right-side	Loop
69.001%	69.619%	69.736%	**69.796%**	69.180%	69.152%

Table 4.1: Q8 accuracy on CB513 using different padding sequences for the AA chains. Padding with 0 both sides performs better than repeating the AA. A combination of padding with 0 on the right side and repeating the AA on the left side is showing the best performance

4.5 Model Assembly

While other modifications to the hyperparameters by fine tuning can boost the performance on the test dataset, we consider that this would not bring much insight to the problem. In this section, we continue to use the same setup us before but this time we merge the validation dataset with the training data. With this technique and using early stopping validation on the CB513 we get the full training potential of the CullPDB dataset.

We train 20 models using our deep CNN architecture with random initialisation of the learnable parameters. In Figure 4.9 we can observe an average increase in accuracy due to the use of the entire training data. The best performing model trained only on local context interactions manages to achieve an impressive accuracy of **70.06 %** on the CB513 benchmark dataset.

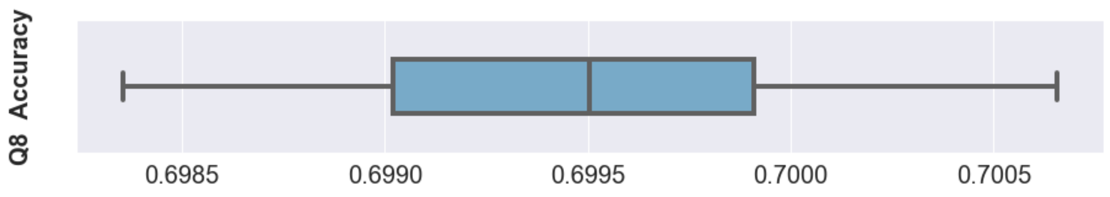


Figure 4.9: Q8 Accuracy variation of 20 models on CB513. Best performing model achieves 0.7006% and on average we get 0.699% ± 0.001

One reliable approach to improve the accuracy of protein secondary structure prediction is to use an assembly of networks and average their predictions when we make the evaluation. In general, the accuracy should increase monotonically as we add more models to the assembly, but with diminishing returns [26]. We are aware that the work of R. Caruana et al. (2004 [27]) on ensemble selection from libraries of models could be beneficial in our case. Taking into consideration that the variation of our models is only due to random initialisation, we believe that a simple selection algorithm is sufficient for the purpose of this work.

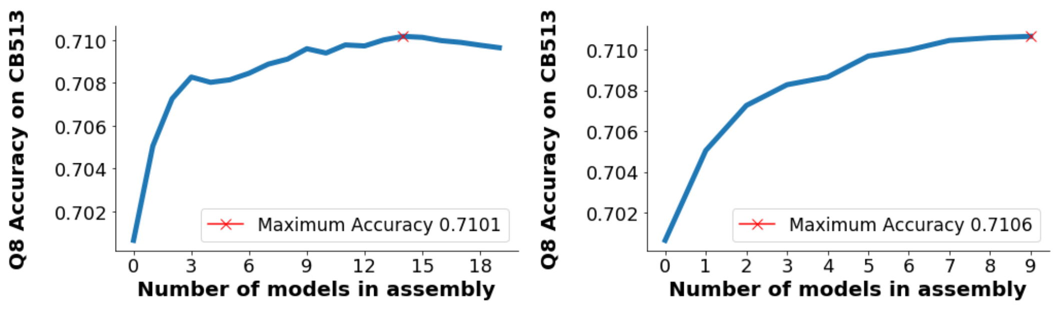
Various methods to combine the results of the models can be used, but we find that a simply average over the outputs performs the best. (4.4)

$$assembly_{pred}(x_i) = \frac{1}{M}\sum_{j=1}^{M}\mathcal{M}_j(x_i) \tag{4.4}$$

Where:

$\mathcal{M}_j(x_i)$: is the prediction made by model j

The 20 trained models are sorted according to the Q8 accuracy and we add them to the assembly by starting with the best one. The performance of the assembly network can be seen in Figure 4.10. As we mentioned above, a major improvement comes from using only two networks and then the accuracy increases gradually. Two methods of model selection are used, and the final assembly consists of 10 models that achieve an Q8 accuracy on the CB513 bechnmark of **71.06%**.

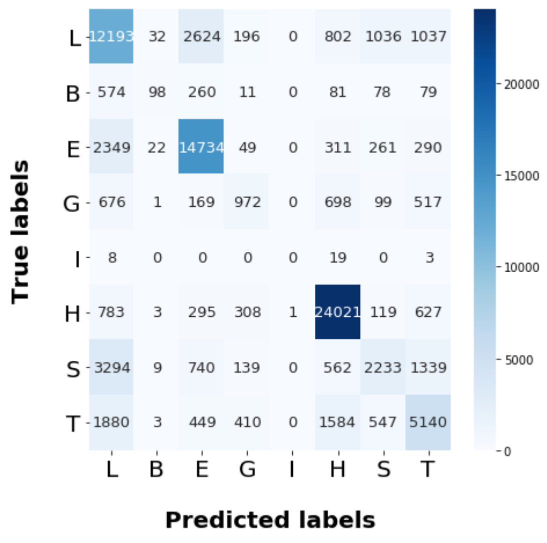


(a) Unsupervised model assembly (b) Supervised model assembly

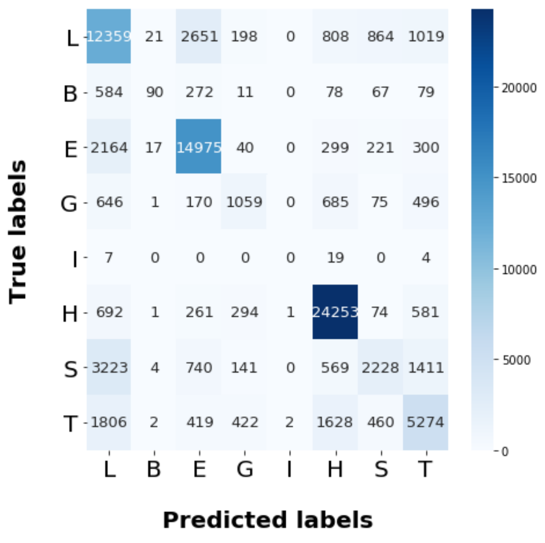
Figure 4.10: Model assembly performance on the test set CB513. In (a) 20 models are added to the assembly one by one in a descending order of performance - it reaches a maximum at around 14 models. In (b) the models are added only if they bring a contribution when making predictions - 10 models perform better than 14 when we supervise the accuracy

Using the ensemble of networks increases the accuracy by 1% compared to the best single performing model. While this is beneficial, the improvement is a bit lower than we expected. We believe that the models tend to agree on most of the local interactions and the boost in performance comes in general from the long-range interactions which are hard to predict by only using a local context window.

To observe what labels are misclassified when making predictions, in Figure 4.11 we have plotted confusion matrixes for our single model and for the average of our assembly of networks. The first thing we notice is the low frequency of labels 'I' and 'B' - the network has a high accuracy even if it ignores the π-helix (I). As we mentioned at the beginning of the report, we can see that there is low confusion for the labels α-helix(H) and β-strand(E) as these can be captured by our low context window. In general, the confusion appears within Q3 classification, showing once again how difficult is the Q8 prediction, especially for the coil like structures (T,S,L). Contrary to our beliefs, it seems that the improvement that the ensemble brings is for AAs that have local interactions - for the β-bridge (B) that relies on long-range interactions there is a decrease in accuracy.



(a) Best model-confusion matrix



(b) Assembly of models-confusion matrix

Figure 4.11: Confusion matrix on the Q8 classification problem. The assembly of models improves the prediction of all labels apart from the β-sheet (B)

4.6 Comparison with state-of-the-art methods

To better highlight the performance of our model, we compare our results with previously published work that uses the same training set and provides the accuracy on CB513. Here, we mainly focus on gathering information from work that focused on developing CNN architectures along with other models that have high performance in general. In Table 4.2 we observe that our model manages to outperform all the architectures discussed in the literature review and it gets close to the results of the current state-of-the art model developed by Busia et al. One of the most interesing point is that we manage to achieve this accuracy only by using local interactions that appear inside a winodow of size 19. These findings are similar to the work of Busia et al. that compared a simple fully connected network with the network developed by Zhen Li et al. (DCRNN) - there is a low information content that a network can learn from the entire input sequence.

We strongly believe that our CNN developed from scratch performs well due to the addition of new techniques borrowed from Computer Vision such as Skip Connections and Multi Scale Convolution. As we mentioned before, the network is not fine tuned and there is a clear overfitting problem - improving these aspects could push the accuracy even further. Our CNN performs better than the Deep Multi-Scale CNN model, showing that forcing the network to use only information from local interactions acts as an improvement (better control over the information flow). The gap between the state-of-the art and our architecture comes from the use of next step conditioning which is a technique that feeds as input the predicted label of the previous AA.

Methods	**CB513 Q8(%)**	**Authors**
Single model		
GSN	66.4	(Jian Zhou et al. [3])
DeepCNF	68.3	(Sheng Wang et al. [4])
DCRNN	69.4	(Zhen Li et al. [5])
biRNN-CRF	69.4	(Rosenberg et al. [12])
Deep Multi-Scale CNN	70.0	(Busia et al. [11])
Our CNN	*70.06*	
Conditioned Deep Multi-Scale CNN	**70.3**	(Busia et al. [11])
Ensemble		
DCRNN	69.7	
biRNN-CRF	70.9	
Deep Multi-Scale CNN	71.0	
Our CNN	*71.06*	
Conditioned Deep Multi-Scale CNN	**71.4**	

Table 4.2: Comparison of Q8 accuracy on CB513 between our CNN and current published models. Our model tends to perform better than most of the previous work even if we are only using local AA interactions as input.

Two metrics that can offer a better insight into this comparison are precision and recall. These measurements allow us to observe the individual scores for each label and compare them with previous work.

$$precision = \frac{\text{True positives}}{\text{True positives} + \text{False positives}} \tag{4.5}$$

$$recall = \frac{\text{True positives}}{\text{True positives} + \text{False negatives}} \tag{4.6}$$

Overall, our single best performing model obtains better precision values compared to Deep Multi-Scale CNN on three labels (G,H,E), showing that there are benefits coming from focusing on local interactions. What is more interesting is that the precision for the

α-helix(H) is even greater than the value from the Conditioned Deep Multi-Scale CNN - this also shows that the local features are disregarded by using deeper networks. Our model also has higher recall values for the G,T,B labels, but in general, the state-of-the art method presents better scores because it has a higher accuracy. The recall value of the β-sheet(B) is higher in our case because the network is not trained to predict this label, therefore, there are a few cases of False negatives.

Q8 label	**Precision**				**Recall**			
	biRNN-CRF	**DMS CNN**	**DMS CNN***	**Our CNN**	**biRNN-CRF**	**DMS CNN**	**DMS CNN**	**Our CNN**
L	N/A	**0.575**	0.565	0.560	N/A	0.651	**0.690**	0.680
B	N/A	0.656	**0.676**	0.583	N/A	0.050	0.041	**0.083**
E	N/A	0.747	**0.767**	0.765	N/A	**0.836**	0.821	0.818
G	N/A	0.454	**0.487**	0.466	N/A	0.309	0.285	**0.310**
I								
H	N/A	0.840	0.841	**0.856**	N/A	**0.936**	0.932	0.918
S	N/A	0.529	**0.548**	0.511	N/A	0.248	0.240	**0.269**
T	N/A	0.571	**0.577**	0.569	N/A	0.524	**0.524**	0.513

Table 4.3: A comparison of precision and recall values for each label on CB513 test set using a **single model**. For biRNN-CRF, no values have been reported and the results for the π-helices do not appear as they do not have a high frequency in CB513. DMS CNN* stands for Conditioned Deep Multi-Scale CNN

When it comes to an ensembly of networks, our CNN shows the same kind of pattern of values compared to the other models. In Tabel Figure 4.4 we can observe that those models that focus more on longer interactions (biRNN-CRF) have higher scores as there is more variation inside the assembly.

Q8 label	**Precision**				**Recall**			
	biRNN-CRF	**DMS CNN**	**DMS CNN***	**Our CNN**	**biRNN-CRF**	**DMS CNN**	**DMS CNN**	**Our CNN**
L	**0.606**	0.576	0.570	0.575	0.672	0.682	**0.707**	0.690
B	0.614	0.696	**0.786**	0.662	**0.099**	0.047	0.047	0.076
E	**0.803**	0.765	0.776	0.768	**0.853**	0.839	0.837	0.831
G	**0.530**	0.499	0.528	0.489	**0.343**	0.308	0.290	0.338
I								
H	**0.878**	0.841	0.846	0.856	**0.936**	**0.936**	**0.936**	0.927
S	0.537	0.587	**0.621**	0.559	**0.288**	0.244	0.237	0.268
T	0.589	0.585	**0.591**	0.576	**0.604**	0.537	0.542	0.527

Table 4.4: A comparison of precision and recall values for each label on CB513 test set using an ensembly of networks

Bibliography

[1] FC Bernstein, TF Koetzle, GJB Williams, EF Meyer Jr, MD Brice, and JR Rodgers. Kennard 0, shimanouchi t, tasumi m. 1977. the protein data bank: A computer-based archival file for macromolecular structures. *J. Mol. Biol*, 112:535–542, 1977.

[2] Wolfgang Kabsch and Christian Sander. Dictionary of protein secondary structure: pattern recognition of hydrogen-bonded and geometrical features. *Biopolymers: Original Research on Biomolecules*, 22(12):2577–2637, 1983.

[3] Jian Zhou and Olga G Troyanskaya. Deep supervised and convolutional generative stochastic network for protein secondary structure prediction. *arXiv preprint arXiv:1403.1347*, 2014.

[4] Sheng Wang, Jian Peng, Jianzhu Ma, and Jinbo Xu. Protein secondary structure prediction using deep convolutional neural fields. *Scientific reports*, 6:18962, 2016.

[5] Zhen Li and Yizhou Yu. Protein secondary structure prediction using cascaded convolutional and recurrent neural networks. *arXiv preprint arXiv:1604.07176*, 2016.

[6] Guillermo Romero Moreno, Mahesan Niranjan, and Adam Prugel-Bennett. Saliency map on cnns for protein secondary structure prediction. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1249–1253. IEEE, 2019.

[7] François Chollet et al. Keras. `https://keras.io`, 2015.

[8] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[9] Helen Berman, Kim Henrick, and Haruki Nakamura. Announcing the worldwide protein data bank. *Nature Structural & Molecular Biology*, 10(12):980–980, 2003.

[10] Guoli Wang and Roland L Dunbrack Jr. Pisces: a protein sequence culling server. *Bioinformatics*, 19(12):1589–1591, 2003.

[11] Akosua Busia and Navdeep Jaitly. Next-step conditioned deep convolutional neural networks improve protein secondary structure prediction. *arXiv preprint arXiv:1702.03865*, 2017.

[12] Alexander Rosenberg Johansen, Casper Kaae Sønderby, Søren Kaae Sønderby, and Ole Winther. Deep recurrent conditional random field network for protein secondary prediction. In *Proceedings of the 8th ACM international conference on bioinformatics, computational biology, and health informatics*, pages 73–78, 2017.

[13] Position weight matrix. `https://en.wikipedia.org/wiki/Position_weight_matrix`, 2019. [Online; accessed 1-November-2019].

[14] Jiyun Zhou, Hongpeng Wang, Zhishan Zhao, Ruifeng Xu, and Qin Lu. Cnnh_pss: protein 8-class secondary structure prediction by convolutional neural network with highway. *BMC bioinformatics*, 19(4):60, 2018.

[15] Chao Fang, Yi Shang, and Dong Xu. Mufold-ss: New deep inception-inside-inception networks for protein secondary structure prediction. *Proteins: Structure, Function, and Bioinformatics*, 86(5):592–598, 2018.

[16] Rhys Heffernan, Yuedong Yang, Kuldip Paliwal, and Yaoqi Zhou. Capturing non-local interactions by long short-term memory bidirectional recurrent neural networks for improving prediction of protein secondary structure, backbone angles, contact numbers and solvent accessibility. *Bioinformatics*, 33(18):2842–2849, 2017.

[17] Iddo Drori, Isht Dwivedi, Pranav Shrestha, Jeffrey Wan, Yueqi Wang, Yunchu He, Anthony Mazza, Hugh Krogh-Freeman, Dimitri Leggas, Kendal Sandridge, et al. High quality prediction of protein q8 secondary structure by diverse neural network architectures. *arXiv preprint arXiv:1811.07143*, 2018.

[18] Matt Spencer, Jesse Eickholt, and Jianlin Cheng. A deep learning network approach to ab initio protein secondary structure prediction. *IEEE/ACM transactions on computational biology and bioinformatics*, 12(1):103–112, 2014.

[19] Claus A Andersen, Henrik Bohr, and Søren Brunak. Protein secondary structure: category assignment and predictability. *FEBS letters*, 507(1):6–10, 2001.

[20] Simon Penel, R Gwilym Morrison, Paul D Dobson, Russell J Mortishire-Smith, and Andrew J Doig. Length preferences and periodicity in β-strands. antiparallel edge β-sheets are more likely to finish in non-hydrogen bonded rings. *Protein engineering*, 16(12):957–961, 2003.

[21] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[22] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[23] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI conference on artificial intelligence*, 2017.

[24] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[25] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[26] CS231n: Convolutional Neural Networks for Visual Recognition. `http://cs231n.stanford.edu/`, 2017. [Online; accessed 1-January-2020].

[27] Rich Caruana, Alexandru Niculescu-Mizil, Geoff Crew, and Alex Ksikes. Ensemble selection from libraries of models. In *Proceedings of the twenty-first international conference on Machine learning*, page 18, 2004.