4-23-2021 10:30 AM

Visualization and Interpretation of Protein Interactions

Dipanjan Chatterjee, *The University of Western Ontario*

Supervisor: Ilie, Lucian, *The University of Western Ontario*

A thesis submitted in partial fulfillment of the requirements for the Master of Science degree in Computer Science

Abstract

Visualization and interpretation of deep learning models' prediction is a very important area of research in machine learning nowadays. Researchers are not only focused on generating a model with good performance, but also they want to trust the model. Our aim in this thesis is to adapt existing interpretation methods to a protein-protein binding site prediction problem to visualize and understand the model's prediction and learning pattern.

We present three deep learning-based interpretation methods: sensitivity analysis, saliency map and integrated gradients to analyze the amino acid residues which create positive and negative relevance to the deep learning models' prediction. As our applications use sliding window protocol, we are particularly interested in learning patterns of the model and identify the important positions. Also, we want to focus on the feature importance through Local Interpretable Model-Agnostic Explanations (LIME).

With our experiment we observe that, in spite of using various features, position specific scoring matrices (PSSM) is identified as the most important feature that helps the model to identify positive classes. As PSSM's importance is proven historically, this finding gives us a strong biological significance as well.

Keywords: Visualization and Interpretation, Deep Learning, Black-box model, Proteins, Bioinformatics

Lay Summary

Deep learning models are often called as *black box* in spite of their tremendous success with respect to accuracy. As the accuracy of the model increases from a decision tree to neural networks, the complexity of the model also increases. The higher the complexity the lesser is the degree of explainability.

When we use deep learning models in sensitive areas like bioinformatics and drug production, it becomes essential to understand their performance well. This motivates us to dig into this area of visualization and interpretation for proteins. We have adapted few existing interpretation methods to our PPI problem to understand how the model is learning while it predicts. Also, we focus on the important features which play key role in model's decision. In this dissertation, we look into the background study and related works, then we discuss the methods we have used and finally we present our findings from the visualizations.

Acknowlegements

Firstly, I would like to acknowledge my supervisor Dr. Lucian Ilie, for choosing me as his masters student. When I started my M.Sc. I had almost zero knowledge about deep learning and bioinformatics. I would like to thank Dr. Ilie for showing me the path and guide me to acquire knowledge in both the areas. My masters thesis would not be possible without his lectures, countless number of video discussions and more importantly the mental support in the time of COVID-19 pandemic.

I would like to express my humble gratitude to my parents Mr. Debashis Chatterjee and Mrs. Sikha Chatterjee, for their unconditional love and support throughout my career. When I started my masters, I had to quit my well-settled IT job, which was not an easy decision to take. I thank my Dad for supporting me to take this hard decision and chase my dreams in spite of limited funds. Also, I would like to thank my fiancee Ms. Anuradha Saha for her support throughout this time of pandemic, without that it was impossible to stay alone in Canada.

I acknowledge all my lab mates : Sourajit Basak, Arnab Mallik, Sabyasachi Patajoshi, Yiwei Li for their guidance and help throughout my masters degree. Also , I would like to mention my classmate Vishwa Sai Kodiyala for sharing his valuable knowledge in this domain.

Last but not the least, I thank the Department of Computer Science at Western University for providing world class facilities and giving me the funding for my graduate studies.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The use of deep learning and machine learning models to solve various problems concerning DNA and proteins increased rapidly in the last few years. While these models generate very good results, the reason behind their performance is still poorly understood. Hence explainability becomes an important criterion for researchers as we all want to see through the black-box, what these models are actually learning. This problem in general for natural language processing (NLP) and computer vision applications is relatively easier than a biological problem as humans can understand by seeing an image or reading a text, which pixels in that image needs to be important for the classification task or what parts of the text make it relevant. But for a biological problem it is not the same as it contains large, imbalanced, numerical data which are very hard to interpret even by the biologists sometimes. Also, bioinformatics applications are very sensitive area of research because it is directly related with vaccine development, drug productions, patient's treatment etc. where we cannot leave any room for mistakes. For medical diagnosis, a doctor can not just treat a patient based on a model's decision without understanding how that decision was obtained. That is why it is so important to trust the model's prediction than just to rely on their performance.

In the initial days, for methods like classification rules, decision trees, regression algorithms, it was easier to understand the reason behind the model's decision. But as the complexity of the model increases the degree of explainability decreases. Models like XGBoost, Deep neural networks consist of highly non-linear mathematical functions with multiple layers which make this interpretation even more difficult.

Thus explainability and interpretability is a very active area of research for biological applications. Also, another important aspect is to visualize the explanations to understand more about the model's decision. Our focus will be on both of these aspects. We will discuss the interpretation methods and visualization techniques we have used for DELPHI [22] and DL-Pred [36], two deep learning based programs for protein-protein binding site prediction. Then we will present some boxplots generated from the data of each explainable method for several proteins to understand the impact by position inside the sliding window. We will also discuss about the most influential feature for DELPHI.

1.1 Motivation

We have found few implementations for DNA based applications ([20], [37]) where researchers have used explainability techniques to discover interesting facts about the model's behaviour. They compared the model's explanation with some known patterns and found out the model is indeed focusing on the part in the sequence containing the patterns while making the prediction. But for proteins to the best of our knowledge no work has been done yet. This is because proteins are much more complex with respect to their structures and alphabet size compare to DNA.

A DNA contains only 4 bases (A,C,G,T) whereas proteins have 20 amino acids. Proteins have a complicated 3D structure whereas DNA has a simple helical structure. Hence understanding model's decision for protein related problems becomes much more challenging. This motivates us to dig into visualization and interpretation of the proteins.

1.2 Problem Statement

We consider the problem of predicting protein-protein interaction binding sites using deep learning models and propose using existing methods for the visualization and interpretation of these models. The goal is to understand the predictions, as well as what causes the actual interactions.

1.3 Thesis Contribution

The major contribution of this thesis is that it is the first approach to visualize and interpret the deep learning model's prediction for protein problems. We consider two best models for protein-protein binding site prediction, DELPHI [22] and DLPred [36]. The models use a sliding window and predict whether the amino acid residue in the center of the window is interacting or not. We present visualizations for both models and observe that some residues have a greater importance when predicting nearby protein-protein interaction binding sites. Also, we present boxplots with the data obtained from each visualization methods to observe the impact by position in the sliding window. The importance of each residue when predicting nearby binding sites seems to be well correlated with the distance between them. Also, we find the position-specific scoring matrix (PSSM) as the most important feature for DELPHI, which is helping the model's prediction.

1.4 Thesis Outline

In **Chapter 2**, we give basic biological information concerning DNA, RNA and Proteins. We also explain the concept of protein-protein interaction binding site prediction. We talk about deep learning and how it replaced the algorithmic based approach in biological applications in the last few years.

Then we focus on the opacity of deep learning and the works done to alleviate this problem in general machine learning applications and some bioinformatics applications.

In **Chapter 3** we discuss the methods: sensitivity analysis, saliency map, integrated gradients and LIME, which we have used for our experiment. Also we mention the limitations of the algorithms and how we adapt them for our application.

In **Chapter 4** we present our experimental results generated from the methods. We provide the visualizations we obtain for protein sequences and also we give the LIME-identified important features in a bar graph.

In **Chapter 5** we summarize our work and also present some possible future research directions to enhance this domain of research.

Chapter 2

Background

All lives are dependent on three critical molecules: DNA, RNA and protein. DNA holds the information of how cells work. The primary role of RNA is to convert the information stored in DNA into proteins. On the other hand, proteins are more complex, large molecules that are responsible to form the body's major components like hair, skin, etc.

2.1 DNA

DNA stands for deoxyribonucleic acid. DNA is made of four nucleotides: adenine(A), thymine(T), guanine(G), and cytosine(C). DNA has double helical structure as discovered by Watson and Crick. The two strands of DNA are held together by hydrogen bonds and A always pairs with T and G always pairs with C.

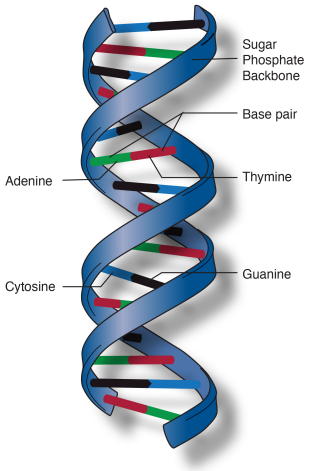

Figure 2.1: Double helix structure of DNA [5]

2.2 Proteins

Proteins are made of amino acids. There are twenty different amino acids present in proteins: Phenylalanine(F), Tryptophan(W), Isoleucine(I), Leucine(L), Methionine(M), Valine(V), Cysteine(C), Alanine(A), Tyrosine(Y), Histidine(H), Proline(P), Glycine(G), Threonine(T), Serine(S), Lysine(K), Arginine(R), Glutamic acid(E), Aspartic acid(D), Glutamine(Q) and Asparagine(N). Within the protein molecules, the amino acids are slightly modified and called amino acid residues. Unlike DNA, proteins have three dimensional structures as shown in Figure 2.2.

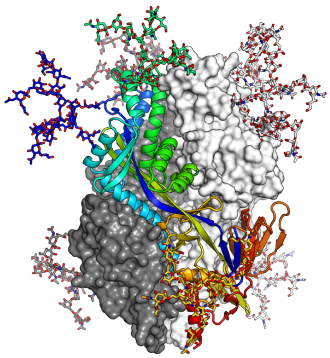

Figure 2.2: Three dimensional protein structure [1]

These amino acids can be classified into several types according to the 'hydropathy','volume' and 'chemical characteristics' classes as described in Figure 2.3.

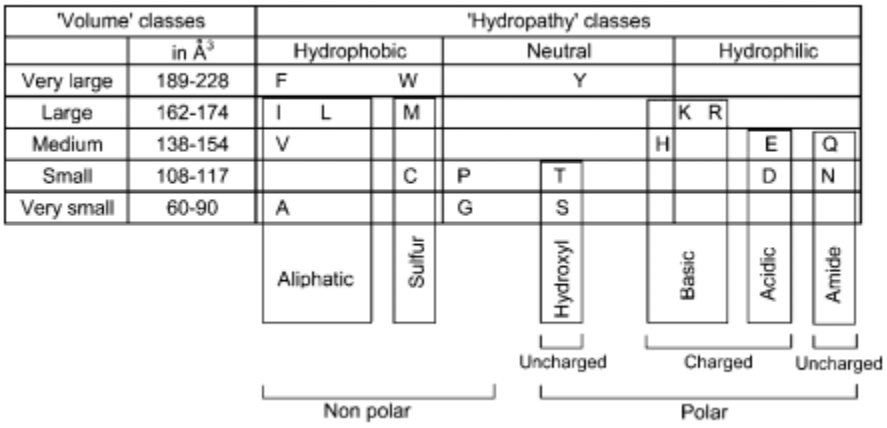

Figure 2.3: Amino acid categories.
IMGT classes of the 20 common amino acids for the 'hydropathy', 'volume', 'chemical characteristics' properties [27]

Now, a common question occurs how information is being passed from DNA to protein. Since DNA resides inside nucleus and protein synthesis happen outside of nucleus, it needs some kind of messenger to transfer the information to the protein, which is done by RNA.

Due to the double helical structure, DNA can replicate itself and can transcripts into RNA. Transcription is a biological process which copies a DNA sequence in the similar alphabet of RNA. The main difference between RNA and DNA is, in RNA thymine(T) is replaced with uracil(U). Then RNA participates in translation process to form amino acid chains, which are the base of proteins. This whole process flow is known as, *central dogma in molecular biology[2].*

$$DNA \longrightarrow transcription \longrightarrow RNA \longrightarrow translation \longrightarrow protein$$

2.2.1 Protein Structure

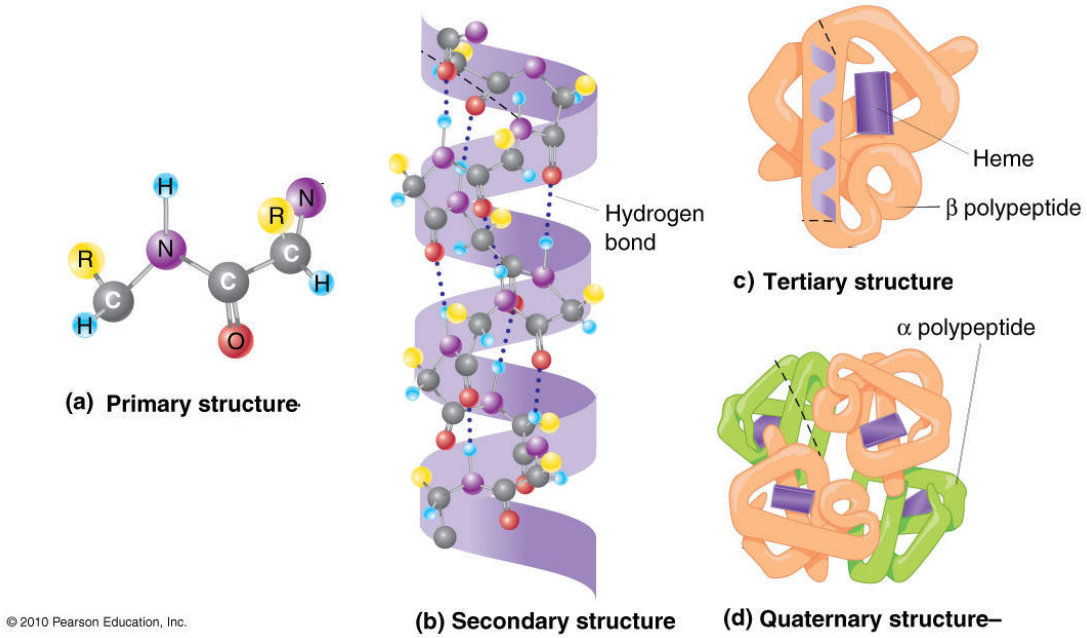

Figure 2.4: Four different structures of protein
They are primary, secondary, tertiary and quarternary respectively [3]

Proteins have four structures: primary, secondary, tertiary and quaternary structures. Primary structure of proteins is the sequence of amino acid residues in a polypeptide chain. Secondary structures are local folded structures that form within a polypeptide. Common secondary structures are: α helix and β pleated sheet, both of them are held by hydrogen bonds. The tertiary structure controls the functions of a protein. This is the structure where the polypeptide chain becomes functional. Some proteins are made up of multiple polypeptide chains, giving them quaternary structure. Figure 2.4 shows the different structures of protein.

The most widely available data for proteins can be found in Uniprot database [13] and the most used format to represent a protein is in FASTA format. In a FASTA file, there are two lines, the first line contains the protein name with a '>' symbol at the start, and the second one is the protein sequence where each letter encodes an amino acid. One example for a FASTA file is shown in Figure 2.5 where the amino acid sequence for the protein named P03051 is given in the second line.

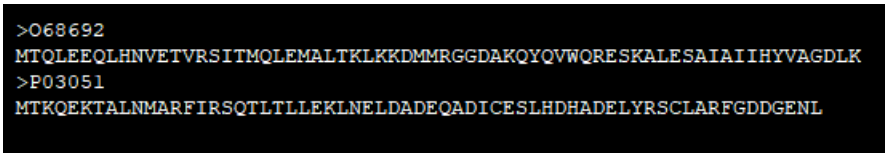

Figure 2.5: Example of a FASTA file
Each protein name has the prefix '>', followed by the protein sequence in the next line

2.3 Protein-Protein Interaction

Protein-protein interaction(PPI) prediction is a fundamental problem in molecular biology. Before start talking about our problem it is necessary to understand what PPI is. Protein-protein interaction (PPI) refers to intentional physical contacts between two or more proteins as a result of biochemical events. That means protein interacts among themselves by binding together. The amino acid residues that connect the proteins are called binding, or interacting residues. PPIs play important roles in various biological processes including cell-to-cell interaction, cell-cycle progression etc. The biological effects of PPI are as follows [6] :

1. Altering kinetic properties of enzymes, which may lead to subtle change in substrate binding effects.

2. Creating novel binding site for small effectors molecules.

3. PPI can inactive or suppress a protein.

4. Changing the specificity of a protein for its substrate by interacting with different binding partners.

Protein interaction data can be represented as a network where nodes denote proteins and edges refer to interactions. Sophisticated PPI networks are the one which create the difference between advanced human organisms and a lower organisms like worm [26]. Because of PPI information, we can identify the unknown proteins and understand their molecular process quite easily. Figure 2.6 shows the idea behind a PPI network. The edges in red are the unknown interactions and a PPI prediction task is to add these red edges to this network.

As shown in the figure 2.7, a FASTA file contains the information of binding and non-binding residues. First line denotes the protein name, followed by the amino acid sequence and their binding information. '1' indicates that the amino acid is a binding residue and '0' stands for non-binding residue.

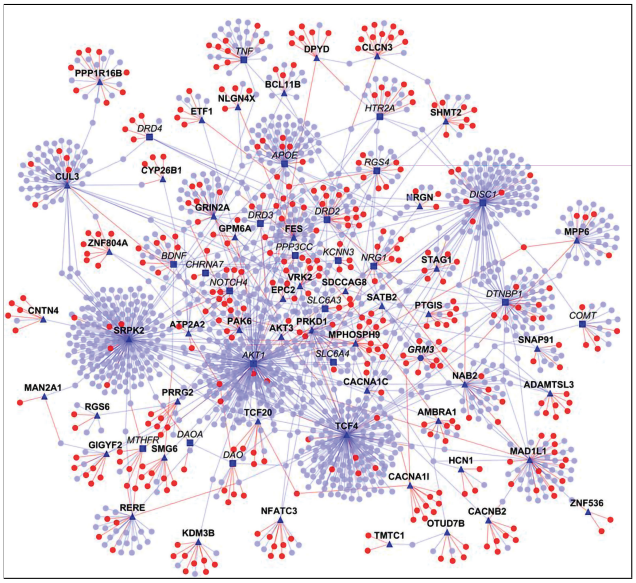

Figure 2.6: Protein-protein interaction network. From : www.genengnews.com

```
>Q81R67
MAEIKITPEELERIAGNFKNAAGEAQSQINRLEGDINSLEGQWAGATQAKFRGEFIQSKQAMQQYIPILEGISTDLKRIADKFRNTDNAY
10111111111111101110111111010111111011111111110011011101111111111111101101110111111111011000

>Q06253
MQSINFRTARGNLSEVLNNVEAGEEVEITRRGREPAVIVSKATFEAYKKAALDAEFASLFDTLDSTNKELVNR
11111111110111101111111111011010011111111100110111111111111111111111111111

>O68692
MTQLEEQLHNVETVRSITMQLEMALTKLKKDMMRGGDAKQYQVWQRESKALESAIAIIHYVAGDLK
01111111111111111111101110110111111110010011011011101101110111110

>3MP7B
IRHFWKESRRAFLVTKKPNWATYKRAAKITGLGIILIGLIGMLIRIVGILILGG
00011011011101111111011001111110111011111101111111100
```

Figure 2.7: Example of a FASTA file with binding sites information.

2.4 Protein-Protein Interaction Binding Site Prediction

All the proteins interact among themselves by binding together. The binding site prediction problem is: Given a protein sequence, predict which residues interact. According to various biological studies, we can sort PPI binding site prediction into two categories :

1. Sequence based methods : This is the most widely used method based on sequence information that use several features extracted from the protein sequence. In our program, we use sequence based method.

2. Structure based methods : Some PPI site predictors use protein's 3D structural information for prediction.

Identification of protein interaction binding site is an important task in bioinformatics for understanding the function of the protein, cell regulatory mechanism and drug design. Inspite of much research in this area, the best classifiers still require improvement. For PPI binding site prediction work, the primary sequence with the known binding sites in a FASTA file is

used as input data to train the model and later this model predicts the binding site for unknown proteins.

2.5 Deep Learning and its Application in Bioinformatics

In the last decade, deep learning is the most talked-about technology and has taken the world by storm. Almost in every field it replaces tradition algorithmic based approaches. But deep learning is not a new concept, it exists from 1980s.

Figure 2.8: Handwritten digit classification example
Predicted value is correctly showing as '7' [4]

But to train a deep learning network a lot of data and high speed machines are required which were not available back in the 80's. So the rise of deep learning started in 2012 with AlexNet [19]. Initially deep learning methods were applied specifically to computer vision and natural language processing(NLP) problems like image classification, handwriting recognition (Figure 2.8). But as the computing power increased, gradually it took over other domains. Figure 2.9 shows how the performance of deep network based approach improves with the amount of data over general algorithmic based approaches.

Unlike computer vision and NLP problems, biological problems are much more complex because of their large, complex, imbalanced data and sometimes biologists can not give concrete explanations how those mechanisms are related. Still in the last decade the number of bioinformatics problem using deep learning gradually increased, because of many factors: improved biological databases, deep learning frameworks like Tensorflow [7], Pytorch [25], libraries like keras [11] etc. In figure 2.10 we can see how much deep learning has taken over biological problems in the last few years.

2.6 Deep Neural Networks

Neural networks are inspired from the activity of the human brain. They are made of layers of neurons which learn complex relationship of data. Neurons are basically mathematical functions which are interconnected through multiple hidden layers thus passing the information from one node to another as shown in the Figure 2.11. A neuron takes one or more inputs and applies mathematical functions on them to generate an output called hypothesis. Input to the neural network is multiplied with a weight, setting the importance of the input. After

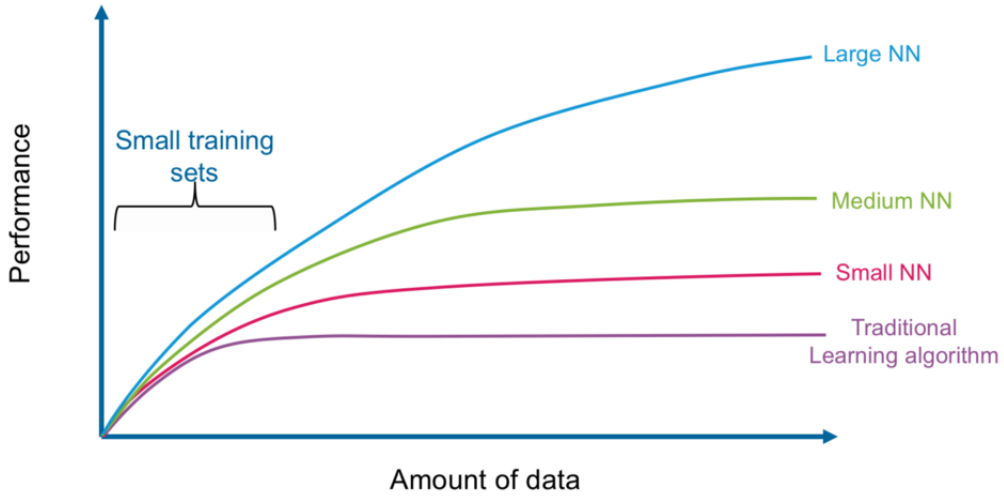

Figure 2.9: Performance comparison with the amount of data between deep learning based approach over other algorithmic approaches. From https://mc.ai/

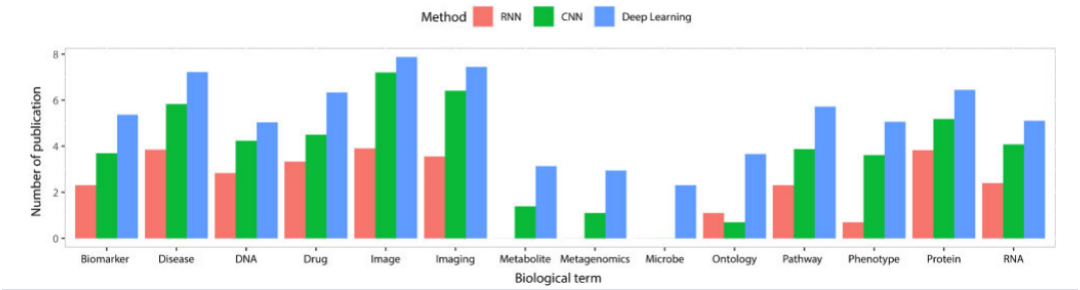

Figure 2.10: Number of publications in several biological topics using deep learning. Bar graphs show the number of publications mentioning terms like 'CNN, 'RNN', 'deep learning'. [21]

calculating the output, a non-linear function is applied to it, which is known as activation function.

Common activation functions are Sigmoid, tanh, ReLU etc. Figure 2.12 shows how a single neuron in the network works.

2.6.1 Convolutional Neural Networks

Convolutional neural networks (CNN) is one popular version of neural networks which was specifically designed for analysis of vision related applications. It takes the input and applies a convolutional operation to classify or predicting a certain application. The computation of a convolution operation is as follows. The key elements of convolution operation are: Input matrix, filter, and output matrix. A sliding window is initially placed on the first row of the input and shifted through the entire matrix. Then an elementwise multiplication is performed between each submatrix of the input matrix and the filter. The summation of the elementwise

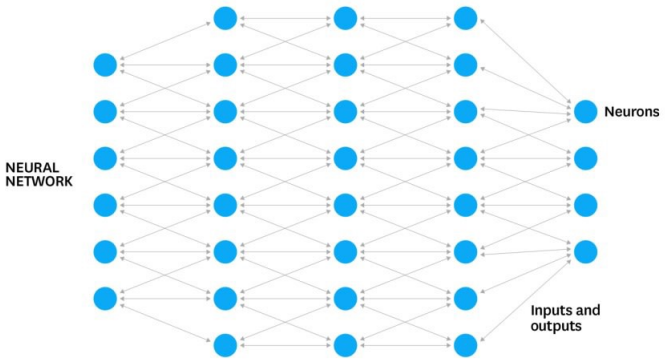

Figure 2.11: Deep Neural Networks. From towardsdatascience.com

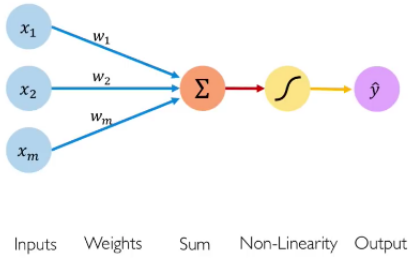

Figure 2.12: Working of a neuron.
x_i is the input, w_i is weight. From medium.com

multiplication is placed as one value in the output matrix. The idea of CNN can be compared to the pattern of a human brain. Widely used layers in CNN architectures are : Convolutional layer, Pooling layer and fully connected layer. This architecture has been used in several state-of-the-art technologies like ImageNet [14], AlexNet [19], VGGNet [32], ResNet [15] etc. Figure 2.13 shows the design of a standard convolutional network.

2.6.2 Recurrent Neural Networks

To deal with sequential data, recurrent neural networks (RNN) come into the picture. For application like text processing, voice recognition, biological sequence analysis RNNs are the best suited architecture. RNNs are the most effective because they can keep track of the information of previous timesteps, which is an essential for any kind of bioinformatics application. RNN generally takes two types of input, one is from current timestep, say t_i, and another is the output of the previous one, t_{i-1}. But often when we have long sequential data, RNN suffers from the vanishing gradient problem [24]. While performing the backpropagation, each node in the network calculates its gradient with respect to the effects of the gradient in the previous timesteps. The value of the gradient may become zero due to repeated multiplication of small numbers. Hence, the importance of the beginning of the input sequence can be lost. To tackle this, some modified versions of RNN like Long short term memory (LSTM) [16], gated recurrent unit (GRU) [12] were proposed later. In Figure 2.14 we can see a classic architecture of a

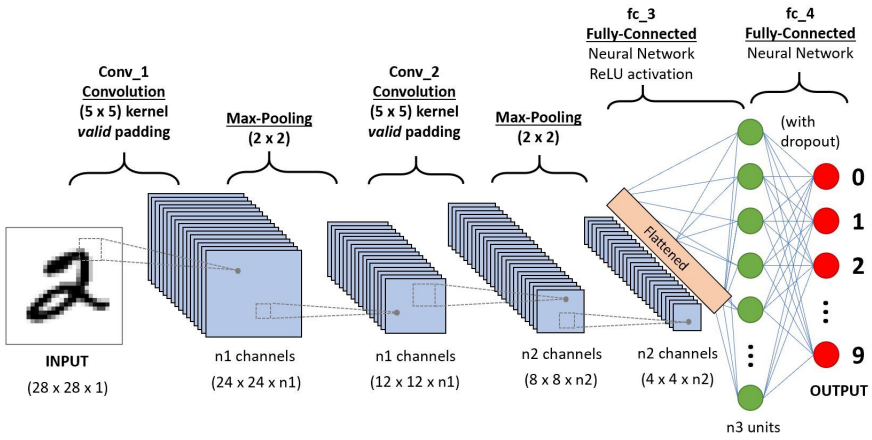

Figure 2.13: Standard architecture of a convolutional neural network
It generally consists of convolutional layer, pooling layer and fully connected layer. From towardsdatascience.com

recurrent network.

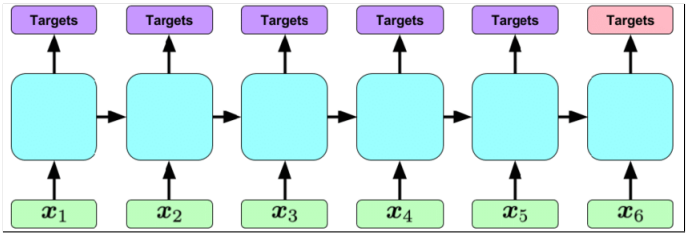

Figure 2.14: Classic RNN architecture
With x_i as input to each timestep and target is the output from each timestep. From ResearchGate

LSTM and GRU remember long term dependencies through various gates like input gates, forget gates, output gates, update gates, reset gates etc. There are activation functions associated with the gates which in turn decide what information to remember and what to forget. Basic diagram for a RNN, LSTM and GRU unit can be visualized in Figure 2.15.

Implementation of LSTM, GRU and basic RNN layers in tensorflow looks like this :

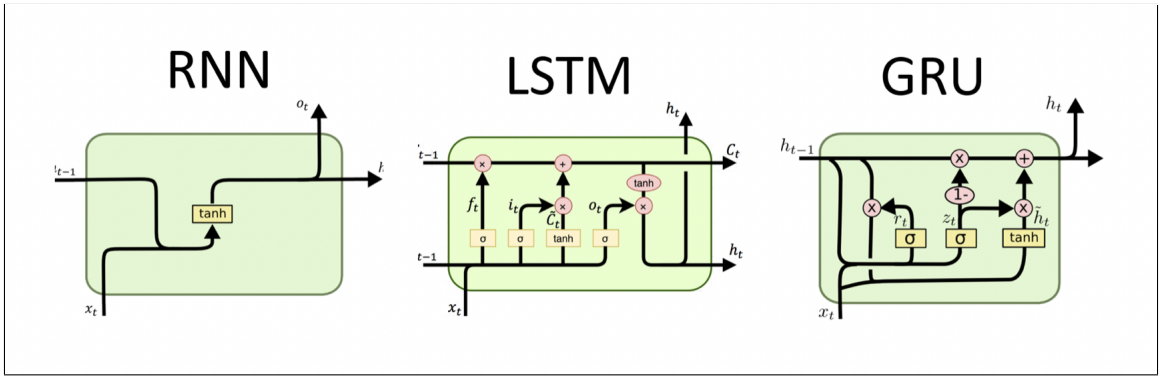

Figure 2.15: Basic unit of a RNN, LSTM and GRU. From http://dprogrammer.org/rnn-lstm-gru

```
import tensorflow as tf
from tf.keras import layers
#RNN layer
tf.keras.layers.RNN(cell, return\_sequences=False,...)

#LSTM layer
tf.keras.layers.LSTM(units, activation='tanh',...)

#GRU layer
tf.keras.layers.GRU(units,activation='tanh',...)
```

2.7 Opacity in Deep Learning

Since deep learning models are composed of highly non linear functions and complex mathematical algorithms, it is difficult to understand what is happening inside the box. With the help of high speed computing systems we can build a model that can give a very high accuracy without any human intervention, but the question still remains why it is performing so well. Over the last three to four years, scientists are specifically interested in the interpretation of these models, that is, in finding what is happening inside the *black-box*. When we build an algorithmic based software we know the inner mechanisms of the algorithm, in other words we *trust* the system. In the biological and healthcare domain many applications are designed with deep learning models, and we are using them for diagnosis of patient, to perform simple surgeries, monitor a patient's condition etc. which are very sensitive areas. When a doctor performs a diagnosis or a driver drives a car we can trust on their abilities, but when a machine learning system is doing the same we can not rely on them completely. Hence, interpretability is especially important for the domains like these where we need to figure out what aspect of the data model is concentrating while obtaining the decision. This area of research is pretty new and emerging so there are a lot of work to be done in this area. Most of the popular interpretation methods have developed for computer vision (images) and for natural language

processing (texts) application. Though in the last few years biological problems are adapting the existing interpretation methods, still it is far from satisfactory. We will discuss some related works for bioinformatics application in the next section.

2.8 Visualization and Interpretation of Biological Molecules

In this section, first we will discuss the works done for visualization of general machine learning applications and then we will come to biological problem's interpretation. We will also discuss some of the limitations of existing machine learning software to use in biological applications.

2.8.1 The Unreasonable Effectiveness of RNN

In 2015, Karpathy et al. [17] first showed using static visualization technique how the hidden cells in the recurrent neural network are performing in language models. Their work showed how a recurrent neural network learned the representation on real world data. They used Leo Tolstoy's *War and Peace (WP)* and the source code of *Linux Kernel (LK)* as their datasets. Then they compared the results of the different recurrent units like LSTM, GRU and vanilla RNN to explain which is the most powerful one to remember long term dependencies, however they could not come to clear conclusion between the power of GRU and LSTM.

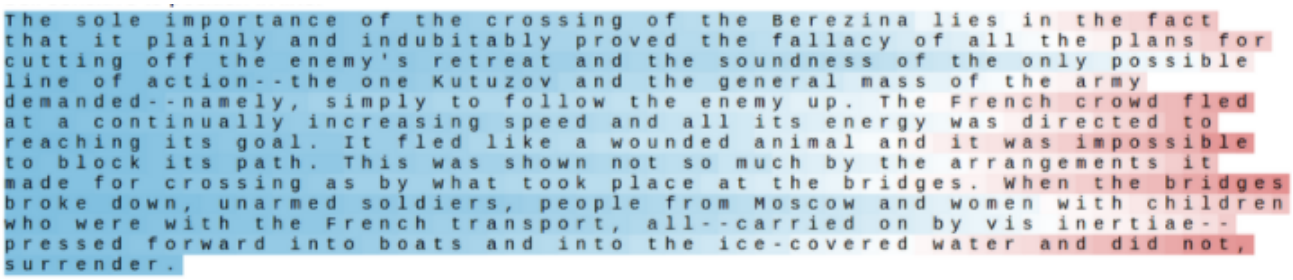

Figure 2.16: A hidden cell of recurrent units which keeps track of the new line.
The change of colour from blue to red indicates that the network clearly identifies the new lines of a text [17].

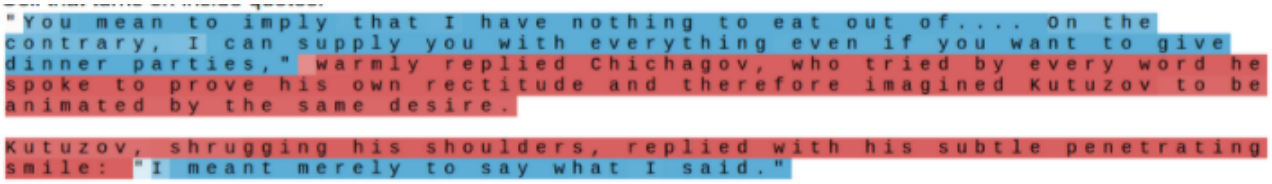

Figure 2.17: A hidden cell of recurrent units which keeps track of the quotes.
We can see that the colour changes to red while it is outside of a quote [17].

Both of them outperform vanilla RNNs, but their performance is mixed. To the best of our knowledge their work was the first to implement visualization technique to interpret a deep neural network. Figures 2.16 and 2.17 show how some particular hidden cells in LSTM keep track

of the information the deep learning models identify as *important*. These visualizations show how much is happening inside the model, because in the training time they did not specifically trained the model to remember these attributes, whereas the model learns itself and understands that these features are probably the important ones for a language model.

2.8.2 LSTMVis - A Tool for Visual Analysis of Hidden State Dynamics in RNN

Unlike [17], Strobelt et al. [34] developed a software tool to visualize the hidden state properties of LSTM. Their tool is composed of two different visual analysis components: *Select View* and *Match View*. In Select View, Figure 2.18 they wanted to show the visual representation of hidden states, which are getting activated for that specific hypothesis. Here the selected phrase 'a little prince' is the hypothesis.

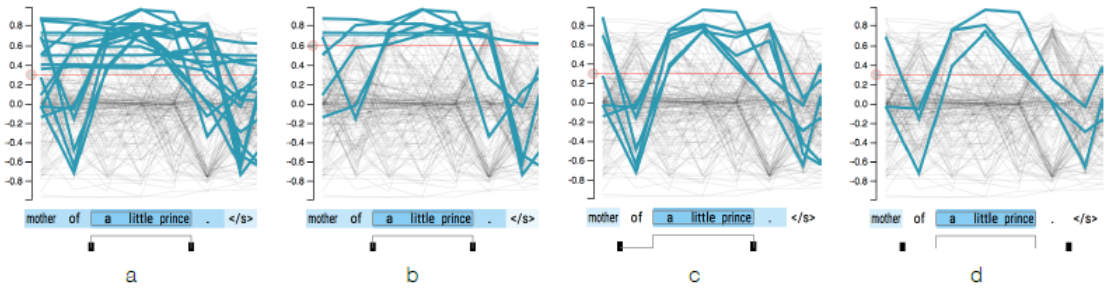

Figure 2.18: LSTMVis Software Description
(a) The selected hypothesis is *'a little prince'*, blue highlighted ones are the hidden cells which gets activated for this particular hypothesis selection with a certain activation threshold. (b) Blue highlighted hidden cells are the ones which gets activated for the same hypothesis but with a high threshold. (c) This shows the activated hidden states which have the activation threshold (l) equal or lower than the selected l after seeing the character 'of'. (d) It removes the hidden states above l after reading '.' [34]

In the *Match View* they proved that the selected hypothesis does have some meaning and there are some other hidden cells which get activated for a similar kind of hypothesis. Figure 2.19 shows the activated hidden cells in the blue line plot, for the hypothesis at the fourth level of nesting, and Figure 2.20 shows the presence of the all matched hidden cells in the network capturing that particular pattern.

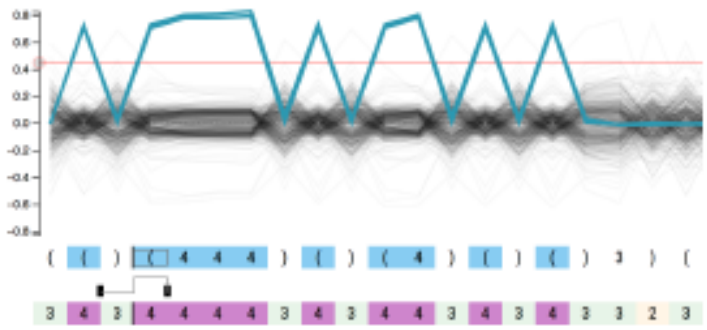

Figure 2.19: An example of 'Select View'
for the hypothesis at the fourth level of nesting. Selected hidden states in 'blue' show the activated hidden cells in the network [34].

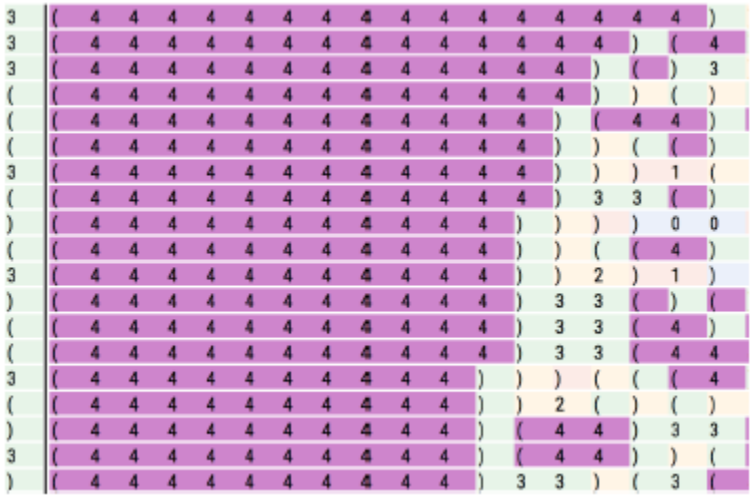

Figure 2.20: These are the hidden cells which show that the network indeed capture the nesting level 4 phrases of different lengths. [34].

2.8.3 RNNVis - Understanding Hidden Memories of RNN

Based on similar concepts as above, Ming et al. [23] developed another software which can explain and interpret the function of different hidden cells on the basis of particular inputs. Their visualization technique is a bit different than LSTMVis [34], they have shown the similar kind of hidden cell clusters as a memory-chip.

Figure 2.21 shows that words like 'they', 'he', 'she' which are pronouns form a cluster, prepositions like 'with', 'for', 'by' create a separate cloud. When they selected the word cluster 1, the first and fourth hidden cell clusters get selected, which indicates that these are the specific hidden cell units which are activated for the prepositions. They also explained few different case studies like sentiment analysis, language modelling etc. to explain the behaviour of a recurrent network.

One potential drawback for both the LSTMVis and RNNVis software is that they are par-

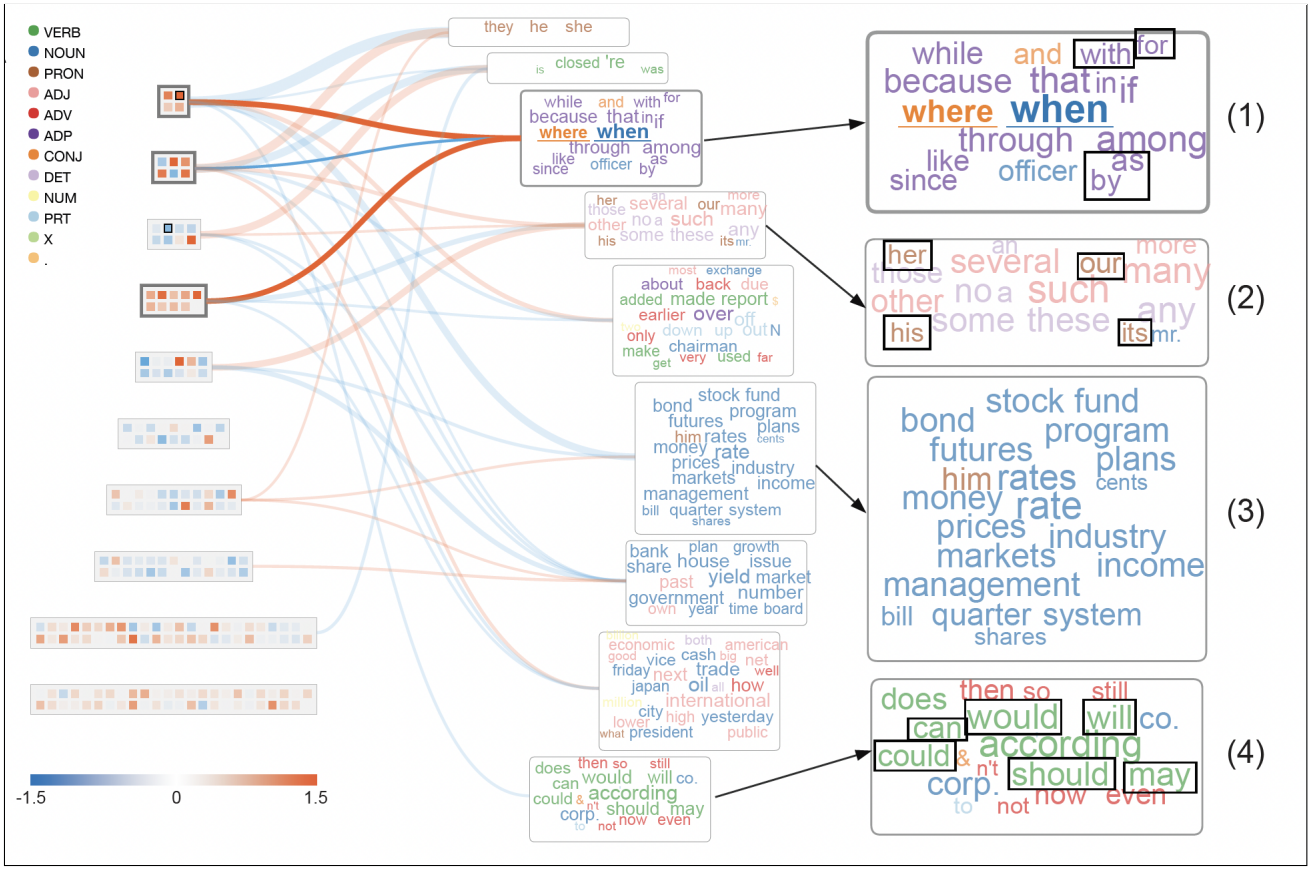

Figure 2.21: A visualization from RNNVis tool to show the hidden cell's activation. At right hand side, small squares inside the rectangles show the hidden cell clusters, in the middle we can see the words coloured according to Parts of Speech(POS) tags. [23]

ticularly dependent on the model type, and to some extent to the type of application. These softwares do well in NLP related applications because for texts we know the meaning of words, and we know what we are looking for. But in biological applications, we generally have numerical data and many complex features. That's why these tools do not show much effectiveness in biological applications.

2.8.4 Deep Motif Dashboard

Lanchantin et al. [20] developed a toolkit called the Deep Motif Dashboard (DeMo Dashboard) for visualization of the patterns of DNA sequences from deep neural network (DNN) for transcription factor binding site (TFBS) classification task. They had investigated the outputs of three types of DNN models: recurrent, convolutional and convolutional-recurrent networks. Their model basically works as a binary classifier. It classifies a DNA sequence to be a positive or negative TFBS sequence. After the prediction, they wanted to investigate the nucleotide importance of each DNA sequence. In other words their aim was to find the parts where the model focuses more while doing the prediction. They used one-hot encoding to represent the nucleotide (A,C,G,T). One-hot encoding is a method that quantifies categorical data. It produces a vector with the same length as the categories in the data set. It assigns the value 1 to a data point that belongs to a given category, whereas other positions are assigned value 0. For example, [0,0,1,0] is a valid one-hot encoding for a data point belongs to category 3. This input goes to the model which classifies the sequence to binding or non-binding sequence after passing through softmax. Softmax is an activation function that converts a vector of numbers into a vector of probabilities, where the probability of each value is proportional to the relative scale of the values in the vector. Figure 2.22 shows their model architecture :

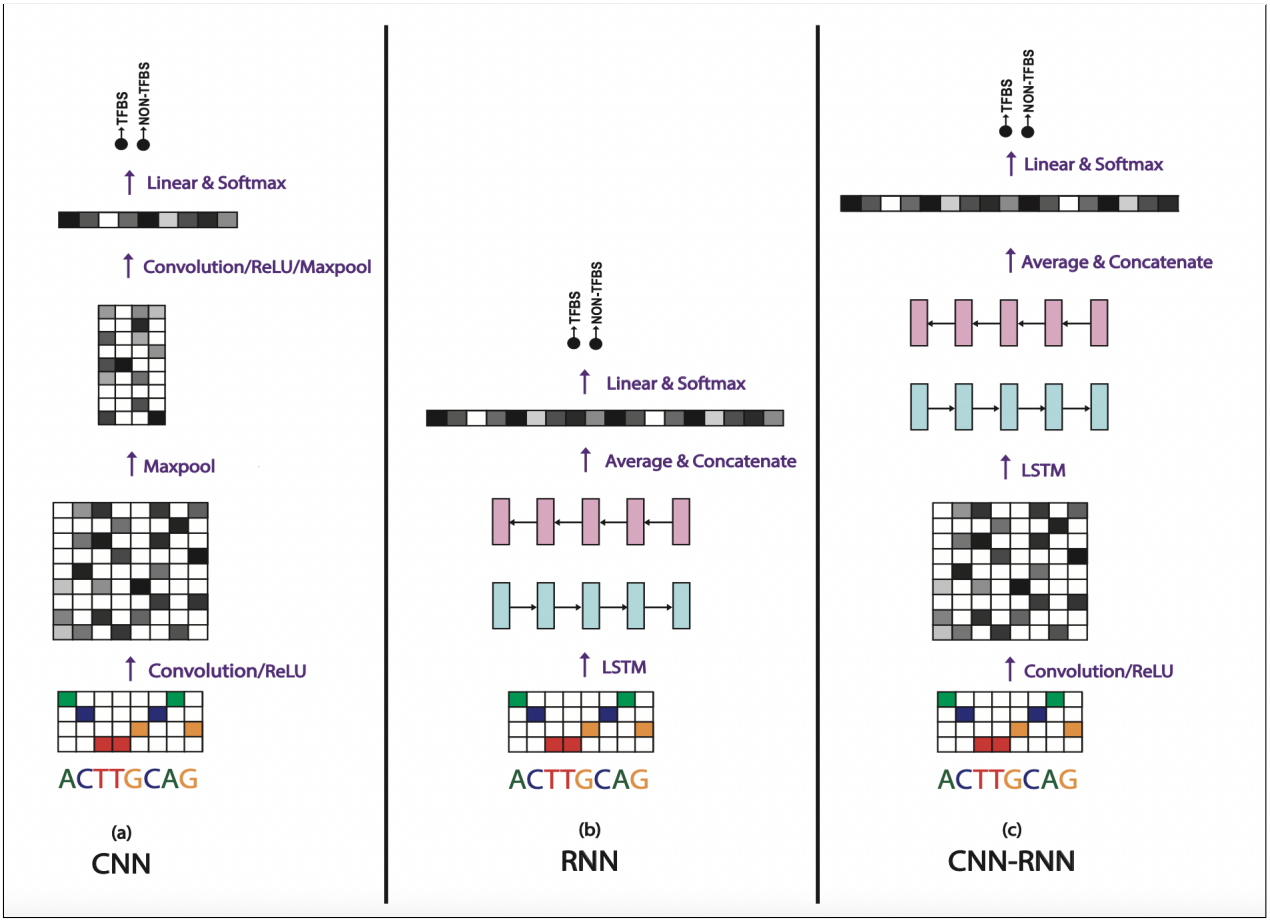

Figure 2.22: DeMo Dashboard architecture
Model is taking one-hot encoded sequence as input and then passed to the respective models. The prediction score is then passed through softmax unit to get the classification. [20]

They used the concept of *saliency map* [31] to visualize the important parts of the sequence. This method is first introduced for images by Simonyan et al. The authors adapted this method and implemented it for a bioinformatics application. Later they compared the identified motif obtained from the sequence using saliency map approach with the known JASPAR motifs for that particular dataset and surprisingly it shows that the model focuses on that JASPAR motif to predict the class labels for the sequence.

Table 2.1 shows the extracted information from the DeMo dashboard where the authors successfully interpret the important patterns from a test DNA sequence and visualize it. Their work is among one of the few where a computer vision concept is applied to a bioinformatics problem for interpretation purpose. We will discuss the concept of saliency map in detail in Chapter 3.

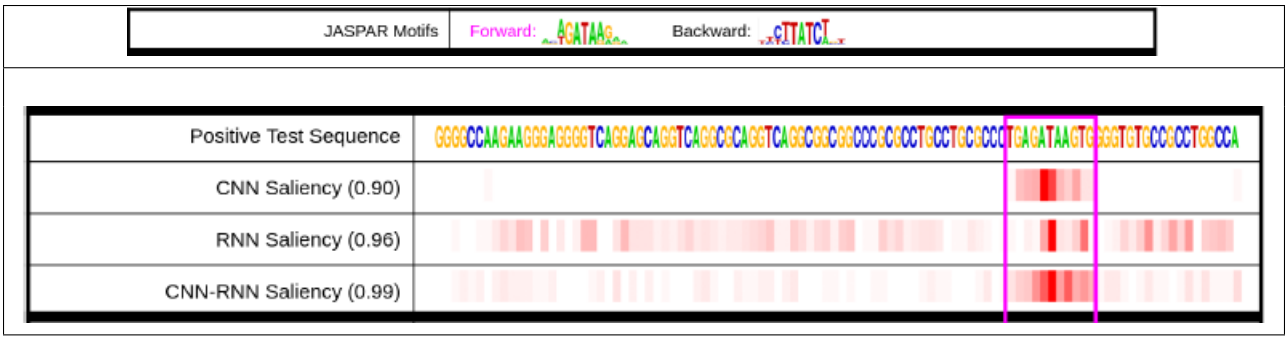

Table 2.1: An example from the DeMo dashboard.
On top, standard JASPAR motif is shown for the GATA1 dataset. At bottom, we can see the saliency maps for a specific positive test sequence (binding sequence). The pink rectangular box shows the part where we can see highest heat in all three models, and this part matches directly with the JASPAR motif.

2.8.5 DNA Transcription Factor Binding Site Prediction

This work concerns finding the motifs for DNA in transcription factor binding site prediction [37]. Authors used the same strategy of saliency map to identify the feature importance for a DNA sequence. The motif obtained from the saliency map is very similar with the true regulatory motif for that sequence. True regulatory motifs are known subsequences of a DNA which hold biological importance in transcription factor binding sites.

Figure 2.23: Visualization for a DNA sequence from this paper.
This bar graph shows the magnitude of the saliency values against the nucleotides in the sequence. Green coloured area is the motif identified by the model which in turn is the true regulatory motif for the sequence. [37]

In Figure 2.23 we can see that they coloured the bars in green where the saliency values are very high for the nucleotides. The deep learning model learned that this sub-part of the sequence is the one where it needs to focus while doing the prediction in spite of having zero information about this.

These are our background studies for this problem. We observed tools like LSTMVis,

RNNVis can be applied to visualize the hidden state dynamics for recurrent networks specifically in NLP task. Later we looked into some interpretation techniques used in bioinformatics applications which pose more complex data than general NLP taks. Most of the available interpretation techniques are either dependent on model or on the input data structure. Also, the interpretation techniques used in bioinformatics application are only for DNA (4 bases) which is much simpler than a protein (20 bases). According to the best of our knowledge, visualization and interpretation methods of DNN models for proteins are yet to be performed. Unlike other problem, our input data structure is much more complex and it uses a sliding window protocol to predict each residue.

Chapter 3

Methodology

In this chapter we describe the programs for protein binding site prediction that we used for our investigation: DELPHI [22] and DLPred [36]. Then we cover the visualization and interpretation techniques we have used to study the behaviour of the two programs.

3.1 DELPHI

DELPHI's model consists of three different parts: a convolutional neural network (CNN), a recurrent neural network (RNN), and an ensemble model which takes the output of CNN and RNN part as its input. The CNN component has one convolutional layer, maxpool layer and a fully connected layer with dropout. The RNN section, uses bidirectional gated recurrent units (GRU) layers.

3.1.1 Features

DELPHI model uses 12 feature groups to represent a protein sequence. These features are: high-scoring segment pair (HSP), position information, position-specific scoring matrix (PSSM), evolutionary conservation (ECO), relative amino acid propensity (RAA), putative protein binding disorder, hydropathy index, physical properties, physicochemical characteristics, putative protein relative solvent accessibility (RSA), 3-mer amino acid embedding (ProtVec1D) and PKx. It uses a 39 dimensional feature vector to represent each input. Table 3.1 shows the features and their dimension in the tabular format. All these features are computed from the protein using various tools; see [22] for details.

3.1.2 Model Structure

DELPHI's model follows many-to-one structure while doing the prediction. This is because to get the prediction of the centered amino acid residue we are using the information of neighbouring 30 residues. As shown in the Figure 3.1 we are using sliding window of size 31 to get the prediction of the centered amino acid. In the pre-processing part, we have used padding of zeros of size 15 in both the beginning and end position, so that we will have *protein length* number of windows for every protein sequence. With the sliding window we can pass useful

Feature	Dimension
High-Scoring segment pair (HSP)	1
Position information	1
Position-specific scoring matrix (PSSM)	20
Evolutionary conservation (ECO)	1
Relative amino acid propensity (RAA)	1
Putative protein binding disorder (RSA)	1
Hydropathy index	1
Physical properties	7
Physicochemical characterestics	3
Putative protein relative solvent accessibility (RSA)	1
PKx	1
3-mer amino acid embedding (ProtVec1D)	1

Table 3.1: DELPHI features

information from one residue to another and also the deep learning model learns well for each residue as its information is getting passed to the model multiple times. Also, this concept triggered the idea of this research, to find important and influential positions from a window through which the deep learning model is getting most of the information for the prediction.

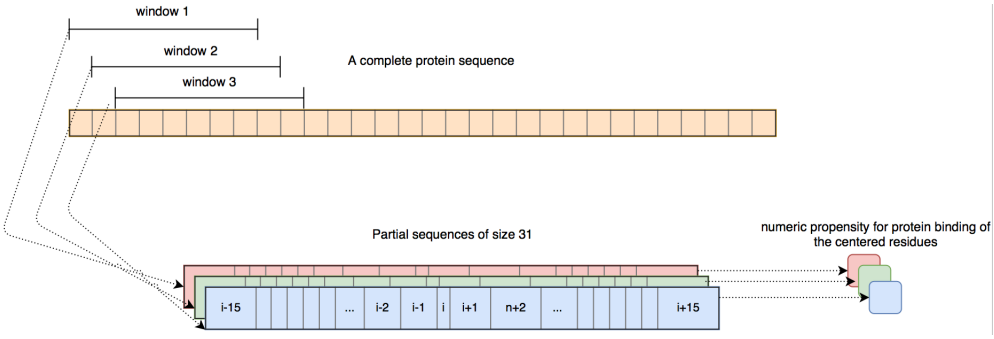

Figure 3.1: DELPHI prediction window.
We are placing a sliding window of size 31, with stride 1 over the input protein sequence, and the prediction score for the middle amino acid is the output we get from the model [22].

CNN architecture Figure 3.2 shows the architecture of the CNN model implemented in DELPHI. It has four different layers: one convolutional layer, one maxpool layer, one flatten layer, and two fully connected layers with dropout. As shown in the figure the input protein sequence feature of size 39*31 is passed through the convolutional layer, followed by a max-pool layer. This output from the maxpool layer is flattened by a flatten layer and then it acts as an input for two fully connected layers with a dropout. The dropout is added to perform regularization [33]. Regularization is a machine learning concept used to avoid overfitting. In the last layer we have a sigmoid activation function which gives us the output as a float value (between 0 and 1) which is the numeric propensity of a residue.

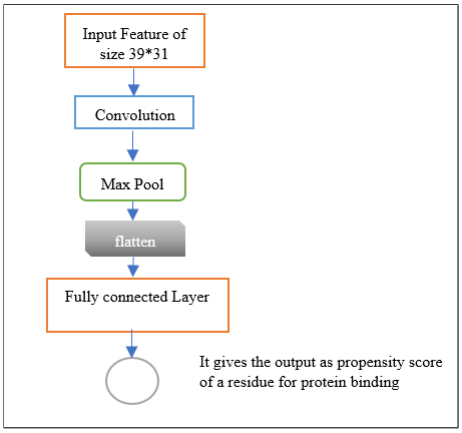

Figure 3.2: CNN architecture of DELPHI model

RNN architecture Figure 3.3 shows the architecture of the RNN model implemented in DELPHI. It has three different kinds of layers: one bidirectional GRU, one flatten layer and two fully connected layers with dropout. Input to the RNN is the same 39*31 shaped feature vector which is getting passed to the bidirectional GRU layer.

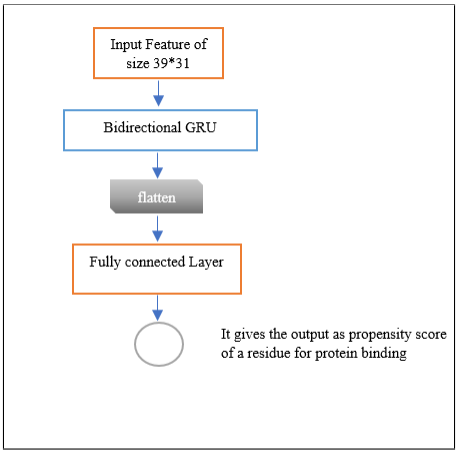

Figure 3.3: RNN architecture of DELPHI model

As we discussed in the background chapter, recurrent networks can capture long term dependencies very well and researchers like Karpathy et al. [17] already proved that they can be

used to capture this information. So here we aim to capture the relationships and information that neighbouring residue passes to the centered amino acid in each window. Finally the output of the GRU layer is flattened and then passed into the fully connected layer with dropout. Like the CNN model, RNN also gives us a float value (between 0 and 1), because of the sigmoid activation function, which is the numeric propensity of a residue.

Ensemble architecture The ensemble model of DELPHI is the combination of CNN and RNN models. Figure 3.4 shows the model structure of the ensemble model. It also takes the input feature vector of shape 39*31 and then computes the weights for the CNN and RNN model separately. Then the flattened output from the CNN layer and RNN layer are concatenated and then passed in two fully connected layers with dropout. Similar as above architectures, in ensemble model we have sigmoid activation at the last layer hence a float value between 0 to 1 is obtained, which is the numeric propensity of the residue.

One important thing here to note is, we load the best performing model weights from the CNN and RNN architecture in a file and then we train the fully connected layers of the ensemble model using the training and validation data. The output of this model is also a real number between 0 and 1, which is the propensity score of a residue, that is, the probability that this particular amino acid residue is binding or interacting. This ensemble model is the final model of DELPHI because it is the best performing model among these three.

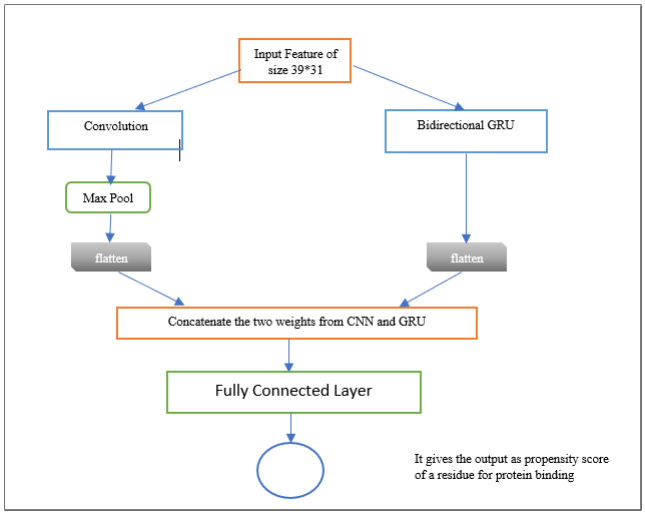

Figure 3.4: Ensemble model architecture of DELPHI

3.2 DLPred architecture

Zhang et al. proposed another software named DLPred which performs protein protein interaction binding site prediction. They introduce and use a simplified long short-term memory (SLSTM) network to implement their architecture. The original DLPred architecture does not use a sliding window protocol like DELPHI, instead they predict interactivity of the amino acid residues at once for the entire protein sequence. We have reimplemented a modified version of DLPred in order to incorporate the sliding window concept of DELPHI so that we can focus on the importance of neighbouring residues for each prediction. We discuss the model structure and features used in DLPred in the following subsections in detail.

3.2.1 Features

DLPred uses 8 feature groups to represent a protein sequence. These features are : Position-specific scoring matrix (PSSM), physical properties, hydropathy index, physiochemical characterestics, PKx, 3D-1D score, conservation score and protein sequence coding. It uses a 73 dimensional feature vector to represent each input. Table 3.2 shows the features and their dimension in the tabular format. All these features are computed from the protein sequence using various tools; see [36] for details.

Feature	Dimension
Position-specific scoring matrix (PSSM)	20
Physical Properties	7
Hydropathy index	1
Physiochemical characteristics	3
PKx	1
3D-1D score	18
Conservation score	1
Protein sequence coding	22

Table 3.2: DLPred features

3.2.2 Model Structure

We have redesigned the model structure of DLPred and used a sliding window of size 31 to predict the centered amino acid of a protein sequence in the modified model architecture of DLPred. We have used a padding of zeros of size 15 in both the beginning and end position of a protein sequence, so that we can have *protein length* number of windows for every protein sequence. We use this concept so that the useful information can pass from one residue to another and also we can focus on the important positions in a window for each prediction. The modified input for the DLPred is similar to Figure 3.1. We use three stacked bidirectional GRU layers followed by two fully connected layers in the model architecture.

3.3 Visualization and Interpretation Techniques

We discuss next the methods we have used to interpret the model's prediction. Also, it is important to visualize what we understand from this interpretation. We cover them both in this section.

To explain a deep learning model's behaviour there are few techniques proposed by researchers. But we can not just simply use them for any application. Some of the techniques depend on the model's structure, specifically on the layers used. As example, DeepLIFT [30] is a known method which gives the importance score to each input feature for a certain model's output. But one drawback of this method is, it can only be applied to convolutional neural networks. So, if the model consists of other layers like LSTM, GRU we can not use this method. Similarly techniques like LRP [10], PatternNet [18] are also compatible only with convolutional networks, so in our case we cannot use them to interpret our model architecture. We have implemented three methods sensitivity analysis, saliency map and integrated gradients to find out the importance of amino acid residues for the prediction. We also discuss the python package *innvestigate* [8], a library that provides the implementation of several model interpretation techniques we have used for our experiments.

We can divide the explainabilty techniques into two types : Model Specific and Model Agnostic. Model specific interpretations are limited to specific model architectures, they particularly rely on the internal structure of the machine learning algorithm. One advantage of this is they can generate more precise explanations as they are directly dependent on the model structure. DeepLIFT [30] is a model specific interpretation technique. On the other hand, model agnostic techniques treat the model as a black-box, hence it does not matter what layer or architecture we have used in our model. These methods generally analyze the data from input and the decision (output) of the model to explain the important features. LIME [29] is an example of model agnostic technique. We have also implemented LIME for DELPHI to see the most influential feature for the model's prediction.

3.3.1 Sensitivity Analysis

Sensitivity analysis tries to analyze the important input features with respect to model's output for an input sample. Mathematically, it computes model's gradient for each particular data point. This is a local explanation technique, which checks which features are the most sensitive to the output. Mathematically it can be defined as square of the gradient. Consider we have a data point x and the model's output $y \in \mathbb{R}$, computed by the model using the function $y = f(x)$, then the sensitivity analysis will compute a relevance score $R_i(x)$ which is computed by using the square of Jacobian of the outputs with respect to that input :

$$R_i(x) = [\frac{\partial}{\partial x_i}(f(x)]^2 \tag{3.1}$$

This method is very fast and easy to compute because we can get the gradients of the output w.r.t each input by one step backpropagation. A higher value of $R_i(x)$ indicates the prediction is sensitive to that input i. This means that an input i with high $R_i(x)$ value is more

likely to influence the model's prediction on input x. Generally the function of neural networks are highly non-linear, hence it can vary between points, can be large in some area, or can be zero in others. This approach is widely used as an explanation method in many deep learning application especially in vision related work like image classification, for sentiment analysis etc. Below is the implementation snippet of this method using *innvestigate* python library. This package we can simply install using *'pip install innvestigate'*.

```
import innvestigate
from keras.models import load_model

model = load_model('DELPHI.h5')
analyzer = innvestigate.create_analyzer("gradient", model, postprocess="square")
analysis = analyzer.analyze(test_feature)
```

Listing 3.1: Code snippet of sensitivity analysis

Figure 3.5 shows a visualization when we apply this interpretation technique on a convolutional model created for digit classification using MNIST dataset. We gave two digits 4 and 0 as test input. The model classifies the digits correctly. When we apply sensitivity analysis, we can see in the heatmap that the edges of the digits are coloured in the shades of whites which indicates that those pixels give the most useful information to the model hence it is predicting the input to that particular class.

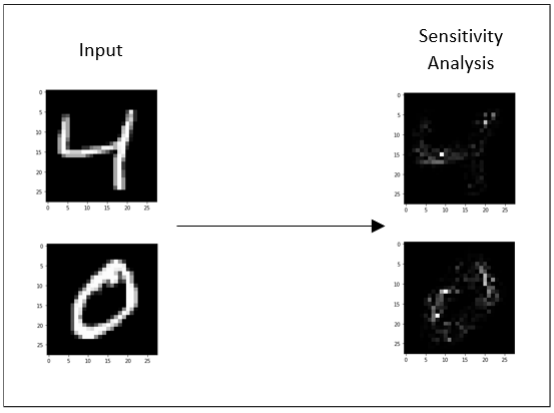

Figure 3.5: Sensitivity Analysis on MNIST dataset

In DELPHI, the shape of an input feature is L * 39 * 31, where L is the protein length. We apply a sliding window of size 39*31 on the input with a stride 1. Every time we pass the information of 31 residues and predict the output of the centered amino acid. When we apply this method to the model, we get an output of shape 39*31. Since we want the contribution of each position (or residue), we sum the 39 values corresponding to that position to obtain one value which represents the relevance score for each residue in that window. So, we get an array of size 31, normalized between 0 to 1, for each input window as the output of sensitivity analysis.

In DLPred, we have 73 dimensional features, so an input protein sequence is represented by L*73*31, where L is the protein length. We apply sensitivity analysis to get an output of shape 73*31. We sum up each 73 values to obtain a single score which represents the relevance score for one residue in that window. Similar to DELPHI, we get an array of size 31, normalized between 0 to 1, for each input window as the output from sensitivity analysis. Figure 3.6 shows how we perform this operation for a single window.

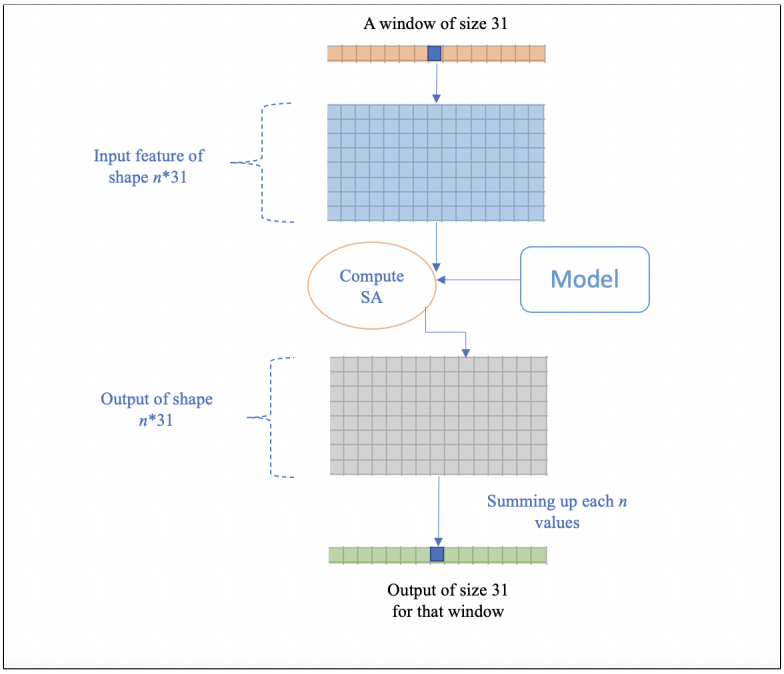

Figure 3.6: Flow diagram of sensitivity analysis for a window. n denotes the dimension of feature vector. For DELPHI n is 39, for DLPred it is 73.

We denote the amino acid residue at position i by $P[i]$. A square in the position (i, j) in the figure indicates the contribution of the residue $P[i]$ to the prediction for the residue $P[i+j-15]$, where $0 \leq i \leq L-1$, $0 \leq j \leq 30$. Our aim is to look for the important amino acid residues which act as the most influential for the model. In the visualization (Figures 4.1 - 4.4), each little square in the plot is coloured by a shade of red, darker shades indicate greater contribution. So, if we get a column coloured with dark shades of red, we can think that the residue $P[i]$ has a good impact in the model's prediction. Figure 3.7 shows the structure of the visualizations for a protein sequence. We apply this same concept while making the visualization for all three methods : Sensitivity Analysis, Saliency Maps and Integrated Gradients.

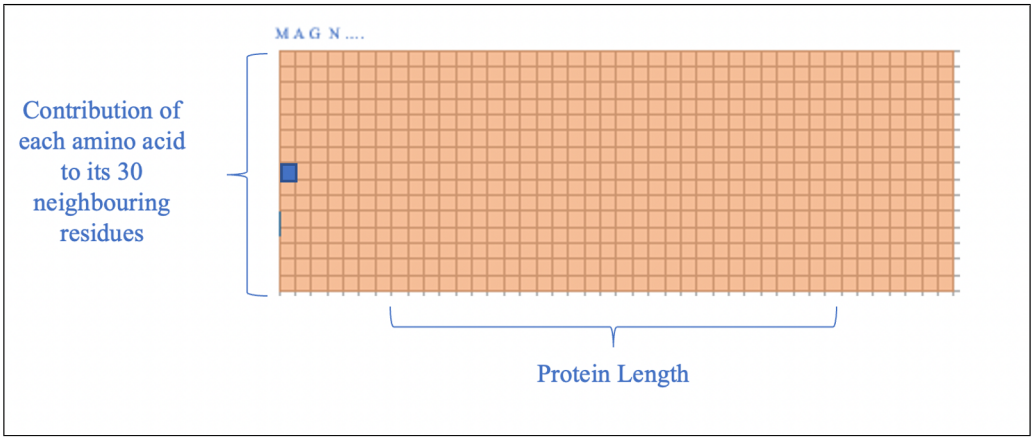

Figure 3.7: General structure of the visualization obtained from Sensitivity Analysis

3.3.2 Saliency Map

We want to visualize the importance of each residue of the protein sequence to its model's prediction. Simonyan et al. [31] introduced the concept of saliency map for images. The simplest definition of saliency map is the point wise multiplication between the model's gradient to its input. For biological applications Zou et al. [37], Lanchantin et al. [20] adapted this method to explain their deep neural network prediction. The term 'saliency' means unique features of an input sequence. To compute a saliency map mathematically we calculate first order Taylor expansion which explains the decision of a model by decomposing the activation function from a neural network. For our application, let us take an input protein sequence window P_0, which belongs to a particular class c and the model gives a score function $f_c(P)$. We know for a deep neural network this is a highly non-linear function hence we can not see the influence of the residues without approximating it. Hence, for some protein sequence window P_0, we can approximate $f_c(P)$ with a linear function by first order Taylor expansion as shown in the equation 3.2,

$$f_c(P) = \sum_{i=1}^{l} w_i p_i + b \tag{3.2}$$

Where w can be defined as the derivative of the function f_c with respect to P at the point P_0 and b is the bias of the model. We can write w as,

$$w = \left.\frac{\partial f_c}{\partial P}\right|_{P_0} \tag{3.3}$$

We perform point-wise multiplication of this derivative with the input features to get a output of shape n*31 where n is the feature dimension which is 39 for DELPHI and 73 for DLPred. Then we sum up the n values to obtain an array of size 31, normalized between 0 to 1, which tells the relevance score for each residue in that window. We use *innvestigate* python library to implement this method. Below is the general code snippet of the implementation.

```
import innvestigate
from keras.models import load_model
```

```
model = load_model('DELPHI.h5')
analyzer = innvestigate.create_analyzer("input_t_gradient", model)
analysis = analyzer.analyze(test_feature)
```

Listing 3.2: Code snippet of saliency map

Figure 3.8 shows a classic visualization of saliency map when we apply them on a convolutional model for digit classification using MNIST dataset. Unlike sensitivity analysis, saliency map visualization contains a large number of negative relevance scores coloured with lighter shades of red, and the important pixels which make the digit belong to the class are coloured with darker shades of red. This is because we took the square value of the gradient while computing sensitivity analysis, but here we compute the gradient and do point-wise multiplication with the input hence we have the negative values in the visualization which shows the negative relevance.

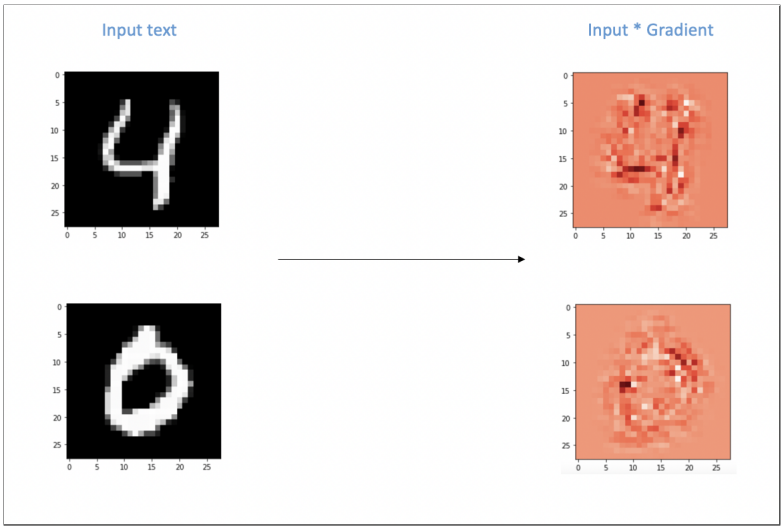

Figure 3.8: Input*Gradient for MNIST digit classifier

We can see from the Figure 3.9 that we are performing a point-wise multiplication of the gradient with the input feature followed by the summation of each 39 values gives us the saliency score for each residue in that window. The structure of the visualization obtained from the saliency map is the same as Figure 3.7.

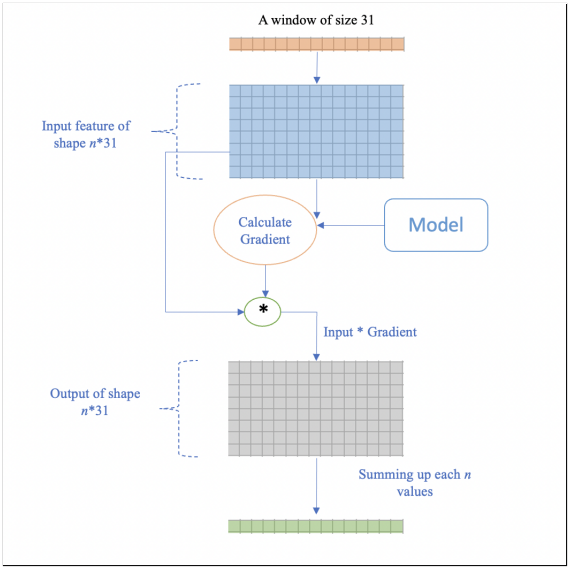

Figure 3.9: Flow diagram of saliency map for a window.
n denotes the feature dimension which is 31 and 73 for DELPHI and DLPred respectively.

3.3.3 Integrated Gradients

Integrated gradients (IG) [35] is another method which can explain the relationship between a deep learning model's prediction to its input. This method is also based on the gradient of the model's output with the input, but it does not follow the classical gradient calculation approach. This method has been widely used as an explanation technique in the applications like image recognition, question classification, neural machine translation, chemical model analysis etc. But in biological application specifically for proteins it is yet to be implemented. The steps of computing IG are as follows :

1. We identify the input and output of the model. In this case, input is a protein sequence of shape L*n*31 where L is the protein length, n is feature dimension and output is the model's prediction which is the output of last layer.

2. We select a baseline input which is a randomly initialized matrix of the same shape as the model input.

3. Then we interpolate the baseline input for some number of steps. Authors suggest to set the steps somewhere between 20 to 300. This step is the measure for the gradient approximation for an input sequence.

4. We get the gradients from this interpolated input and then approximate the gradient using Riemman approximation.

Mathematically IG can be defined as,

$$IG_i^{approx}(x) ::= (x_i - x_i') * \sum_{k=1}^{m} \frac{\partial(F(x' + \alpha * (x - x'))}{\partial x_i} * \frac{1}{m} \tag{3.4}$$

where x is the model input, x' is the baseline input, α is the interpolation constant which is denoted as $\frac{k}{m}$, m is the hyperparameter which imply the number of steps for gradient approximation, $(x_i - x_i')$ denotes the difference between the original input and baseline input. For our application, we choose a matrix of zero as a baseline input which is of the same shape of a input window, $n * 31$, where n is the dimension of the feature. Then we interpolate that input with steps 50, and compute gradient for the interpolated input with the aforementioned steps. To obtain the visualization from the integrated gradients, we follow the same structure as discussed in Section 3.2.1, Figure 3.7, where each column refers to the contribution of that amino acid to its neighbouring residues.

We have used the same python package *innvestigate* to implement integrated gradients with our keras model. Below is the code snippet of the implementation.

```
import innvestigate
from keras.models import load_model

model = load_model('DELPHI.h5')
baseline_ip = np.zeros(test_all_features_np3D[0:1].shape)
analyzer = innvestigate.create_analyzer("integrated_gradients",
           model,reference_inputs=baseline_ip)
analysis = analyzer.analyze(test_all_features_np3D[0:1])
```

Listing 3.3: Code snippet of integrated gradients

3.3.4 LIME : Local Interpretable Model-Agnostic Explanations

LIME [28] is a novel model-interpretability technique which is used to explain the prediction of any classifier in an interpretable manner. Most of the available deep learning model interpretability techniques are dependent on the architecture of the model, hence they are not well compatible with all the applications. However, LIME is a model-agnostic technique which treat the model as 'black-box' as shown in Figure 3.10. Since LIME works as 'local explainer', it focuses on explaining the model's behaviour against individual instances. The idea behind LIME is to perturb the input sample and observe the impact on the model's behaviour during prediction. When we apply LIME on any application, it gives us the result of how important each feature is while making prediction for that sample. Mathematically the explanation of a datapoint x from LIME can be defined by the equation:

$$\xi(x) = \arg\min_{g \in G} L(f, g, \Pi_x) + \Omega(g) \tag{3.5}$$

where x is the original feature, f is the model used for explanation and $f(x)$ is the probability that x belongs to certain class, Π_x stands for the promiximity measure that indicates how large the neighbourhood of the instance x is, g is the explanation model where $g \in G$ and G is the class of potential interpretable models like decision trees, linear models etc., $\Omega(g)$ is the level of complexity of interpretation for a model. For example, for a decision tree $\Omega(g)$ may be the depth of the tree or if we consider linear explanation models it can be the number of non-zero weights. The model for explanation the instance x is g. It minimizes the loss L which

denotes the accuracy of the explanations to the prediction. The value of $\Omega(g)$ is low for ease of interpretation.

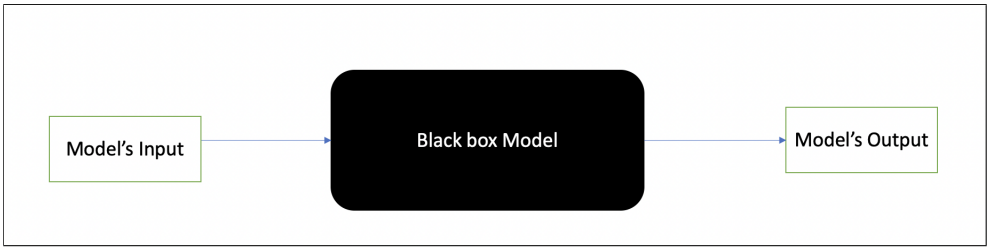

Figure 3.10: Intuition behind LIME
Figure idea taken from: towardsdatascience.com

LIME can be applied on different applications such as: texts, images and tabular data. For DELPHI, each input window is a shape of 39*31, where 39 is the dimension of features and 31 is the number of residue in that window which is known as sample in machine learning. For a tabular data, LIME considers each row in the tabular format as a sample and each column as a feature. To use this method, we tweak the model architecture a bit because a test protein sequence input is a 3D shape. Let us assume a protein sequence of length L. So, the shape of the input to DELPHI is L*39*31. We flatten this input to get an input shape of L*1209. We pass this input to the model, and in the next layer we reshape this input again to original 3D shape. We also double check the performance of the model after this modification, and it is unaffected as expected. When we run LIME for a single window of a sample protein sequence we get 31 explanation scores for each 39 feature. We take the maximum value from each 31 scores which is considered as the importance score for that particular feature in that specific window. DELPHI model consists of 12 features among which some features like PSSM, Physical properties, Physiochemcial characteristics have dimension of 20, 7 and 3 respectively. Again, we take the maximum value from these each multidimensional features which is the final importance score for that particular feature.

Python environment gives the package 'LIME' which we can use while implementation of the method. We can install it by simply using *pip install LIME*. We create the visualizations from LIME using matplotlib library. Below is the code snippet of how we can implement LIME in python :

```
import numpy as np
import lime
import lime.lime_tabular
from keras.model import load_model

model=load_model('DELPHI.h5')
explainer = lime.lime_tabular.LimeTabularExplainer(train_data)
exp = explainer.explain_instance(test[i],model.predict,
labels=(1,))
exp.as_pyplot_figure(label=1)
```

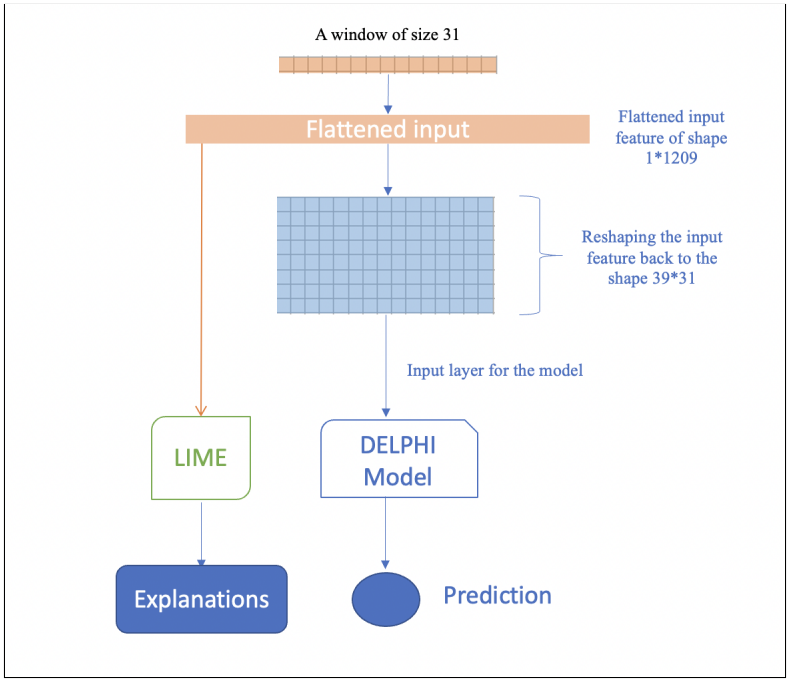

Figure 3.11: LIME on DELPHI

Listing 3.4: Code snippet of LIME implementation

Figure 3.11 explains the input data structure of LIME and original model. Both LIME and DELPHI model take the same input data for explanation and prediction respectively, but in a different shape. We also check the biological importance of the features and compare them with the explanation obtained from LIME. From our experimental result we observe that they are the same, which makes LIME explanation relevant for our application.

Chapter 4

Experimental Results

4.1 Operation Environment

Our entire experiment is implemented in python. We have used our computing cluster dusky in SHARCNET (Shared Hierarchical Academic Research Computing Network) for the computations. We have also used Google Colab platform and graham cluster of SHARCNET occasionally to work with GPU based models. The configurations of dusky are described below:

- Processor : 32 cores
- RAM : 1000 GB
- Operating System : CentOS 6.3

We have created python virtual environment in the clusters where we install all the required packages. We have created a file Requirements.txt which stores all the packages needed for our application. Virtual environment can be created in the following way:

```
#load the python module
module load python/3.5

#create the virtual environment
virtualenv ~/[Environment_name]

#activate the virtual environment
source ~/[Environment_name]/bin/activate

#Install python packages
pip install [python_package]
```

Computation of sensitivity analysis, saliency map and integrated gradients are very fast as they all compute gradients of the output with respect to model's input in different ways, and they are just one step backpropagation. But as we have L sliding windows for our input, where L is the protein length, the computation time is directly proportional to the protein length. For

LIME in tabular data, we need some good amount of memory as it requires to compute the statistics of each column (features) from the training data, and our training input consists of approximately 9000 protein sequences.

4.2 Experimental Setup

To generate visualizations from sensitivity analysis, saliency map and integrated gradients we have used python package called *innvestigate*. Python already provides LIME as a separate package, so we used that while generating explanations for the prediction. These packages can be installed in the virtual environment in the following way:

```
#installation of the package innvestigate
pip install innvestigate

#installation of the package LIME
pip install LIME
```

4.3 Visualization of the Proteins

While producing the visualization we chose several proteins for which DELPHI achieves the highest precision and accuracy. The formulae to calculate these metrics are as follows:

$$Precision = \frac{TP}{TP + FP}$$

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

Here, true positives (TP) and true negatives (TN) denote the correctly classified binding and non-binding residues. Whereas, false positives (FP) and false negatives (FN) are incorrectly identified binding and non-binding residues respectively. We aim to produce visualizations for the proteins from both DELPHI and DLPred and focus on the different aspects of model's learning procedure.

Now we present the visualizations for selected proteins with high accuracy and precision. For a single visualization we have two parts, the left side is produced for the DELPHI model and the right side is for the DLPred model. In every visualization we add the model's prediction graph at the top to observe the model's behaviour between binding and non-binding residues. Prediction graphs show the model's prediction for each amino acid in the protein sequence. When the amino acid is predicted to be a binding residue the prediction score is high, and vice versa.

We show next our visualizations, each represented as a (31*n) matrix where column *i* gives the influence of the residue at position *i* on the predictions corresponding to the sorrounding 31 residues. (The residues at the beginning and at the end of the proteins have fewer residues to impact, which explains the missing corners of the matrices.) For visual appeal, we plot the values as a heatmap, with larger positive values in darker red and lower negative values in darker blue. Darker red means stronger influence in favour of prediction, whereas darker blue is working against the prediction. We have two different types of visualizations for saliency map and integrated gradients. In the visualizations with the shades of red, we clipped the negative relevance scores to zero, so we only focus on the residue which reacts positively for the model's prediction. At the bottom of every visualization we add two plots with a shade of red-blue, where shades of blue indicates the negative relevance for the residues. The order of the visualizations from top to bottom is as follows: sensitivity analysis, saliency map, integrated gradients, saliency map with negative relevance and integrated gradients with negative relevance.

We observe from the visualizations obtained by sensitivity analysis that most of the amino acids are influential to their own prediction only, whereas in the other two methods some dark red columns are generated. This indicates that some residues have a much greater influence to their neighbouring residues. In the bottom two visualizations we see a large number of negative relevance is present in the shade of blues which indicates that those amino acids are not influential to the model. Hence it is evident that few positions of the amino acid residue act as an informative section for the model. Figure 4.1 - 4.4 demonstrate the visualizations created for proteins: `3MP7B`, `O68692`, `Q81R67`, `Q06253` respectively.

To achieve a more clear picture about the program's behaviour for two softwares, we compute the correlation for each visualization between DELPHI and DLPred. We have used Pearson's correlation coefficient metric to check the relationship between the data obtained by each method for DELPHI and DLPred. This can be defined as the covariance between the two variables divided by the product of standard deviation between the data samples. The output of the coefficient lies from the range of -1 to 1, where any value which is below -0.5 or above 0.5 is considered as a notable correlation. We have used the function *pearsonr* from python library *scipy* to implement this.

Figures	Sensivity analysis	Saliency map	Integrated gradients(IG)	Saliency map with negative relevance	IG with negative relevance
Figure 4.1	0.412	0.219	0.349	0.087	0.2
Figure 4.2	0.179	0.138	0.408	0.033	0.328
Figure 4.3	0.109	0.134	0.281	0.022	0.179
Figure 4.4	0.228	0.116	0.31	0.011	0.188

Table 4.1: Correlation between the visualizations

Table 4.1 shows the Pearson's correlation coefficient value between the matrices corresponding to each of our five visualizations. We can see there is very weak, or no, positive correlation between them. So, this is an indication that each program seems to be using the information from the input sequences differently. This information is important, as it may indicate room for future improvement in terms of interpreting biological applications.

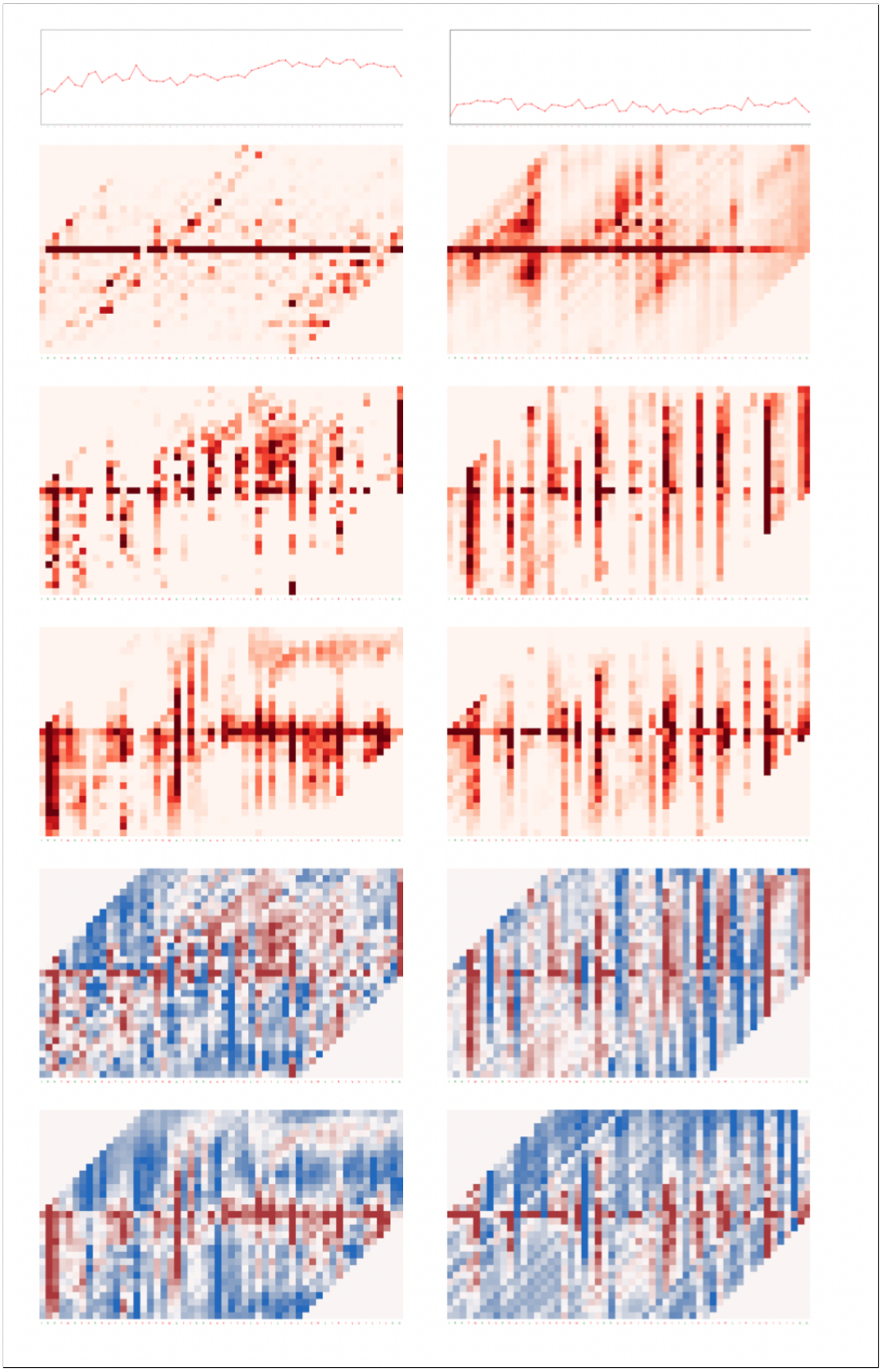

Figure 4.1: Protein 3MP7B: DELPHI(left),DLPred(right)

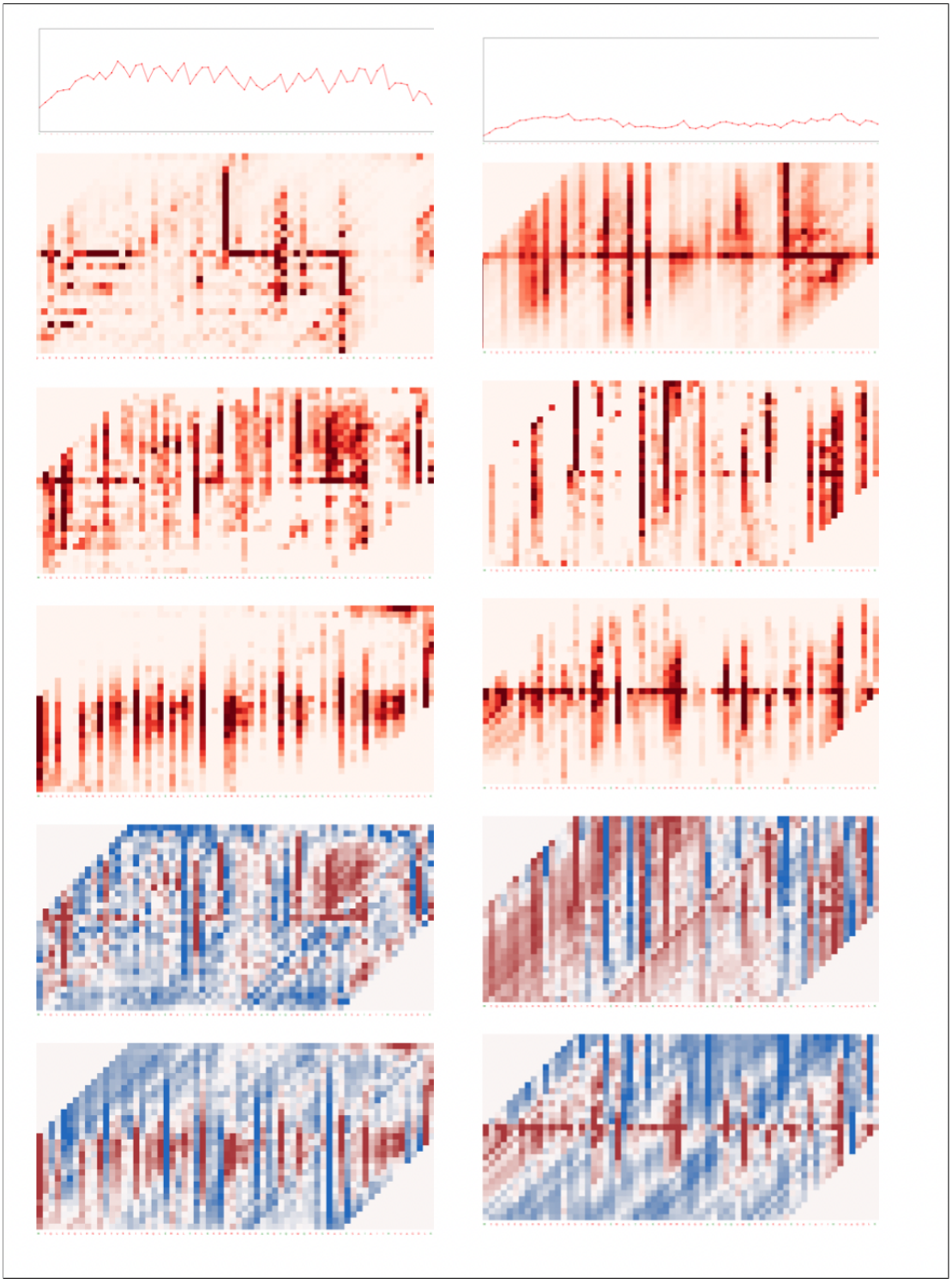

Figure 4.2: Protein O68692: DELPHI(left),DLPred(right)

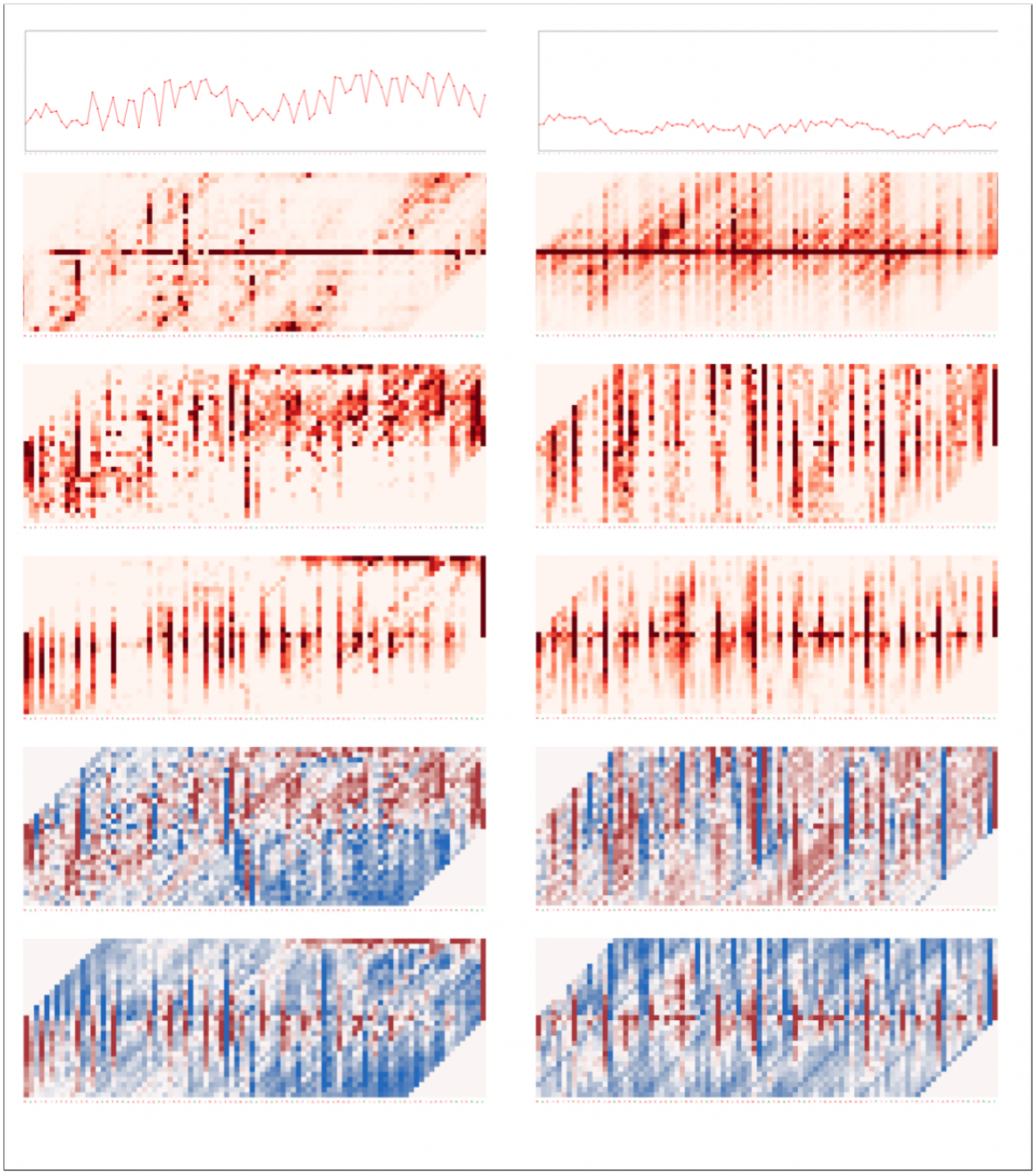

Figure 4.3: Protein Q81R67: DELPHI(left),DLPred(right)

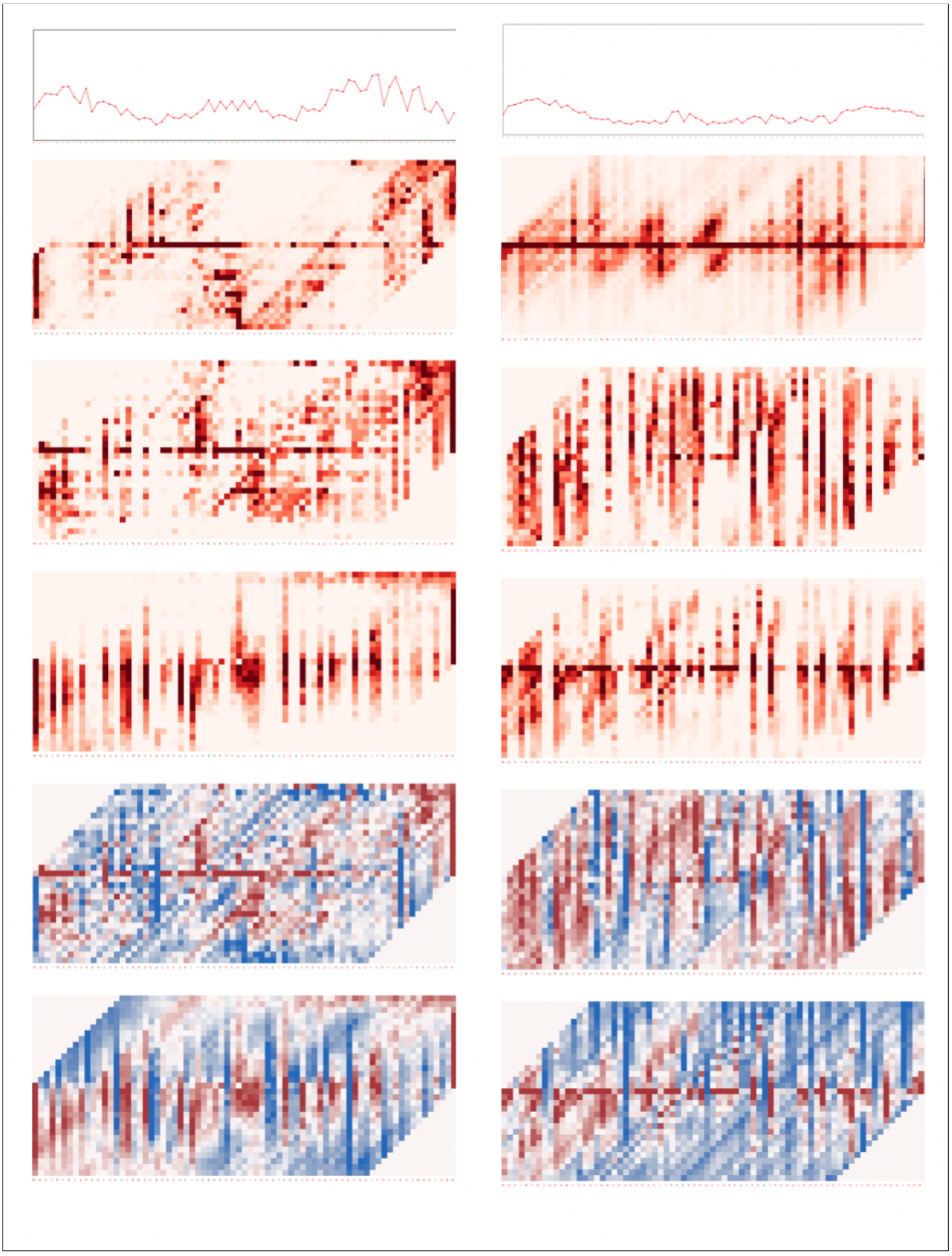

Figure 4.4: Protein Q06253: DELPHI(left),DLPred(right)

4.4 Impact by Position

Since both models are designed on sliding window protocol, we want to see how the positions in the window impact the prediction. We have a sliding window of size $2k + 1$, where k is 15. So, for any i, where $-k \leq i \leq +k$, we want to measure the impact of position i on the prediction. We get a distribution from three visualization methods: sensitivity analysis, saliency map and integrated gradients from both models. In each plot, the impact of position i on the prediction for centered position is normalized between 0 to 1. Figures 4.5 - 4.6 represent the boxplot created from sensitivity analysis data; Figures 4.7 - 4.8 show the boxplots created from saliency map data; Figures 4.9 - 4.10 show the boxplots generated from integrated gradients data.

We observe from the boxplots below that the impact of position 15 on the prediction is the highest for both models. Whereas, the farthest positions in the window do not make much impact. But the nearby regions of the center point contribute well to the prediction of the centered residue. From the sensitivity analysis plots, we can see only the middle position of a window contributes significantly for both the softwares whereas in saliency map and integrated gradients the contribution varies throughout the positions, but the middle one remains the highest. These plots give us an idea about the important positions in the sliding window.

To understand the relationship between the boxplots, we measure Pearson correlation between the medians in the plots below for DELPHI and DLPred. Table 4.2 demonstrates the result for the three methods, where we can see strong correlation among the two data samples. This finding is encouraging as it indicates that the behaviour of impact by position between both models is similar. This is in contrast with the lack of correlation between data for individual proteins, as seen in the previous section. The results in the previous section indicate that the two models behave very differently. This makes the strong correlation from Table 4.2 all the more interesting, as it seems to reveal intrinsic features of the proteins and their interactions, rather than of the models.

Sensitivity analysis	Saliency map	Integrated gradients
0.958	0.643	0.788

Table 4.2: Correlation between the medians in the boxplots

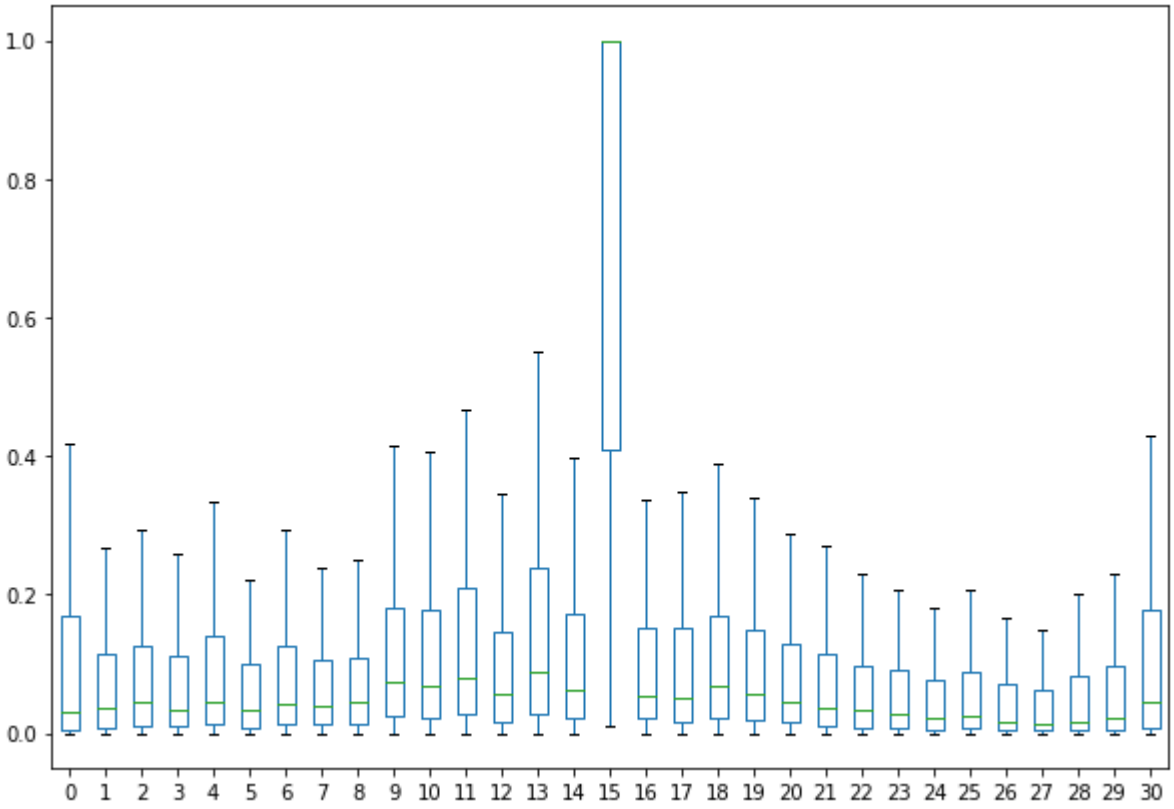

Figure 4.5: Box plot from Sensitivity Analysis: DELPHI

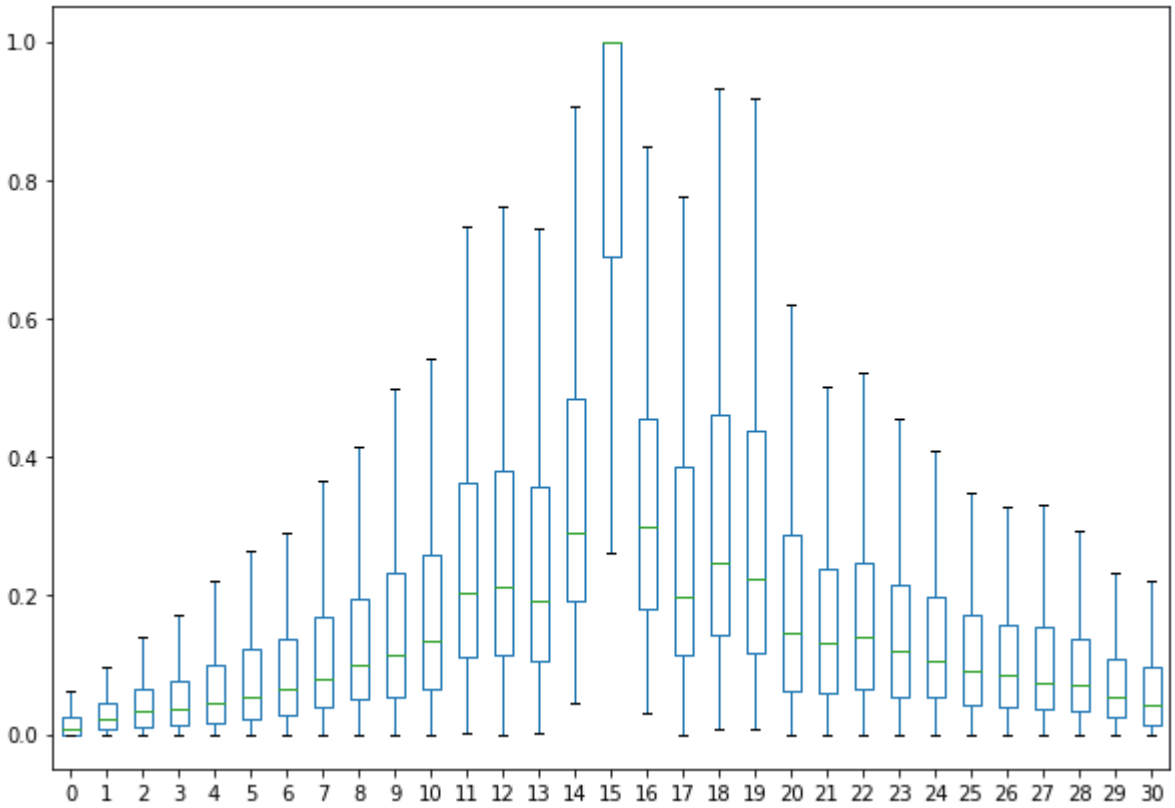

Figure 4.6: Box plot from Sensitivity Analysis: DLPred

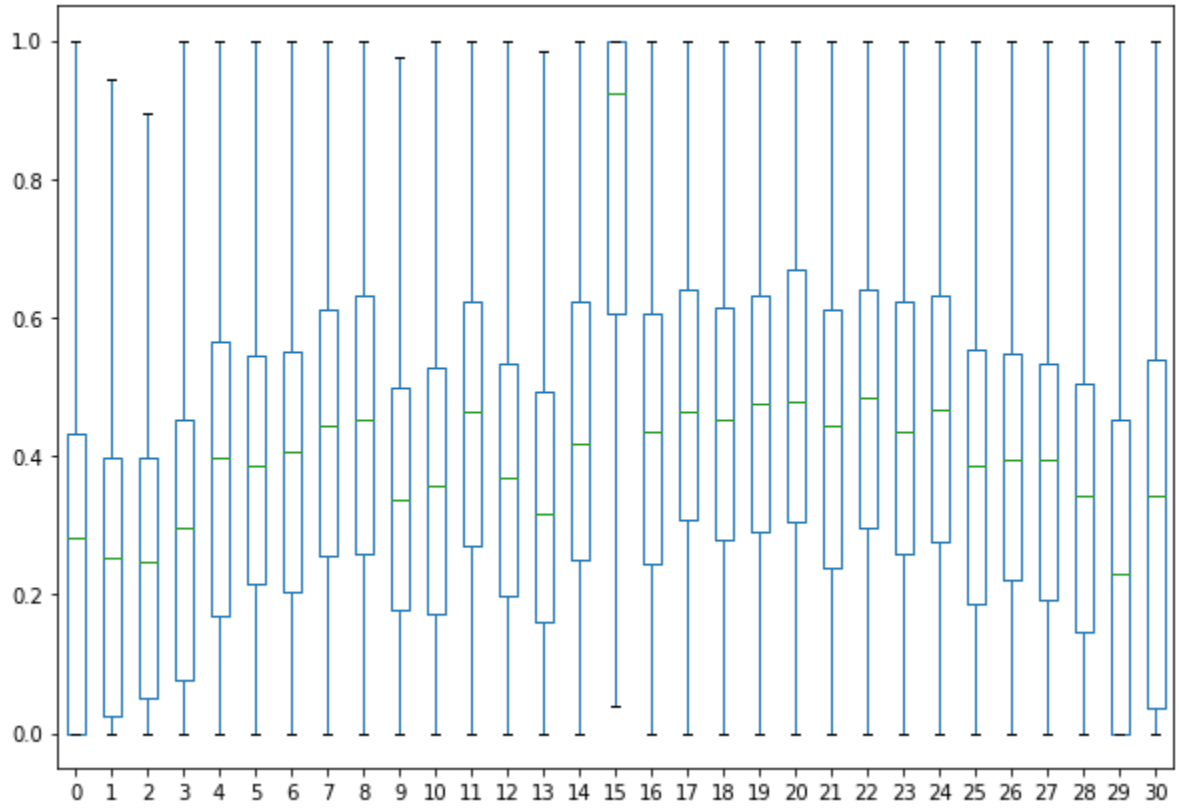

Figure 4.7: Box plot from Saliency Map: DELPHI

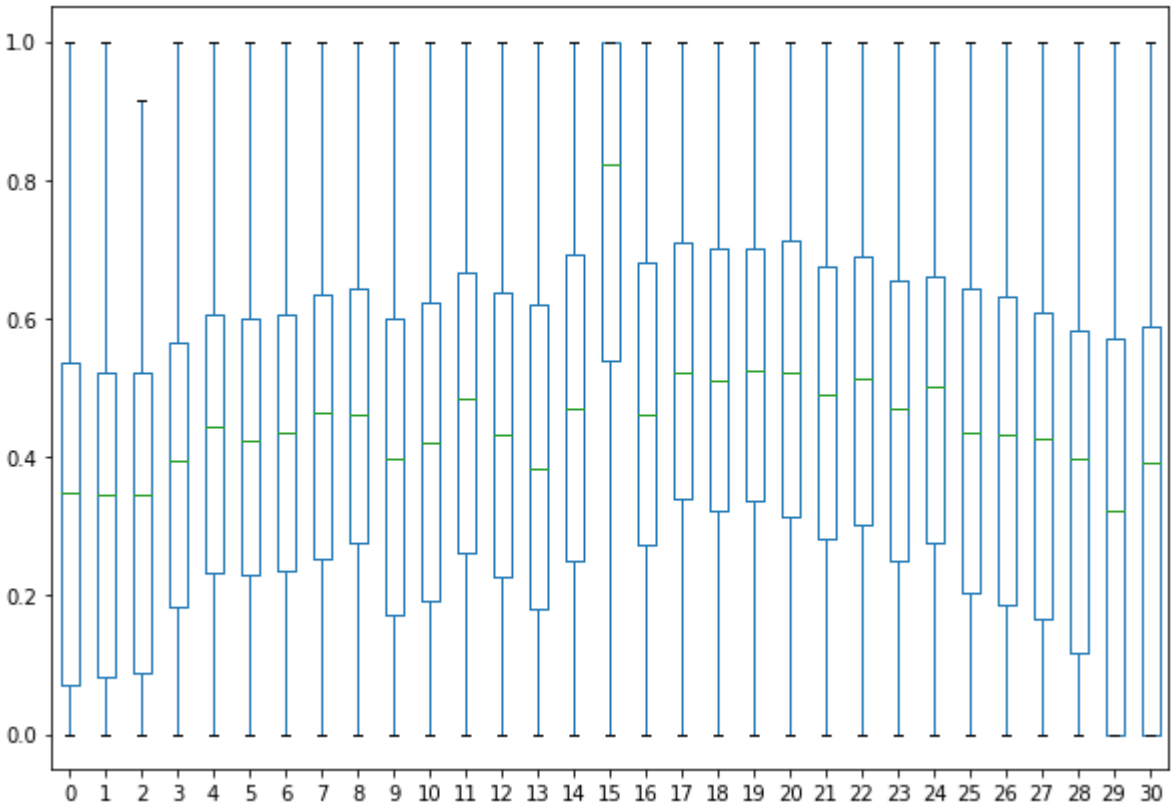

Figure 4.8: Box plot from Saliency Map: DLPred

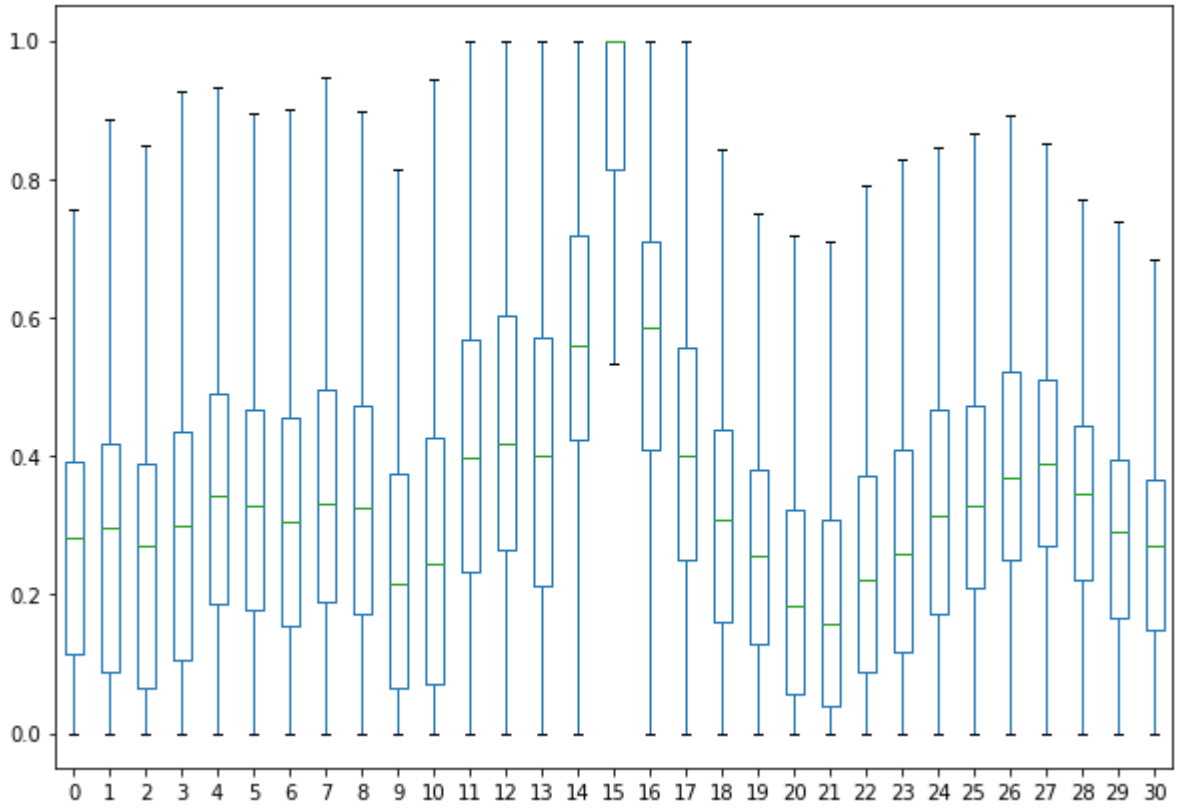

Figure 4.9: Box plot from Integrated Gradients: DELPHI

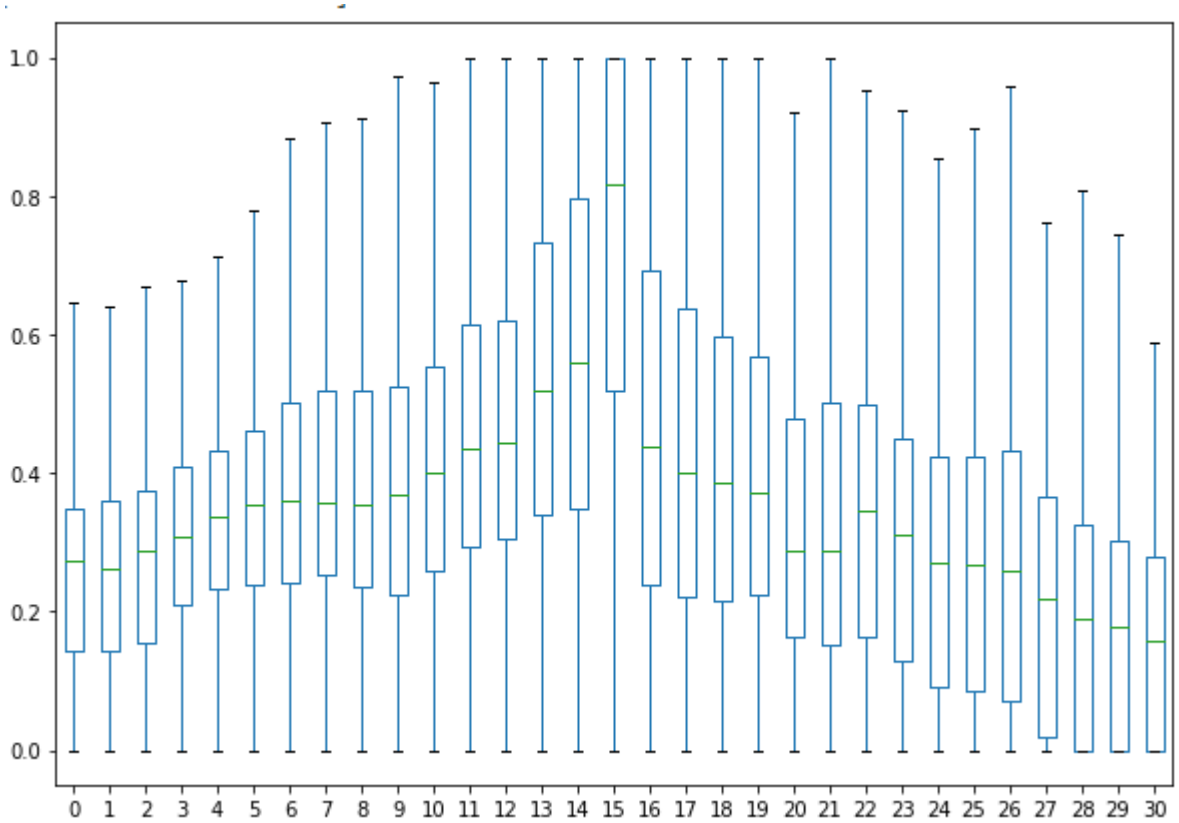

Figure 4.10: Box plot from Integrated Gradients: DLPred

4.5 LIME Identified Features

We apply LIME to DELPHI to identify the feature importance of a protein sequence for positive class prediction. We have done our experiment with several input sequences, and all of them show the similar kind of result, that position-specific scoring matrix (PSSM) is the most influential feature which is helping the model to predict positive class. PSSM matrices have been used widely in protein interaction problems. PSI-BLAST [9] computes these matrices. It contains evolutionary conservation for each amino acid position by aligning an input sequence with protein databases. Below is a sample protein sequence in fasta format for which we are presenting LIME explanations :

```
>1ml0D
AVTCCYNFTNRKISVQRLASYRRITSSKCPKEAVIFKTIVAKEICADPKQKWVQDSIDHLDKQT
11111111100000000000000000010110001000000011100000000000000000000
```

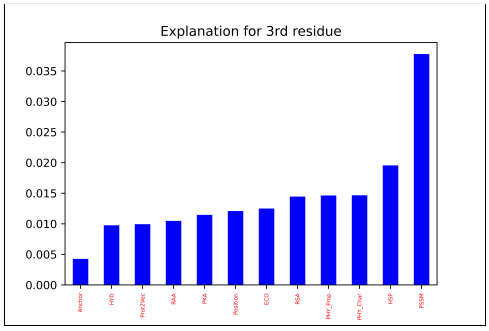

Figure 4.11: LIME explanation for 3rd residue *T*

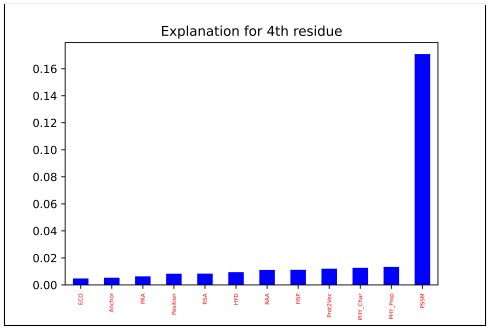

Figure 4.12: LIME explanation for 4th residue *C*

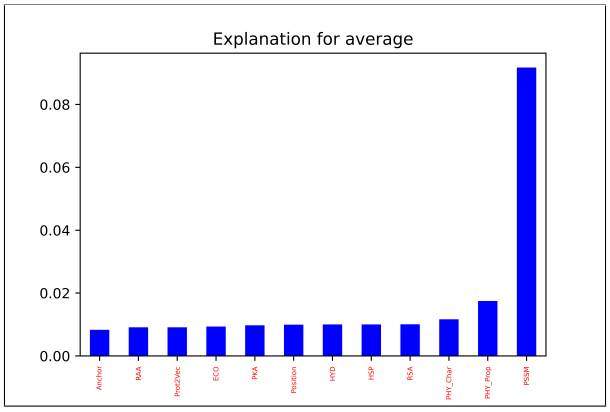

Figure 4.13: LIME explanation calculated from the average scores from all the positive classes

Figure 4.11 and 4.12 shows the LIME explanation from the protein sequence `1ml0D`. We present the explanation made for third and fourth residue from the protein sequence respectively, where we can see in the bar graph that the score for PSSM is very high compare to other features which imply that this is the most significant one for model's prediction. Also,

we calculate the average of LIME scores for each feature from all the positive classes to get conclusive evidence of our finding as shown in Figure 4.13. It is also biologically relevant as PSSM is the most important features to represent a protein for a protein protein interaction (PPI) task.

Chapter 5

Conclusion and Future Works

In this chapter, we summarize our work and discuss possible future research to achieve more clarity in the interpretation of deep learning models for protein interaction.

5.1 Summary

In this thesis, we have adapted existing deep learning interpretation methods in two protein problems, which is according to our best knowledge is a novel approach. We have discussed various existing methods, their limitations and further we have implemented three gradient based methods to investigate the influence of amino acid residues to protein interactions. Also, we have implemented Local Interpretable Model-Agnostic Explanations (LIME) to see the feature importance for the residues with positive class. From the visualizations of both models, we identify there are few important amino acid residues present for the proteins which are influential to the model. We calculate Pearson correlation between the visualizations for both models to understand how they behave for different software. We find that there is very little correlation, so each program seems to be using the information differently. As both models are implemented using a sliding window protocol, we want to see how the positions in the window influence the model's prediction. We observed from all three methods, that the impact of the middle position in the window, for which is prediction is made, is the highest. Whereas for saliency map and integrated gradients the neighbouring regions also contribute well. When we check the correlation between medians in the plots, it shows us a high positive correlation between the two softwares. From the experiment with LIME to DELPHI, we found that position-specific scoring matrix (PSSM) is identified as the most influential feature for the model among the 12 features.

5.2 Conclusion

This is the first work on deep learning interpretation for protein sequences. The contributions of the thesis are as follows: First, the dark red columns in the visualization matrices indicate clearly that some residues have a much greater influence on the interaction properties of nearby positions than others. Second, the influence of each residue on nearby positions seems to be well correlated with the distance between positions, as indicated by the correlation between

medians of the box plots in Figures 4.5 - 4.10. Third, we identify the position-specific scoring matrix (PSSM) as the most important feature for protein interactions. These findings open a new area of investigation to understand the above mentioned phenomena for protein-protein interaction binding sites prediction, with implications for two fundamental problems: Improving protein interaction site prediction through better understanding of interaction relationship between residues, and improving biological understanding of the interactions themselves.

5.3 Future Works

As this is a very young research area, there is scope for future improvement. Firstly, we can develop some newer interpretation methods to work with the biological problems. Most of the existing methods are not well compatible with all the neural networks, more specifically, they work well with convolutional neural networks, such as images. As per our research, there is a scope to develop new methods which can work with any model irrespective of domain. Although there are model agnostic methods available like LIME, they are still dependant on some particular data structures and they generate explanations locally. If we can come up with some methods which can explain a model's behaviour in general that would be great. Also, from the correlation of the visualizations between the two softwares is very weak, this also may indicate room for improvement.

Secondly, we think for biological applications a lot more understanding of the data is needed to understand the model's explanation. Because for applications like images, texts human can interpret the important parts, for biology it is not the same. So, if we can improve this area of understanding, it will be easier for the researchers to develop and generate new methodology to interpret biological problems.

Bibliography

[1] 3D structure of a protein. https://www.bnl.gov/newsroom/news.php?a=213084.

[2] Central dogma of molecular biology. http://www.cs.ukzn.ac.za/ hughm/bio/docs/IntroTo-BioinfAlgorithms.pdf.

[3] Different protein structures. https://www.slideshare.net/SabahatAli9/protein-structure-levels.

[4] Digit classification. https://github.com/vneogi199/Handwritten-Digit-Recognition-Using-Random-Forest.

[5] Double helix structure of DNA. https://www.genome.gov/genetics-glossary/Double-Helix.

[6] Protein-protein interaction. https://www.creative-proteomics.com/services/protein-protein-interaction-networks.htm.

[7] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016.

[8] Maximilian Alber, Sebastian Lapuschkin, Philipp Seegerer, Miriam Hägele, Kristof T. Schütt, Grégoire Montavon, Wojciech Samek, Klaus-Robert Müller, Sven Dähne, and Pieter-Jan Kindermans. innvestigate neural networks! *Journal of Machine Learning Research*, 20(93):1–8, 2019.

[9] Stephen F Altschul, Thomas L Madden, Alejandro A Schäffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J Lipman. Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic acids research*, 25(17):3389–3402, 1997.

[10] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140, 2015.

[11] François Chollet et al. Keras: The python deep learning library. *Astrophysics Source Code Library*, pages ascl–1806, 2018.

[12] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[13] UniProt Consortium. Uniprot: a hub for protein information. *Nucleic acids research*, 43(D1):D204–D212, 2015.

[14] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[16] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[17] Andrej Karpathy, Justin Johnson, and Li Fei-Fei. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015.

[18] Pieter-Jan Kindermans, Kristof T Schütt, Maximilian Alber, Klaus-Robert Müller, Dumitru Erhan, Been Kim, and Sven Dähne. Learning how to explain neural networks: Patternnet and patternattribution. *arXiv preprint arXiv:1705.05598*, 2017.

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[20] Jack Lanchantin, Ritambhara Singh, Beilun Wang, and Yanjun Qi. Deep motif dashboard: Visualizing and understanding genomic sequences using deep neural networks. In *Pacific Symposium on Biocomputing 2017*, pages 254–265. World Scientific, 2017.

[21] Haoyang Li, Shuye Tian, Yu Li, Qiming Fang, Renbo Tan, Yijie Pan, Chao Huang, Ying Xu, and Xin Gao. Modern deep learning in bioinformatics. *Journal of molecular cell biology*, 2020.

[22] Yiwei Li, G Brian Golding, and Lucian Ilie. DELPHI: accurate deep ensemble model for protein interaction sites prediction. *Bioinformatics*, 08 2020. btaa750.

[23] Yao Ming, Shaozu Cao, Ruixiang Zhang, Zhen Li, Yuanzhe Chen, Yangqiu Song, and Huamin Qu. Understanding hidden memories of recurrent neural networks. In *2017 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pages 13–24. IEEE, 2017.

[24] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318, 2013.

[25] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, pages 8026–8037, 2019.

[26] Sylvain Pitre, Md Alamgir, James R Green, Michel Dumontier, Frank Dehne, and Ashkan Golshani. Computational methods for predicting protein–protein interactions. In *Protein–Protein Interaction*, pages 247–267. Springer, 2008.

[27] Christelle Pommié, Séverine Levadoux, Robert Sabatier, Gérard Lefranc, and Marie-Paule Lefranc. Imgt standardized criteria for statistical analysis of immunoglobulin v-region amino acid properties. *Journal of Molecular Recognition*, 17(1):17–32, 2004.

[28] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Model-agnostic interpretability of machine learning. *arXiv preprint arXiv:1606.05386*, 2016.

[29] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 1135–1144, 2016.

[30] Avanti Shrikumar, Peyton Greenside, Anna Shcherbina, and Anshul Kundaje. Not just a black box: Learning important features through propagating activation differences. *arXiv preprint arXiv:1605.01713*, 2016.

[31] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[32] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[33] 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.

[34] Hendrik Strobelt, Sebastian Gehrmann, Hanspeter Pfister, and Alexander M Rush. Lstmvis: A tool for visual analysis of hidden state dynamics in recurrent neural networks. *IEEE transactions on visualization and computer graphics*, 24(1):667–676, 2017.

[35] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. *arXiv preprint arXiv:1703.01365*, 2017.

[36] Buzhong Zhang, Jinyan Li, Lijun Quan, Yu Chen, and Qiang Lü. Sequence-based prediction of protein-protein interaction sites by simplified long short-term memory network. *Neurocomputing*, 357:86–100, 2019.

[37] James Zou, Mikael Huss, Abubakar Abid, Pejman Mohammadi, Ali Torkamani, and Amalio Telenti. A primer on deep learning in genomics. *Nature genetics*, 51(1):12–18, 2019.

Curriculum Vitae

Name:	Dipanjan Chatterjee
Post-Secondary Education and Degrees:	West Bengal University of Technology Kolkata, India 2011 - 2015 B.Tech in CS University of Western Ontario London, ON 2019 - 2021 M.Sc.
Honours and Awards:	Western Graduate Research Scholarship 2019-2020
Related Work Experience:	Graduate Teaching Assistant The University of Western Ontario 2019 - 2020 Graduate Research Assistant The University of Western Ontario 2019-2021 Programmer Analyst Cognizant Technology Solutions 2018 - 2019 Senior Systems Engineer Infosys Limited 2015-2018