University of Glasgow | School of Computing Science

Honours Individual Project Dissertation

# Creating a server for protein disorder prediction based on deep neural networks

**Ryan Gregson**

March 24, 2023

# Abstract

Protein disorder prediction is an important problem in the field of bioinformatics as it provides an insight into the function and structure of proteins. In this project we developed multiple deep neural networks to predict intrinsic disorder within protein sequences. We assessed different methods for feature encoding and implemented convolutional and recurrent neural networks.

We found our neural networks could identify intrinsic disorder. We discovered that CNNs using 1D convolutions and long short-term memory (LSTM) RNN architectures performed similarly to each other. These archictures were both significantly better than the other simpler CNN using 2D convolutions. In addition, we found position-specific scoring matrix (PSSM) feature input significantly improves prediction accuracy compared to a one-hot encoding approach.

We evaluate our models against state-of-the-art methods using the CASP10 benchmark experiment, demonstrating our models lacked the capability to identify disorder precisely. This was apparent via the result of a high false positive rate. We acknowledge this as a limitation of our study and believe further experimentation with our models will be useful to address this issue.

Furthermore, we developed a web server that uses our trained neural network, providing bioscience researchers with a user-friendly tool for analysing disorder in protein sequences. This web server can be run locally; accessed from: `https://github.com/ryangregson01/L4-dissertation`.

# Acknowledgements

I would like to thank my supervisor, Dr. Kevin Bryson, for the guidance, support, and encouragement given throughout the entire project.

# Education Use Consent

# Contents

# 1 Introduction

## 1.1 Motivation

Intrinsically disordered proteins do not have a fixed or ordered three-dimensional structure and there are significant differences between the amino acid sequences of intrinsically disordered proteins (IDPs) / intrinsically disordered regions (IDRs) compared to structured globular (functional) proteins (Martinelli et al. 2019). Figure 1.1 demonstrates the difference between unstructured and structured proteins, where a solid ribbon-like representation is used to depict the well-defined three-dimensional structure, such as alpha helices and beta sheets, and disordered regions of the proteins are depicted using thinner strands, indicating the lack of fixed structure. The structural order of these sequences can be manually labelled from experiments with x-ray crystallographic analysis (Wikipedia contributors 2023*d*). These experiments can be difficult and the number of newly occurring proteins submitted to the Protein Data Bank (PDB) (Berman et al. 2000) each year makes it difficult to manually experiment with each one, meaning many proteins' disorder structure have not been assessed.

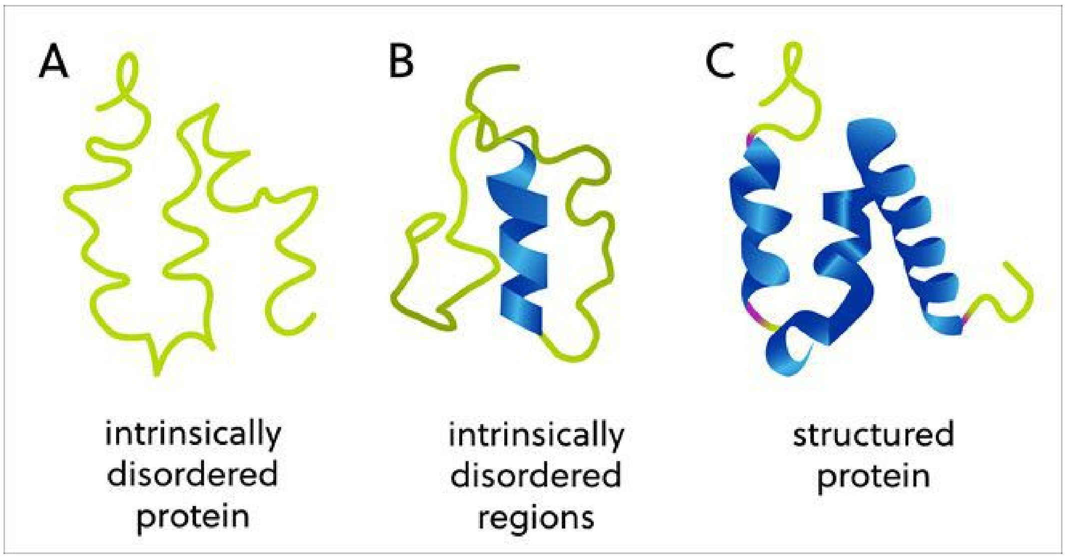

A
B
C
intrinsically disordered protein
intrinsically disordered regions
structured protein

***Figure 1.1:*** *An example of ordered and disordered proteins (Tenchov 2022). Proteins that are disordered do not have a defined or fixed structure in contrast to ordered proteins.*

We can use bioinformatic tools to label the disordered regions in these protein sequences by creating a model that can learn the differences between ordered and disordered amino acid sequences. With more labelled sequences, doctors and researchers can understand more about disordered protein sequences. Insights about these disordered regions helps drive research for the development of drugs that are applied to disordered proteins, understanding the functionality of these regions, and contributes to keeping a well-maintained knowledge database of proteins. It is important this research is supported as disordered proteins cause neurodegenerative diseases such as Alzheimer's, Parkinson's and Huntington's. Therefore, it is important drugs can be designed and applied to IDPs in an attempt to counteract these diseases. Furthermore, some disordered regions in proteins are vital for creating functionality in the protein as they can give elastic properties to proteins. These elastic properties are usually difficult to create with ordered protein chains. An example of this is the entropic spring in Titin, which is composed of both ordered and disordered regions (Morgan and Saleh 2017). This entropic spring allows muscle fibres to stretch and contract efficiently, making it an essential component of muscle function and demonstrating that IDPs can be useful within our bodies.

Deep learning approaches have become incredibly popular recently, and these approaches currently dominate classification tasks. Deep learning architectures are prevalent in classifying the IDRs in protein sequences, where state of the art predictors such as SPOT-DISORDER2 (Hanson et al. 2019) also have their own online protein disorder prediction servers for others to use their deep learning model.

## 1.2 General problem and aim

Protein sequences are made up of amino acids. The problem we are attempting to solve is to see if we can use these sequences to predict whether each amino acid is ordered or disordered, hence identifying the intrinsically disordered regions. We will produce deep neural networks which will take in protein sequences and predict the disordered regions within them.

The aim of this project is to explore how different deep learning approaches and architectures perform at predicting protein disorder. To assess these deep learning methods, we will:

- Discuss current solutions to protein disorder prediction (Section 2.3).
- Discuss our approach for creating a protein disorder classifier (Chapter 3).
- Design and implement different deep neural networks, such as a convolutional neural network (CNN) and recurrent neural network (RNN) (Chapter 4 and Chapter 5).
- Evaluate our implemented neural networks using standard metrics and the test datasets proposed by CASP (Moult et al. 1994), which have been used for evaluating protein disorder prediction models (Chapter 6).

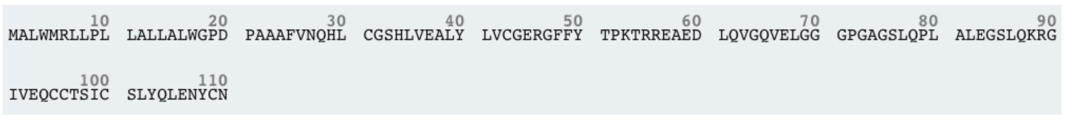
An aim alongside comparing the deep neural networks is to create a web server. This web server will let a user input a protein sequence via a web page form and visualise the predicted disordered regions. This will make the trained models accessible, which will result in bioscience researchers being able to interact with our developed solution easily.

# 2 Background

## 2.1 Biology overview

### 2.1.1 What is a protein?

Proteins are a linear sequence of amino acids (O'Connor et al. 2010). The Insulin protein has a precursor molecule with a linear sequence of 110 amino acids (Figure 2.1). This globular protein controls blood sugar levels, and diabetics need to inject human insulin to control their blood sugar.



```
        10         20         30         40         50         60         70         80         90
MALWMRLLPL LALLALWGPD PAAAFVNQHL CGSHLVEALY LVCGERGFFY TPKTRREAED LQVGQVELGG GPGAGSLQPL ALEGSLQKRG

       100        110
IVEQCCTSIC SLYQLENYCN
```

***Figure 2.1:*** *Human insulin amino acid sequence as presented by UniProt. (The UniProt Consortium 2023). Insulin was the first protein ever sequenced, obtained by Frederick Sanger in 1955.*

There are twenty possible amino acids that protein sequences can be built from, and each of these amino acids have a unique side chain which have different structural properties (Wikipedia contributors 2023*a*). Different proteins have different sequences of amino acids. The amino acids in the sequence will "interact with each other to form a well-defined, folded protein" (Cheriyedath 2019). The folded protein will have a three-dimensional structure, and this three-dimensional structure determines the protein's function (BBC 2023). This function can be supporting frameworks inside cells, catalysing biological reactions, communication between different parts of the body, and proteins are used for function inside the immune system (BBC 2023). Therefore, it is important that the folding of proteins is done correctly as misfolded proteins will lose their structure and function, such as the above insulin molecule, which can lead to diseases.

### 2.1.2 Intrinsically disordered proteins (IDPs)

IDPs are different to structured proteins. These unstructured, disordered proteins are suggested to cause a number of diseases. For example, disordered mutations of the functional insulin molecule can form amyloid structures. These structures can lead to amyloid disease, such as Alzheimer's, and Parkinson's. IDPs amino acid sequences differ and contain a "low content of bulky hydrophobic amino acids and a high proportion of polar and charged amino acids" (Wikipedia contributors 2023*d*). In comparison to structured proteins that fold correctly these sequences cannot fold into stable globular proteins, meaning that the protein lacks an ordered three-dimensional structure. The root cause of this lack of structure, causing disorder, is the proteins' primary structure: the amino acid sequence. When disordered amino acids are grouped, this is labelled as an intrinsically disordered region (IDR) and is where the misfolding occurs. Therefore, making predictions about the probability that an amino acid residue is ordered or disordered within a protein sequence will help determine where this incorrect folding will occur.

This means by focusing on the amino acids' structure we can determine the outcome of the protein functioning correctly.

A current problem is the lack of labelled data about the structure of protein sequences as determining these via experiments such as nuclear magnetic resonance (NMR) and small-angle X-ray scattering (SAXS) is very costly and time-consuming (Alshehri et al. 2020). To tackle this challenge, bioinformatic tools have been created for protein sequence structure labelling. These tools are often accessible via a web server, such as the DISOPRED server (Jones and Cozzetto 2015), which uses machine learning methods to make predictions. This project will produce a similar tool for classifying intrinsically disordered proteins after experimentation with different deep learning architectures and approaches.

## 2.2 Deep learning overview

### 2.2.1 Deep Neural Networks

Deep neural networks are made up of stacked neural network layers. As these layers are stacked, the output of one layer will be used as input to the next layer in the network. Each of these layers performs feature extraction on their input. Each neural network layer consists of an arbitrary number of neurons with associated weights and bias parameters that are updated throughout training. During training, inputs are classified by the model during a forward pass through the model. These predictions are compared to the true value using a loss function. Through the process of backpropagation (Goodfellow et al. 2016), the outcome of the loss function is used and the parameter weights within the network are changed in an aim to improve the model's predictions. To complete this training step, an optimisation method will update each layer of the network, changing the weights of the model as proposed by backpropagation. This training procedure is done with each item in the training set and for multiple iterations of the entire training set, to solve the optimisation problem by minimising the loss and finding optimum weights for the neural network. The machine learning framework, PyTorch (Paszke et al. 2019), provides the functionality to build neural network layers that extract features; use a loss function with an implemented backpropagation method, and create an optimiser to update a deep neural network model during training. PyTorch also provides efficient tensor computations and can utilise GPUs while training. This software will be useful for creating our deep neural networks due to its efficiency and comprehensive components.

### 2.2.2 Convolutional Neural Networks (CNNs)

A CNN is a type of neural network that uses convolution kernels to extract features from an input. Convolution kernels can be 1D kernels of size $1 \times K$ primarily used for sequence or series data, or 2D kernels of size $N \times M$ primarily used for classifying images. These kernels move across a grid shaped input, checking if a feature is present. With our protein sequences, we can use a kernel of $1 \times K$ to slide along a $1 \times \textit{sequence length}$ grid, treating each amino acid as a different input channel and capturing unique features. We can also represent protein sequences as images by creating a $20 \times \textit{sequence length}$ grid, where each amino acid is a $20 \times 1$ vector and this lets us use a different 2D convolution kernel over the protein sequence. These ideas are discussed further in our design (Section 4.2.2).

In classification tasks convolution kernels are translation-invariant and require few parameters. These are beneficial as this means it does not matter where the location of the disordered region exists in the sequence. Less parameters means the model can be trained well with smaller training sets. This is good as the labelled datasets of manually annotated disordered proteins, such as the curated database Disprot (Quaglia et al. 2022), are small in comparison to the data provided for other deep learning tasks. Jones and Cozzetto (2015) have stated further annotated training labels will be beneficial to train protein disorder prediction tools; therefore, using a model architecture that can cope with smaller datasets should be effective.

**Fully Convolutional Networks**

During most classification tasks, input of the same shape is passed to the CNN. However, with our protein sequence data the lengths of our sequences range from 19 residues (the protein Conantokin-Pr1 (The UniProt Consortium 2023)) to 34350 residues (the muscle protein Titin (The UniProt Consortium 2023)), so we would need to unnecessarily pad many sequences with lots of redundant blank values. Therefore, we have taken an approach to use a fully convolutional network (FCN). Modern classification networks have been adapted to fully convolutional networks by Long et al. (2015), demonstrating that classification can be done using inputs of arbitrary size. Thus, for these protein sequences of varying length, an FCN will be suitable at handling these inputs.

### 2.2.3 Recurrent Neural Networks (RNNs)

A RNN is a type of neural network that is well-suited for processing sequential data. Most sequential data that is processed by RNNs are time series or natural language; however, many RNNs handle biological sequences (Hawkins and Bodén 2005). RNNs can capture sequential dependencies of an input sequence by using hidden states, which help to make predictions about later elements in the sequence. The hidden states are updated at each item of the sequence; using both the current input and the previous hidden state. This allows a RNN to share knowledge about previous information in the sequence, meaning that the hidden state used to make the prediction about a given element of the input sequence has a greater understanding of the context of the sequence.

With protein sequences, using an RNN is useful because maintaining sequential context stores information about the dependencies between adjacent amino acids in a protein sequence, which is important for predicting whether a given amino acid is likely to be part of a disordered region. Specifically, the hidden state will capture the context of the surrounding amino acids, like capturing information using the width of a convolution kernel, and this provides information about the amino acid at a position of the sequence, which helps make accurate predictions about whether it is ordered or disordered.

**Long Short-Term Memory (LSTM) networks**

Traditional RNNs experience the vanishing gradient problem. This happens during backpropagation when the gradients become very small, which results in the RNN being incapable of learning long-term dependencies (Arbel 2018). LSTMs address this problem by preserving gradient information using memory cells and gating mechanisms. This controls the information flow through the network, selectively maintaining some previous sequential context, which enables the network to learn long-term dependencies throughout the entire sequence. This can be effective for protein sequences as there are long-term dependencies and relationships between amino acids throughout the sequence, and this additional contextual information can increase the accuracy of predictions about attributes of the protein, such as where the disordered regions are.

## 2.3 Review of current approaches

There are currently many approaches to protein disorder prediction, with over 100 different predictors proposed (Zhao and Kurgan 2022). Disorder classifiers implement different methods with the main approaches regarded as: composition-based methods (PONDR VLXT (Romero et al. 2001)); hydropath-based methods (DISOPRED (Ward et al. 2004)); structural methods (IUPred (Dosztányi 2018)); physiochemical methods (FoldIndex (Prilusky et al. 2005)); machine learning methods (DISOPRED3 (Jones and Cozzetto 2015)) and the use of meta-predictors (Metapredict (Emenecker et al. 2021)). The Critical Assessment of protein Structure Prediction community (CASP) (Moult et al. 1994) motivated solutions for identifying disordered regions in target proteins between 2010-12 using CASP9 and CASP10. Results from the CASP10 assessment (Moult et al. 2014) show that the top disordered region classifiers consist of machine learning and meta-predictor methods (Zhao and Kurgan 2022). CASP has discontinued the evaluation of IDR prediction, however the Critical Assessment of Intrinsic Protein Disorder community (CAID) (Necci et al. 2021), continued to release withheld testing datasets to assess new prediction tools. These assessments found that deep learning predictors had the best performance at identifying disordered regions accurately (Zhao and Kurgan 2022). This motivates a reason for approaching this task using a deep learning method.

As well as performing well for intrinsic disorder prediction, deep learning methods also dominate in other protein structure prediction tasks such as AlphaFold (Jumper et al. 2021). Improved technology has assisted these neural networks to feasibly increase their depth as well as the increase of digitalised labelled data. Studies have analysed different deep learning architectures for protein disorder prediction and state: "that the architectures of current deep learners are considerably diverse," and there is no decided optimal architecture yet (Zhao and Kurgan 2022). This motivates us to experiment and compare different deep learning architectures. Furthermore, the top performing models in CAID "utilize deep convolutional and/or recurrent neural networks," (Hu et al. 2021) demonstrating these architectures are important for classifiers tackling this prediction task.

### 2.3.1 Convolutional neural networks

A model using a convolutional neural network architecture will process the amino acid sequence in a sliding window fashion using a convolution kernel. Older competitive machine learning based methods such as DISOPRED, FoldIndex and IUPred use a sliding window within their algorithm, which justifies the feasibility of a convolution kernel. There are not many prominent models which solely use convolutional layers for protein disorder classification. However, two studies propose using a CNN for similar tasks: predicting DNA-protein binding (Zeng et al. 2016), and protein secondary structure prediction (Ema and Adnan 2022). The first study concludes that experimentation of different CNN architecture design demonstrated improvements to a state-of-the-art solution and this work "will be important for sequence-based tasks in genomics," (Zeng et al. 2016). This suggests experimentation using a CNN will be useful for classification of disordered regions as we will be using a sequence-based approach with our protein sequences. The second study successfully implements a (2D) CNN with a hybrid machine learning layer and uses protein sequence data for protein secondary structure prediction, which is a similar task to our protein disorder prediction problem. Although CNNs are very good at capturing local relationships, they are more limited in identifying medium and long-range sequence information. Wang et al. (2015) proposes DeepCNF-D, which uses a deep convolutional neural fields architecture (comprising of a deep convolutional neural network and conditional random field) to reach state-of-the-art performance on the CASP benchmark datasets. The result of using a CNF architecture demonstrates the benefit of capturing complex dependencies from the amino acid sequences.

### 2.3.2 Recurrent neural networks

Neural networks that can capture the long-range and complex sequence dependencies that are encoded in IDPs have demonstrated an improved performance at prediciting protein disorder. Recurrent neural networks can use their ability to maintain a hidden state to capture long-term dependencies, which is shown by the Long Short-Term Memory (LSTM) deep recurrent neural network SPOT-DISORDER (Hanson et al. 2017). This study achieved the best performance or matched best performing protein disorder prediction methods for all independent test sets and demonstrated "the capability of the LSTM technique to pick up long-range, non-local interactions" (Hanson et al. 2017). This approach also used a bidirectional recurrent neural network architecture, which is known to capture long-term dependencies and perform better than a traditional LSTM (Graves and Schmidhuber 2005). Therefore, these bidirectional LSTM architectures are more likely to enhance protein disorder classification methods and should be considered with our approach.

### 2.3.3 Ensemble approach

As no definitive optimal architecture has been found, taking the consensus from multiple predictions using different models with different architectures can improve prediction accuracy. An improvement to the previous LSTM version of SPOT-DISORDER takes this ensemble approach, supporting that "the use of an ensemble predictor minimizes the effect of generalization errors between models" (Hanson et al. 2019). This SPOT-DISORDER2 ensemble uses CNN and RNN (LSTM) models, which improves prediction performance by removing "spurious false predictions" (Hanson et al. 2019). This paper finds benefits of using multiple models and highlights the benefit of experimenting with different architectures. These ensemble methods have also been motivated by meta-predictors which use other disorder predictors to reach a consensus prediction (Emenecker et al. 2021). These meta-approaches succeed at reaching top results and demonstrate the need for exploration of different model architectures.

### 2.3.4 Accessibility of disorder classifiers

Having multiple prediction models also allows researchers to create their own consensus of disorder prediction using different models. It is important that these prediction tools are easily accessible. Most trained classifiers are accessible to download and use with protein sequence data, which is beneficial for bioinformaticians. Furthermore, many studies produce an online web server to benefit less technical researchers, where they can make predictions on protein sequences using the trained classifiers via their browser. With this easy access, research can benefit from accurate predictions about protein sequences. These web servers display highlighted disordered regions within the protein sequence or graphs demonstrating the probability that a region is intrinsically disordered. This makes it clear from visual inspection where the protein is likely to misfold.

### 2.3.5 Summary

The findings of these current approaches demonstrate that intrinsic disorder prediction in proteins is successful with deep learning classifiers and that the exploration of different accurate models will allow for future ensemble and meta predictions to be taken. The use of protein disorder prediction by researchers also makes it clear why an easily accessible web server is important in allowing interaction with the prediction tool.

# 3 Analysis

## 3.1 General deep learning problem

In our discussion of intrinsically disordered proteins (Section 2.1.2), we demonstrate that the sequences of these proteins are different compared to ordered protein sequences. If we evaluate which amino acids appear within specific regions and where they appear, we can draw conclusions about where these disordered regions are. Therefore, we will use a supervised learning approach with our deep neural networks, like current approaches, by using these amino acid sequences alongside their labelled disordered regions.

## 3.2 Our dataset

As the identification of these disordered regions requires complex experimentation (Section 2.1.2), we must use a curated database of disordered proteins. The Disprot database (Quaglia et al. 2022) has been used for a variety of machine learning tasks such as the training and testing of DISOPRED3 (Jones and Cozzetto 2015), PONDR (Xue et al. 2010), MFDp2 (Mizianty et al. 2013) and SPOT-DISORDER (Hanson et al. 2017), which all involved classifying disordered proteins. This database is regarded as the "major repository of manually curated data for intrinsically disordered proteins" (Quaglia et al. 2022) and is widely used by researchers studying disordered proteins. Disprot stores information about the positions of IDRs in sequences and uses a UniProtKB accession number to identify the protein. This identifier will allow us to retrieve the full protein sequence from the UniProt database (The UniProt Consortium 2023), another manually curated database for proteins, to help us train our network.

## 3.3 Labelled data

Compared to other problems solved by deep neural networks, such as protein structure prediction (Jumper et al. 2021), classifying images (Xie et al. 2017) and language modelling (Devlin et al. 2018), the size of our manually labelled dataset is much smaller. Disprot only contains around 2500 protein sequences. However, we are not classifying an entire sequence as either ordered or disordered, we are classifying a single amino acid within the sequence as ordered or disordered as seen in Figure 3.1. This is because we know the region of the protein can be disordered due to the characteristics of the grouped amino acids. This gives us sufficient labelled data. Using these

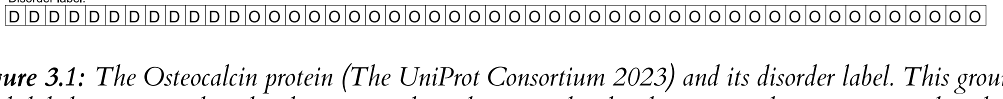

Protein sequence:
Y L D H G L G A P A P Y P D P L E P R R E V C E L N P D C D E L A D H I G F Q E A Y R R F Y G I A
Disorder label:
D D D D D D D D D D D D O O O O O O O O O O O O O O O O O O O O O O O O O O O O O O O O O O O O O

***Figure 3.1:*** *The Osteocalcin protein (The UniProt Consortium 2023) and its disorder label. This ground truth label, represents disordered amino acids with a D and ordered amino acids are represented with O. Here we see the disordered region is at the start of this small protein sequence*

labelled sequences as our ground truth, we can utilise our deep neural networks to recognise patterns and relationships about the amino acid sequence that causes intrinsic disorder in proteins through our supervised learning approach.

## 3.4 Approaches

Machine learning algorithms cannot operate on our protein sequences directly. Therefore, to ensure our model is equipped to handle the various amino acids, we will consider them as distinct categories of data and employ feature encoding techniques to generate a suitable numerical input (Kumar 2020).

### 3.4.1 One-hot

Our first feature encoding approach is one-hot encoding (Fawcett 2021). We can treat each amino acid as a vector by using the one-hot encoding technique. Our vectors will have a size of 20 because of the twenty possible amino acids, therefore as these feature vectors are not excessively large, this one-hot approach is suitable. This has let us transform a raw amino acid sequence into a feature matrix, of shape $20 \times \textit{sequence length}$, which now represents the sequence.

### 3.4.2 Position-Specific Scoring Matrix (PSSM)

Our other feature encoding approach is to generate a position-specific scoring matrix (PSSM) (Wikipedia contributors 2022). This technique is used in many protein structure problems, such as predicting protein secondary structure (McGuffin et al. 2000), identifying sequence-contact relationships (Wang et al. 2017) and used within current approaches previously discussed in our review of the literature. A PSSM is a "matrix that involves information about the probability of amino acids or nucleotides occurrence in each position, which is derived from a multiple sequence alignment" (Mohammadi et al. 2022). These PSSMs provide an informative representation about the amino acids at positions within the protein sequences. Specifically, PSSMs capture information about the evolution of the protein in terms of which amino acids can occur at different positions in the sequence and this is used to determine whether a specific residue position will be a small amino acid, a bulky amino acid, a negatively charged amino acid, or have any other specific amino acid property. This information will be helpful for our network to identify patterns and relationships within protein sequences. Furthermore, as PSSMs are also of shape $20 \times \textit{sequence length}$, the neural network models will be able to take both one-hot encoded sequence matrices and PSSMs as input without any changes, allowing us to compare these different feature encoding methods.

## 3.5 Architectures

Many problems in deep learning can be tackled using different neural network architectures. These networks can vary in terms of depth, complexity, and computational efficiency, among other factors. As shown by our literature review (Section 2.3), different convolutional and recurrent neural networks have been successfully applied to protein disorder prediction. Therefore, to build upon this knowledge, we will implement different models using these architectures. With these models, we will compare their predictions of protein disorder and consider the effect that different architectures have to the performance of the task.

### CNNs

The first neural network architecture we consider is a CNN. We know CNNs can capture local patterns throughout the sequence and consider dependencies between amino acids (Section

2.2.2). We aim to experiment with different CNN architectures; using both one-dimensional and two-dimensional convolution kernels for different models as there is a varied use of these within the literature.

A CNN using 2D convolution kernels will capture spatial information about the arrangement of the amino acids within a protein sequence very well. As our feature encoded sequences have two dimensions, height and width, we can apply these 2D convolution kernels over an entire window of vectorised amino acids to capture local patterns and dependencies between neighbouring amino acids and their structural features. This method can identify patterns and features that appear in the different disordered regions.

A CNN using 1D convolution kernels will also recognise patterns and dependencies between amino acids in a protein sequence. Using CNNs with 1D convolutional kernels are more commonly used for analysing and classifying sequential data. Therefore, we can also evaluate if the CNN architecture that has a more natural application to sequences will perform better at predicting protein disorder within our protein sequences. The structure and complexity of the CNNs we built are discussed later in our design (Section 4.2.2).

#### RNNs

A different neural network architecture we consider is the RNN. This architecture processes sequential data and can maintain the sequential context about surrounding amino acids (Section 2.2.3). Protein sequences have many amino acids, and amino acids along the sequence can affect the folding of other amino acid residues further along the sequence, resulting in a disordered region. Furthermore, we know that RNNs face the vanishing gradient problem, and protein sequences have many long-term dependencies; therefore, the LSTM architecture has been considered. The LSTM architecture has been used in recent state-of-the-art models (Section 2.3.2); therefore, we will create models using this architecture to evaluate and compare our LSTM with different feature inputs and our CNN models. Details about the design of the LSTM are discussed later in (Section 4.2.3).

## 3.6 Evaluation metrics

With our trained deep neural network, we will evaluate its predictions using similar approaches to the literature. As this is a binary classification task classifying amino acids as either ordered or disordered we can define our outcomes as:

- True positive (TP): When the model correctly predicts that a disordered residue is disordered.
- True negative (TN): When the model correctly predicts that an ordered residue is ordered.
- False positive (FP): When the model incorrectly predicts that an ordered residue is disordered.
- False negative (FN): When the model incorrectly predicts that a disordered residue is ordered. (Google Developers 22)

By computing these outcomes, we can evaluate the accuracy, precision, recall (sensitivity) and specificity of our model. These metrics are widely used in various machine learning tasks as standard performance measures (Wikipedia contributors 2023*b*). We can also calculate the F1-score which is the harmonic mean of the precision and recall and is another useful measure of how accurate our model is. Furthermore, as our dataset is imbalanced, with a majority of ordered labels, we will consider the Matthews correlation coefficient (MCC). This metric is a more reliable measure when dealing with an imbalanced and disproportionate dataset (Chicco and Jurman 2020).

With these evaluation metrics we will be able to compare our approaches and architectures against each other. We can also compare our models to known current approaches using the CASP datasets which have been used for benchmarking these protein disorder prediction models (Moult et al. 1994).

## 3.7 Django server

In addition to evaluating deep learning models, it is necessary to create a Django-based server that enables users to submit a protein sequence and obtain a comprehensible representation of the disordered regions of the protein. From the literature, papers proposing protein disorder prediction tools also often set up an online server so their tool can be used such as UCL's PSIPRED workbench (Buchan and Jones 2019) and the recently published Metapredict server (Emenecker et al. 2021). These protein prediction servers are important as they allow researchers to easily access these tools, allowing them to understand the structure and function of proteins which is useful for protein engineering (Wikipedia contributors 2023*e*).

A brief analysis of the server's requirements are stated below; following the MoSCoW prioritisation method (Agile Business Consortium 14):

### 3.7.1 Functional requirements

- Must have: The server will have a web interface, so it is easily accessible for use by researchers.
- Must have: Users can submit a single protein sequence via a form.
- Must have: The server can make a prediction on a protein sequence using a deep neural network. This model will be chosen from our evaluation.
- Should have: The web interface will have a clear visualisation displaying the sequence to the user with clearly identified IDRs.
- Could have: The web interface will have a graphical representation of the identified IDRs.
- Could have: The visualisation of the disordered sequence can be saved, allowing for further analysis.
- Will not have this time: The server will not allow multiple sequences to be entered in a single form submission.

### 3.7.2 Non-functional requirements of our server

- Must have: The web interface will be intuitive and easy to use. Both when entering information in via the form and interpreting the outputted sequence with highlighted disordered regions.
- Should have: The web application should be containerised to follow best practices and be easily portable.
- Could have: The Django server could interact with an external database for efficient lookup of previously entered sequences and their representations.
- Will not have this time: The web page will not be hosted and accessible via a searchable domain by the conclusion of the project.

# 4 Design

## 4.1 Data processing

We will need to build a data processing pipeline where we will gather specific information about each unique protein from the DisProt dataset, such as their full sequence and the location of all the disordered regions, so we can create our ground truth label. This ground truth label is vital for our supervised learning task. Given our DisProt dataset, we identify unique proteins using a UniProt accession identifier. Therefore, we should use this accession identifier as a key in hash map data structures. This will let us easily lookup the feature encoded sequence for our DNN's input and a sequence's true disordered regions which let us create our label for evaluating our prediction. We build the label by creating a vector the size of the sequence, and from our looked up disordered regions of a given protein we will set the amino acids where intrinsic disorder occurs to 1 (as disorder is our positive class) and ordered amino acid positions will be set to 0. This vector will be used in our loss function which is discussed later in Section 4.3.

Processing is also required to encode these sequences before they are looked up. We have two different vectorised representations of these sequences: a one-hot encoded matrix and a PSSM. The one-hot encoded matrix is a sparse matrix and is made using standard one-hot feature encoding. An example of the beginning of a one-hot encoded sequence is shown in Figure 4.1. The generation of the PSSMs is more computational than the one-hot encoding and involves using a sequence alignment search tool alongside a large protein sequence database. This processing gives us information about the conservation of amino acids between similar sequences which conveys more information about the sequence than a one-hot representation. An example of a PSSM is shown in Figure 4.1, and each PSSM can be parsed into a 2D data structure which creates the appropriate matrix format our deep neural network can use. A limitation of using PSSMs is that they are much more computationally expensive to create than our one-hot encoding approach as they perform sequence alignment using a very large protein database such as UniProt (The UniProt Consortium 2023). For this reason, we have also implemented a one-hot encoding approach to see if generating PSSMs is worth the computational expense.

### 4.1.1 Training, validation and testing data

Our final key processing step is to divide our dataset into a training, validation and test set before any learning takes place. We follow the common approach of using 20% withheld data for the test set, and the rest will be used for training and validating the model. We will further separate this training and validation set by 75% and 25% respectively so that the overall split of training, validation and testing data is 60%, 20% and 20% respectively. Random sampling (with a seed to ensure reproducible splitting) will be done to separate these sequences. However, as protein sequences can have a similar ancestor, this means they can be homologous and there may be data leakage between datasets. This means information from the test dataset can accidentally appear in our training dataset (Cook 2022). To detect this data leakage, we will look for homologous sequences by employing a homology clustering search. We will avoid any data leakage by moving sequences between datasets if they are homologous, such that all datasets will not have homologous sequences between them. Despite movement of sequences, we still aim to have a division of approximately 60%, 20%, 20% between our three datasets and will verify this before we perform further evaluation.

*(a) One-hot encoding*

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

*(b) Position-specific scoring matrix*

| | M | A | S | R | E | ... |
|---|---|---|---|---|---|---|
| A | −2 | 4 | 0 | −2 | −1 | |
| C | 0 | 0 | 0 | −2 | −2 | |
| D | −3 | −2 | 0 | −1 | 0 | |
| E | −4 | −3 | −2 | −1 | 4 | |
| F | 1 | −1 | −1 | −2 | −2 | |
| G | −3 | 0 | −1 | −2 | −2 | |
| H | −2 | −2 | −2 | −1 | −1 | |
| I | 2 | 0 | −1 | −2 | −2 | |
| K | −2 | −2 | −1 | 0 | 0 | |
| L | 3 | 0 | −1 | −1 | −2 | ... |
| M | 7 | −1 | −1 | −1 | −2 | |
| N | −3 | −2 | 0 | −1 | −1 | |
| P | −3 | −1 | −1 | −2 | −2 | |
| Q | −2 | −2 | −2 | 2 | 0 | |
| R | −3 | −4 | −3 | 3 | −2 | |
| S | −3 | 0 | 4 | −2 | 0 | |
| T | −1 | −1 | 0 | −2 | −1 | |
| V | 1 | 0 | −1 | −2 | −1 | |
| W | 0 | −1 | −1 | −1 | −2 | |
| Y | 0 | −1 | −1 | −1 | −1 | |

***Figure 4.1:*** *Two possible ways of encoding our protein sequences. (a) shows a one-hot encoding approach, where 1.0 identifies the amino acid at each sequence position. (b) shows the PSSM, where larger positive values are given to amino acids that are expected to appear at each position of the sequence. This protein is Human adenovirus C serotype 5 (HAdV-5) (Human adenovirus 5), the first protein sequence in the DisProt dataset, and we show the first five amino acids from this sequence.*

## 4.2 Our neural networks and how we train them

### 4.2.1 Model input

Giving the model input is the initial stage in training a DNN. The input to our models will be our feature encoded sequence (either a one-hot or position-specific scoring matrix), which will have shape 20 × *sequence length*. We will only give a model one protein sequence to evaluate at each forward pass as we know that sequence lengths differ between proteins; therefore, we cannot batch multiple sequences together. Batching multiple sequences would create an unexpected, irregular shape, which neural network layers cannot handle. Thus, we use a batch size of one as input to our DNN models.

### 4.2.2 CNNs

Our CNNs must actually be fully convolutional neural networks (FCNs) because our protein sequences have different lengths, therefore we cannot use a fully connected layer as it expects a fixed size tensor (Long et al. 2015). So, we will only use convolution layers which create feature maps for our model to identify information about the protein sequences used for the disorder classification task. These layers must also reshape our input to a $1 \times \textit{sequence length}$ length tensor for our final disorder prediction.

There are two different convolutional neural networks we can design: one that treats our protein sequence as a black and white image and uses 2D convolutions, or one that treats each row in the encoded matrix (different amino acids) as a different feature representation of the input sequence and uses 1D convolutions over these separate feature input channels.

#### Using 2D Convolutions

Usually, a 2D convolutional kernel is applied to images, therefore given our $20 \times \textit{sequence length}$, the height of the proposed image is the number of possible amino acids, and the width of the image is determined by the sequence. Our input to a CNN using 2D convolutions must follow the [batch size, channels, height, width] constraint, therefore our input will be a tensor of shape [1, 1, 20, sequence length] due to batching of size 1 and treating the colour channel as black and white.

As discussed earlier, we want our convolution kernel to behave like a sliding window (Section 2.3.1), therefore it must overlay the entire height of the image to take in a full context window of amino acid information. Capturing the entire height involves using a kernel of size 20, however we cannot appropriately pad the width of the sequence with this kernel shape such that the same width is returned after convolving with a kernel of this shape. The width cannot be changed as it is necessary to compare each amino acid disorder classification against its true label. Therefore, we will use a non-square convolution kernel so that the width of our kernel is an odd value and appropriate padding can be included.

After the first convolution layer, layers after this will be dependent on the feature map created by this first layer and further layers. The number of feature channels produced by this first convolution layer can be experimented with. We chose to maintain a single hidden channel throughout the network. After further convolution layers we want our final feature map output to take shape $1 \times \textit{sequence length}$ so our predictions can be compared against the true label.

#### Using 1D Convolutions

Our other fully convolutional network uses 1D convolutions. Usually, a 1D convolution kernel is applied to sequences and accepts an input of $1 \times \textit{sequence length}$ shape. For our neural network, we will treat each amino acid feature representation as a separate input channel, giving us 20 channels, each with shape $1 \times \textit{sequence length}$. The input to the model must follow the [batch size, input channels, width] constraint, therefore our input will be a tensor of shape [1, 20, sequence length]. This 1D convolution kernel also behaves like a sliding window, and a kernel will produce one output feature map which represents all the amino acid input channels. This ensures all amino acid information is captured for a context window at each position along the sequence. Furthermore, an odd numbered kernel must be chosen so that appropriate padding can be included to the width of our sequence similarly to the 2D CNN. Like the 2D CNN, all further convolution layers can be changed within the network as long as the final layer produces a $1 \times \textit{sequence length}$ feature map as its output for comparison with our ground truth labels.

### 4.2.3 RNN

Our RNNs will use the long short-term memory (LSTM) architecture. These LSTMs combat the vanilla RNN's vanishing gradient problem, and they can identify long-term dependencies which we know is useful from our literature analysis. Usually, LSTMs are applied to sequential data, and they expect a batch size, a sequence length and a number of feature channels. These features are represented in the same way as our 1D CNN initial feature channels; where each amino acid row from our vectorised protein sequence is used to represent the sequence as a separate feature vector. As using batching requires our sequences to be the same length, which we do not have, we will continue to batch with a single sequence in each batch instead of padding our sequences. Therefore, our input to the LSTM model will be a tensor of shape [1, sequence length, 20].

We also need to initialise our hidden state for our LSTMs. These will be initialised with all zeros every forward pass, so they are reset each sequence. It is common practice to set the hidden state to an all zero tensor with a shape considering the batch size and the number of hidden dimensions we want. In our model we will choose to have 32 hidden dimensions.

Finally, our LSTM will be a bidirectional LSTM (BLSTM). This performs better than a traditional LSTM and can capture long-term dependencies about the IDPs using information from the start and end of these sequences. With information flowing in both directions, we can capture more contextual information about each amino acid. Furthermore, a protein sequence does not have a flow of information like a time series, and the start (N-terminus) and end (C-terminus) of a sequence is chosen arbitrarily, meaning this protein could be written in the opposite direction. Thus, it is useful to assess each sequence from both directions. A limitation of implementing a bidirectional network is that it will be much slower to compute because forward propagation requires both forward and backward recursions, which creates a long dependency chain and affects our gradient calculation using backpropagation (Zhang et al. 2021).

Like our other networks, our LSTM will use a final linear layer to reduce our feature map down to a single feature (the disorder classification), so that a $1 \times \textit{sequence length}$ output is returned for the rest of our training loop to handle.

### 4.2.4 Activation functions

Between each layer in our neural networks, we use the rectified linear unit activation function (ReLU) as it is "the most widely used and a goto activation function for most types of problem," (Keerthi 2022). This is because ReLU can be more computationally efficient than other activation functions, and using ReLU in networks tends to give a better convergence performance (Beren-Luthien 2016). After our final layer, we employ the sigmoid activation function because this gives us a value in the range [0, 1]. We can treat this value as a probability as it shows how likely that this position in the sequence is a disordered amino acid. This value alongside a threshold of 0.5 will be used to identify the predicted IDRs; where any position with a value greater or equal to 0.5 can be regarded as disordered. Furthermore, these probability values can be evaluated under various thresholds, letting us assess the effectiveness of our models, which is further discussed in our design of the evaluation (Section 4.4.1).

## 4.3 Discussion about loss function and optimisation

To allow our deep neural network to learn and update its many different weights in each layer, we must use an optimisation method and an appropriate cost function. We know our problem is a binary classification task from our analysis of the problem; therefore, we can use binary cross-entropy (also known as log loss) as our loss function (Godoy 2018), shown in Figure 4.2.

With these two possible outcomes we are treating a disordered amino acid as the positive class and an ordered amino acid as the negative class. From analysing our DisProt dataset, we see that our two classes are imbalanced: there are a lot more ordered amino acids within these intrinsically disordered proteins, specifically there are approximately five ordered amino acids for every disordered amino acid. One way of handling this imbalanced dataset is using a weighted loss function. For this, we calculate the number of ordered and disordered amino acids in our training dataset. Then, we can calculate a weight multiplier to be used to give the disordered amino acids loss more weighting. By scaling the loss of predictions about the amino acids we will emphasise predictions that should take a disordered label which will help the model better identify this disproportionate class because the model will be penalized more heavily for misclassifying disorder.

This binary cross-entropy loss function is useful for gradient-based optimisation algorithms because it is easy to compute and is sensitive to small changes in the predicted probabilities. We plan to use stochastic gradient descent (SGD), which is a popular machine learning optimisation algorithm. We will also experiment using the Adam optimiser, however as this optimisation method usually works well with large datasets and SGD is known to generalise better than Adam (Zhou et al. 2020), we believe SGD may be a better optimisation method and produce a better model.

$$\ell(y, \hat{y}) = -\frac{1}{N} \sum_{i=1}^{N} w_i [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)] \tag{4.1}$$

***Figure 4.2:*** *Binary Cross-Entropy (Log Loss) Function. This equation represents the loss function used in binary classification tasks. For our task, $y$ is the true label of order or disorder, $\hat{y}$ is the predicted probability (i.e., the output of the sigmoid function), and N is the number of amino acids in the sequence. The function measures the difference between the predicted and true labels. The weight vector w, is used to penalise the model more when it is more certain about incorrect predictions of the disordered class. Having an all ones vector for w is default when no weights are specified.*

## 4.4 The design of our evaluation experiments

During the training of our model, we will evaluate the performance of the optimisation task. We can use the total loss of our training set and validation set predictions to ensure our model is learning at a sufficient rate and is generalisable on withheld data. Using the validation dataset as our first withheld dataset, we can optimise hyperparameters, such as the learning rate and regularisation weights to make sure our validation loss decreases along with our training loss. Furthermore, we will plot loss curves to make it visually clear our model is generalisable to our validation dataset (Brownlee 2019). With a generalisable model we are more likely to achieve correct classifications on future unseen data.

After tuning the important hyperparameters for our different model configurations we will retrain our model using the 80% of the DisProt dataset allocated to training data (this includes our training and validation datasets). Using the other 20% of the DisProt dataset withheld for testing, we will evaluate our models using the evaluation metrics discussed in our analysis (Section 3.6). Discussion about these metrics and the equations for these standard machine learning evaluation measures are given below.

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

Accuracy, where TP (true positives) is the number of correctly classified disordered amino acid instances, TN (true negatives) is the number of correctly classified ordered amino acids instances, FP (false positives) is the number of incorrectly classified disordered amino acids instances, and FN (false negatives) is the number of incorrectly classified ordered amino acids instances.

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

Precision measures the proportion of amino acids classified as disordered among all disorder predictions.

$$Recall = \frac{TP}{TP + FN},$$

Recall measures how many disordered amino acids are actually detected among all actual disordered amino acids.

$$Specificity = \frac{TN}{TN + FP},$$

Specificity measures the proportion of amino acids classified as ordered among all order prediction instances.

$$F1\ score = \frac{2 \times Precision \times Recall}{Precision + Recall},$$

F1 score is the harmonic mean of precision and recall and this provides a balanced measure of both these metrics.

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

MCC considers all four confusion matrix elements. It ranges from -1 to 1, with 1 indicating a perfect correlation, 0 indicating no correlation, and -1 indicating a perfect negative correlation.

### 4.4.1 Evaluation of classification thresholds

Finally, we consider the probability-based values returned from our sigmoid activation function in our evaluation. In a binary classification task, classifying the positive class can be imperative, such as cancer detection where false negatives could lead to delayed diagnosis and missed treatment opportunities. Ardila et al. (2019) agree that classifying the positive class can be critical with their production of a DNN for lung nodule detection. In some scenarios, it may be more important to predict every disordered amino acid correctly than it is to misclassify more ordered amino acids. However, for our evaluation it is important our model still behaves sensibly. To assess a model's capability at predicting disordered amino acids correctly we can use receiver operating characteristic (ROC) curves and the area under the ROC curve (AUC). This lets us determine what an appropriate threshold for identifying disorder can be and if our models perform better than a dummy classifier that predicts at random.

### 4.4.2 Evaluating on different datasets

After our full evaluation is complete, we will finally train our model using all (100%) of our data from the DisProt database as the separate datasets have served their purpose in helping us find the best model. Training with this extra data means that our model should have at least the performance accuracies as it did in our evaluation, and this is a best practice before deploying a machine learning model (Brownlee 2017). After this step, our trained model is ready to be deployed on our web server and can be assessed using other unseen datasets.

Furthermore, we will use the CASP10 dataset to evaluate our model. This dataset contains 94 protein sequences (Moult et al. 2014) and has been used to assess many protein disorder classifiers. Details of other protein disorder prediction methods performances can be found in other papers; therefore, we can compare our findings against other approaches and architectures.

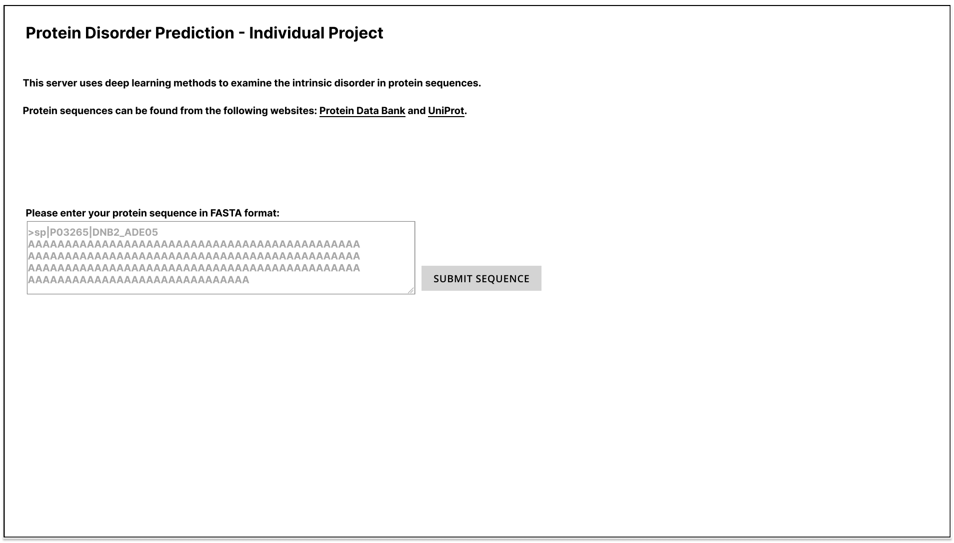
## 4.5 The web server

### 4.5.1 User interface

The user interface was designed to be similar to other protein disorder prediction websites. All websites used an input form, stated what input was valid and displayed the output to the user in a timely manner for single sequences. From checking popular protein prediction web servers, we felt the most user-friendly websites included an example of what sequence input was valid, therefore we made this a consideration when building the website.

We have two pages on our web server: the input form and the displayed disorder prediction.

**The input form web page.** This page was made to be intuitive, and its main feature is a form to let the user input a protein sequence. This satisfies must have functional requirements for the web server (Section 3.7). User input can only be a protein sequence in valid FASTA format, which is explained on the web page. We have also included links to PDB, UniProt and DisProt to assist users in finding protein sequences. The wireframe for this page is shown in Figure 4.3.



**Protein Disorder Prediction - Individual Project**

This server uses deep learning methods to examine the intrinsic disorder in protein sequences.

Protein sequences can be found from the following websites: Protein Data Bank and UniProt.

Please enter your protein sequence in FASTA format:

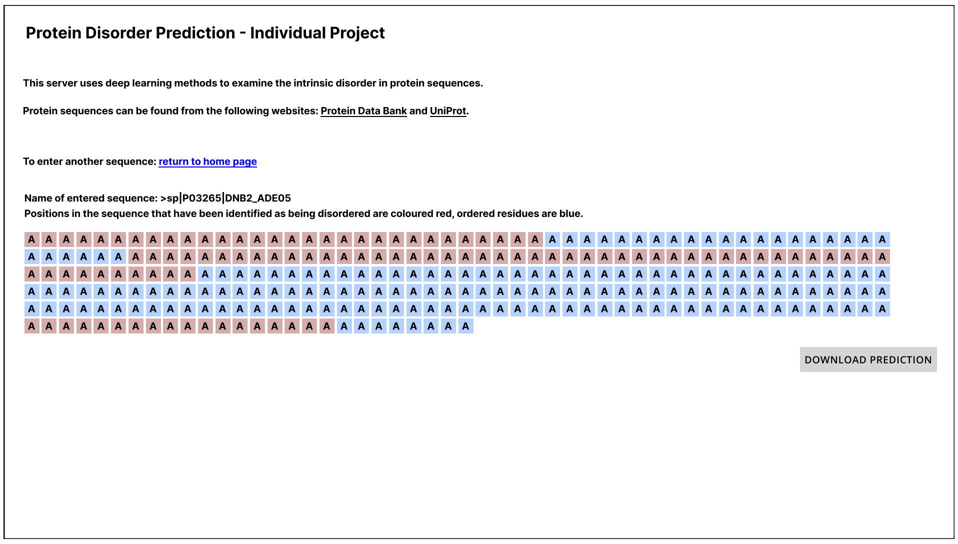
```
>sp|P03265|DNB2_ADE05
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

*Figure 4.3: The home page of our web server. This wireframe shows that users can navigate to external protein database websites, and enter their protein sequence in FASTA format to our input form.*

**The disorder prediction web page.** This page displays the IDR predictions. We have used a sequence-based visualisation, which demonstrates which amino acids are disordered and where the disordered regions are. Other web servers such as ESpritz (Walsh et al. 2012) have also used a sequence display, but this website is older and uses a 'D' or 'O' label next to the amino acid instead of a coloured label. We believe that the coloured approach is more effective at representing the disordered regions and has also been adopted by web servers that have been maintained and updated recently like PSIPRED (Buchan and Jones 2019). The wireframe of this page is shown in Figure 4.4.

## 4.6 Tools and technologies

It is important that the trained deep neural network model must be easily accessible on the web server. Using PyTorch for the deep learning and Django for the web server will allow us to integrate our trained PyTorch model into our web application easily, using Python for consistency.


**Protein Disorder Prediction - Individual Project**

This server uses deep learning methods to examine the intrinsic disorder in protein sequences.

Protein sequences can be found from the following websites: Protein Data Bank and UniProt.

To enter another sequence: return to home page

Name of entered sequence: >sp|P03265|DNB2_ADE05
Positions in the sequence that have been identified as being disordered are coloured red, ordered residues are blue.

*Figure 4.4: The disorder prediction web page. This wireframe demonstrates how the prediction from our deep neural network will be displayed after a protein sequence has been submitted.*

### 4.6.1 Deep learning

To create our different deep neural network architectures, the PyTorch framework (Paszke et al. 2019) was used. PyTorch lets us construct our models using their neural network library. Using Python allows us to carry out our pre-processing and evaluation steps with different libraries that help us make efficient computations along with appealing and informative visualisations about the performance of our prediction task. Using Python also lets us utilise Google colab's notebooks (Bisong 2019), where we can use a GPU for decreased training times and include markdown throughout our notebook, which helps with the understanding of our code.

An external tool is required to generate the encoded features for our model's PSSM input. We have chosen to use MMSeqs2 (Many-against-Many sequence searching) (Steinegger and Söding 2017). MMSeqs2 can perform multiple sequence alignment searches much faster than PSI-BLAST (Altschul et al. 1997) and HHblits (Remmert et al. 2012) which are other software suites used for PSSM generation. Despite HHblits giving slightly more accurate alignments, this task is much more computationally efficient with MMSeqs2, which still generates accurate alignments. Recent papers, such as NetSurfP-2.0 (Klausen et al. 2019) have used MMSeqs2 to generate their PSSMs for large sequences due to MMSeqs2 being less resource intensive than HHblits and found little differences in the models' performance trained using PSSMs generated by the different suites.

### 4.6.2 Web server

To create our web server, we will use the Django web framework (Django 2005). Web pages will be built using web technologies (HTML, CSS and JavaScript). In Django's backend we will have the views which can process requests and manipulate data. We will save our trained neural network as a serialized PyTorch state dictionary, which is a Python dictionary that includes the weights, biases, and other model state variables for a PyTorch model. This saved model can be reloaded within the views of the Django backend so that the web application can make use of it. Furthermore, this should be straightforward to do as Django is a Python web framework, PyTorch can be easily imported and loaded amongst the web application logic. Finally, we used Docker (Merkel 2014) along with Nginx (Reese 2008) and Gunicorn (Wikipedia contributors 2023*c*), so that our web server can be containerised and is portable which is an initial requirement.

# 5 Implementation

## 5.1 Data processing

### 5.1.1 The IDPs

Our data processing started with getting annotated disordered proteins and their disordered regions from DisProt (Quaglia et al. 2022). We used the DisProt dataset that was released in June 2022 (release 2022_06) and kept the default settings that gave individual rows for each disordered region in a sequence, included ambiguous sequences and excluded obsolete sequences. We downloaded the TSV format of this file because it works particularly well with pandas (McKinney 2010), which we used at the start of our data processing stage. We stuck with this dataset throughout the entire project to maintain consistency in our experiments. To download this dataset, DisProt's RESTful API search feature (Quaglia et al. 2022) was used. We customised the request to match the terms given for the default settings on the main website and downloaded this file to our data folder. These terms can be modified, so that the most up to date dataset can be downloaded, processed and used to train and evaluate our models in the future.

DisProt only contained information about disordered regions, such as the start and end of these disordered regions, but not the consensus of all disordered regions for a given protein sequence or the full sequence itself. However, DisProt uses the UniProt accession identifier to identify proteins. Therefore, we grouped all disordered regions for a single identifier together, so they could be looked up to build a ground truth label. We also used these unique identifiers to make requests using UniProt's REST API (The UniProt Consortium 2023), where we retrieve a sequence in FASTA format. We take the substring of the FASTA file, so that we just have the sequence. During these pre-processing stages we found that some protein sequences have been deprecated from UniProt (as it is updated more regularly than DisProt). To ensure extensibility of this processing stage we remove proteins from our DisProt dataset if the UniProt REST API identifies them as deprecated. This did not have an impact on the size of the dataset; we removed three protein sequences. Furthermore, we also remove sequences that are returned by UniProt with ambiguous amino acids. This is because these amino acids were not recognised in the list of twenty known amino acids and our vectorisation feature encoding cannot handle these unknown characters. This also has no effect to our dataset; we removed a further seven protein sequences.

Overall, this first processing stage of the IDP data from DisProt has produced two important data structures we needed: a map to all of our disordered protein sequences, and a map to all of our disordered regions for each protein. These are both connected via an accession identifier.

### 5.1.2 Feature encoding

The next data processing step is performing feature encoding steps, so that a deep neural network can use protein sequences as input. With the one-hot encoding approach: for each sequence, if the amino acid at a position is present, then that amino acid row is given a 1 at that position, and all other amino acids are given a 0 at that position. This was done in Python, where we manipulated all zero NumPy arrays, and updated the amino acid at this position with 1 using a look up dictionary that let us access the feature vector representing that amino acid. Finally, all of these feature vectors were combined to create the full one-hot encoded matrix. For our PSSM approach: we first used the MMSeqs2 software suite (Steinegger and Söding 2017), then cleaned the generated PSSMs so they are numerical tensors. Using MMSeqs2 required a protein database, so we used the UniProtKB, which is assembled of both UniProtKB/Swiss-Prot and UniProtKB/TrEMBL (The UniProt Consortium 2023). A FASTA file of all of the protein sequences was required, so we created this using the UniProt REST API. We did not modify our sequence string and wrote the full FASTA to an external file. This ensured PSSMs would only be generated for the sequences we had validated. We performed the multiple sequence alignments (MSAs) on these sequences with the downloaded database. All of these sequences were in the same FASTA file, meaning MMSeq2's powerful scalability could be utilised. From the MSAs we can generate our PSSMs using MMSeq2's tools. Finally, these are converted to a 2D tensor using Python's string splitting functionality. For both of our feature encodings, we pair the numerical representation of the sequence with its accession identifier. This lets us quickly access our feature encoded sequences.

### 5.1.3 Separating datasets

For our training, validation and testing datasets we randomly sampled sequences using a seed so that this experiment can be repeated. The design discussed problems with our datasets (Section 4.1.1), revealing we cannot have sequences homologous to each other in different datasets. To find these homologous sequences we used MMSeqs2 clustering functionality (Steinegger and Söding 2017). We already have our training, validation and testing dataset splits; therefore, each sequence was tagged with the dataset it was in. By examining these sequences using similarity clustering between sequences we can identify which sequences are homologous to each other. We found there was 286 sequences that have homologies with sequences from other datasets, therefore our datasets were not independent. To establish independence and ensure sequences that are related to each other did not overlap different datasets, we moved these flagged sequences from their dataset to the dataset the similar, homologous sequence is in. An example of this is shown in Table 5.1. This removed the issue of leaking data. From these separated accession identifiers, three lists were created for the train, validation and test set. Using these lists of accession identifiers, we could get the sequences, ground truth labels and encoded features for any of our datasets as the unique accession identifiers are the key element to looking up all these pre-processed items.

***Table 5.1:*** *This table shows protein sequences assessed with MMSeqs2 clustering, the sequences detected as homologies, and where the assessed protein sequences were moved to create independent datasets. Sequences have been identified using their UniProt accession identifier and have the dataset they were randomly sampled to prepended to this. For example, Abscisic acid receptor PYL9 (Q84MC7) was in the test dataset, is similar to Abscisic acid receptor PYL1 (Q84MC7) which is in the train dataset, therefore we move these so that they are both in the same train dataset.*

| Sequence | Homology detected via clustering | Moved sequence |
|---|---|---|
| TEST_Q84MC7 | TRAIN_Q8VZS8 | TRAIN_Q84MC7 |
| VAL_Q63716 | TRAIN_O08807 | TRAIN_Q63716 |
| TEST_Q13162 | TRAIN_O08807 | TRAIN_Q13162 |
| VAL_Q03692 | TEST_Q00780 | TEST_Q03692 |
| TRAIN_B7T1D9 | TEST_B7T1D7 | TEST_B7T1D9 |

## 5.2 Deep learning architectures

We have implemented our deep neural networks in PyTorch (Paszke et al. 2019). The PyTorch neural network library (torch.nn) provides building blocks to add different architecture layers and activation functions.

### 5.2.1 CNNs

As discussed in our design (Section 4.2.2), our CNNs are fully convolutional networks (FCNs), therefore they have no final fully connected layer. One CNN uses multiple 2D convolution kernels and its network architecture is described in Figure 5.1. We have chosen to use a $20 \times 21$ shape for our first convolution kernel as we want to reduce the height of our tensor to 1 as discussed in our design about CNNs (Section 4.2.2), and we also found 21 as a reasonable width for classifying these sequences. The next layers use $1 \times 21$ convolution kernels. Furthermore, there is no increase to the channels as this caused the model to not be able to generalise with unseen data. After each convolution kernel the ReLU activation function is employed, until the final layer where the sigmoid activation function is used as discussed in our design of activation functions (Section 4.2.4). However, we implemented BCEWithLogitsLoss as our loss function instead of BCELoss meaning that we exclude this final sigmoid activation function in the implemented model. This is because this sigmoid layer is computed within the loss function, offering more numerical stability than the sigmoid activation function followed by BCELoss. This numerical stability is because of the log-sum-exp trick (Gundersen 20). Furthermore, this means that any output from this and other models must have the sigmoid function immediately applied to it, so we can assess our predictions of where the disordered regions are.

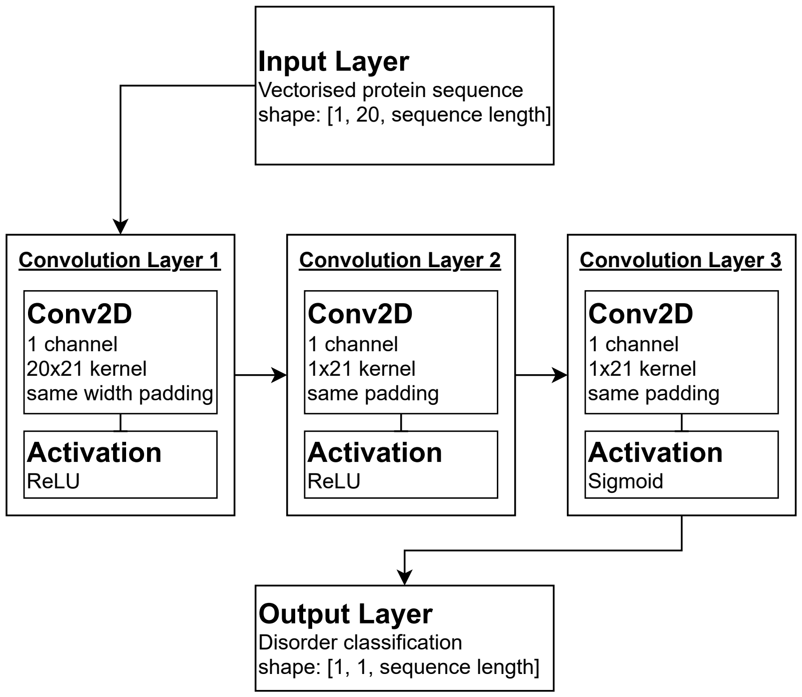


*Figure 5.1: A schematic representation of the first CNN. This neural network applies three convolutions to our vectorised protein sequence. These layers use two-dimensional convolution kernels. The first two layers use the ReLU activation function. Our final feature map is given to a sigmoid activation function to return disorder predictions.*

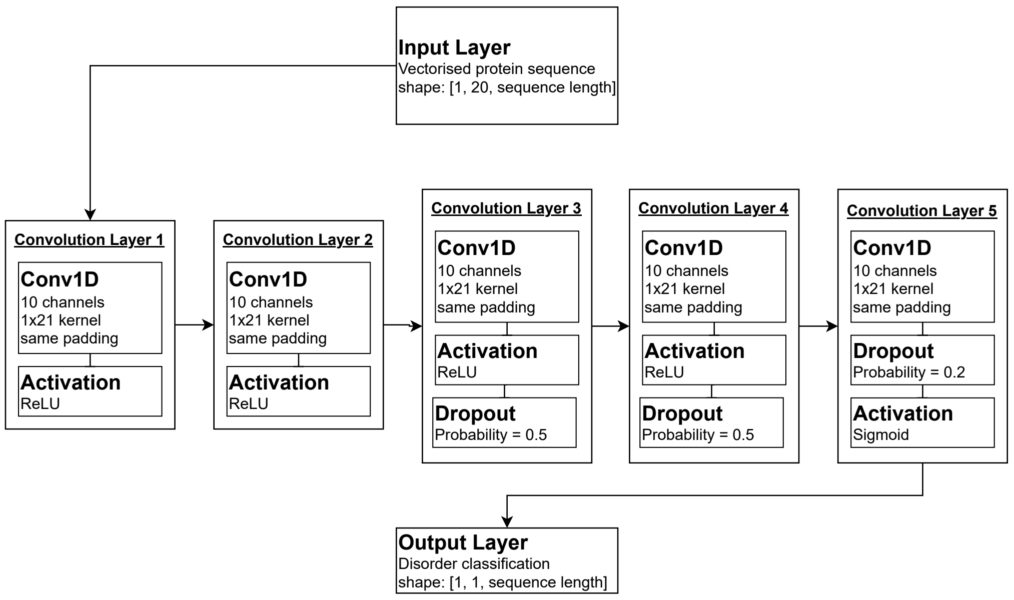


***Figure 5.2:*** *A schematic representation of the second CNN. This neural network applies five convolutions to our vectorised protein sequence. These layers use one-dimensional convolution kernels. Network layers use the ReLU activation function, until the final layer where sigmoid is applied. Dropout is used at the end of the network to add regularisation to more complex feature maps.*

Our other CNN uses multiple 1D convolution kernels and its network architecture is described in Figure 5.2. We used a convolution kernel of size 21, like the width of our 2D convolution kernels. We found using 10 channels as our hidden dimension was appropriate in helping our model learn, as keeping this channel size as 1 did not reduce training loss and using more than 10 channels prevented generalisability with our validation set. This model uses the same activation functions as our above model and the BCEWithLogitsLoss loss function. Finally, for this model, we employed dropout because it was overfitting and this regularisation added during training helped reduce the overfitting.

### 5.2.2 RNN

Our long short-term memory recurrent neural network (LSTM) is bidirectional, and its network architecture is described in Figure 5.3. We have set batch first as true, meaning that the input shape to our LSTM layer would be [batch size, sequence length, input features]. Our data loader used by the other models returns our sequences as [batch size, input features, sequence length], therefore this input is transposed to fit the required input shape. We did this using PyTorch's einsum functionality. The hidden states are important in the LSTM, and these are initialised to zero as this is the common approach when initialising the LSTM hidden states. From PyTorch documentation on LSTMs, using all zeros is also the default if no hidden state is created (Paszke et al. 2019). Furthermore, the hidden state is initialised every new forward pass, therefore it is re-initialised every sequence. This means our hidden state is used to capture dependencies of each sequence separately. Finally, our LSTM is bidirectional, therefore it processes the sequence in both directions: from the beginning to the end and from the end to the beginning, as discussed in our design of RNNs (Section 4.2.3).

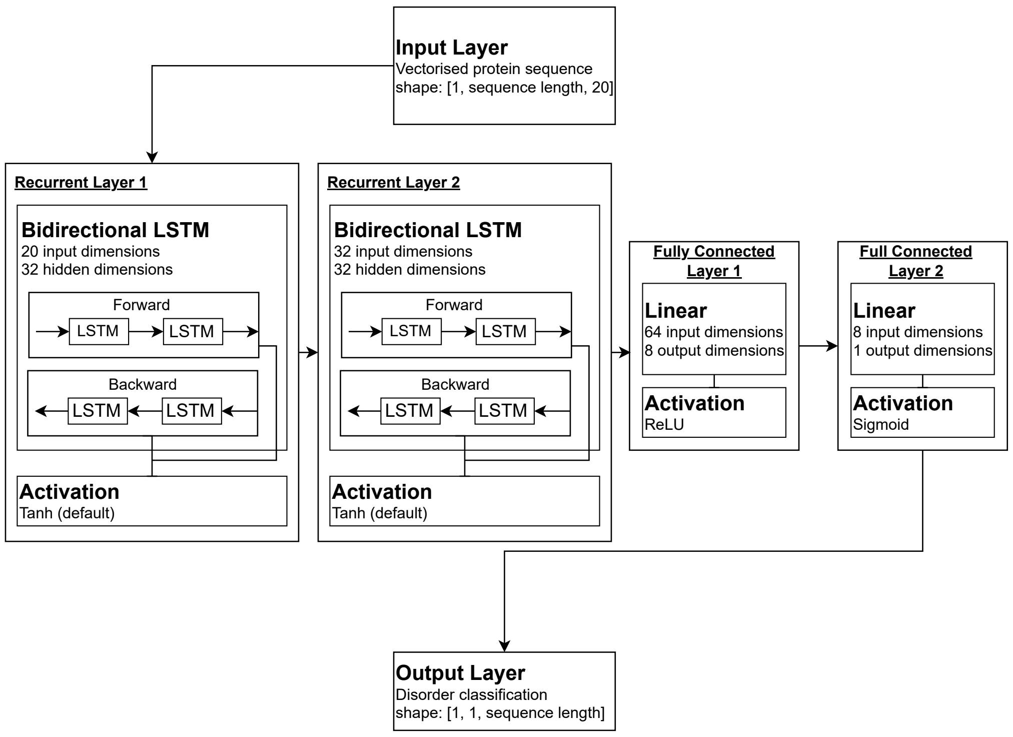


***Figure 5.3:*** *A schematic representation of the LSTM. This neural network applies two stacked bidirectional LSTM layers followed by two fully connected layers to our vectorised protein sequence. These LSTM layers use 32 hidden dimensions, meaning that the LSTM maintains 32 hidden states at once. Furthermore, our LSTM is bidirectional, meaning it will in fact maintain 64 states at once, hence the first fully connected layer takes in 64 input dimensions. Our final feature map is again given to a sigmoid activation function to return disorder predictions.*

## 5.3 Training the network

All of the models were trained using a training loop, which was run for many epochs, with each model requiring a different number of training epochs. We determined the required amount of epochs using manual inspection of training and validation loss curves, which monitors when a model would start to overfit. This gave us the maximum number of epochs that model would be trained for. Each model and its number of training epochs can be seen in Appendix A.1. Throughout training, calculations are made more efficient using the CUDA GPU (Nickolls et al. 2008), which can be run on Google Colaboratory. This lowered training times. To utilise CUDA within the training loop we first moved our tensors to this GPU device. Then, our PyTorch training loop is as expected: first evaluating the model's predictions through our BCEWithLogitsLoss loss function, setting our current gradient to zero, using backpropagation through time to calculate a gradient, then making a step with our optimiser from the gradient calculation. Furthermore, as we were using a batch size of one, we experimented with gradient accumulation, which imitates having a larger batch size (Bhattacharyya 20). This feature did not give improvements to our learning task, so we removed it. The optimiser we used for training was SGD and this was seen to work well with a standard momentum of 0.9, low learning rates and low weight decay values as discussed in our hyperparameter tuning section (Section 5.4).

## 5.4 Hyperparameter optimisation

Many options for the depth of these neural networks, the size of neural network layer parameters and optimisers were experimented on. We used the validation dataset to find trained models with a good fit to unseen data. From early manual inspection, we found that most models responded well to low learning rates and low weight decay. We optimised the hyperparameters for all of our final architectures and these final architectures are discussed in Section 5.2. RayTune (Liaw et al. 2018), a scalable hyperparameter tuning framework, was used to further tune our hyperparameters given a sensible range of values for the learning rate and regularisation term. We optimised these parameters for the SGD optimiser. Due to computational costs, we only considered 10 parameter combinations for every model. Table 5.2 demonstrates what parameters RayTune selected due to performing the best on the validation dataset by achieving the lowest loss. These values were used to train our final models used in our evaluation (Section 6).

***Table 5.2:*** *This table shows the optimal learning rate and weight decay parameters to be used with the SGD optimisation method given different models. Low learning rates and a variety of weight decays were chosen as the optimal parameters.*

| Model | Feature input | Learning rate | Weight decay (regularisation) |
|---|---|---|---|
| CNN (2D) | One-hot | 0.0005 | 0.01 |
| | PSSM | 0.0005 | 0.005 |
| CNN (1D) | One-hot | 0.0005 | 0.001 |
| | PSSM | 0.001 | 0.005 |
| RNN (LSTM) | One-hot | 0.001 | 0.001 |
| | PSSM | 0.001 | 0.0025 |

## 5.5 Django server

We have also implemented a small web application to make our solution accessible. This web server has two web pages. These visually follow the wireframes, and this user interface aims to be user-friendly. For our first page, we use a Django form which when submitted sends a POST request where the Django views conducts the classification of the disordered regions in the protein sequence. This form validates that the inputted sequence is a FASTA file and that every amino acid code is one of the 20 recognised amino acids. If these constraints are not satisfied, the classification will not continue as our solution cannot handle invalid amino acids. As we used Python throughout our deep learning evaluation, we have used code from our notebooks in a helper file to generate our features and make a prediction using our best performing model.

Our other page is the sequence visualisation indicating the IDRs using colour: where red implies disorder and blue implies order. The purpose of these colours is to improve usability and aid identification of the disordered regions. Furthermore, this colour palette is colour blind friendly which improves accessibility. We implemented the visualisation using a JavaScript function that creates a fake table so that the sequence characters are equally spaced, making the visualisation pleasant. Furthermore, this table has 60 columns, therefore is 60 characters long, because this is how many amino acid characters there are before the newline character in a FASTA file from UniProt. FASTA files can be any length; however, it is customary to use 60-70 characters before a newline, with the recommendation that "each line of sequence be no longer than 80 characters" (National Center for Biotechnology Information 1988). Therefore, this table size was chosen to maintain consistency with how protein sequences are read.

### 5.5.1 Containerising the web application

We containerised this web application by setting up a simple Docker, Nginx and Gunicorn configuration. Using this container provides a consistent environment for the application to run, which can help to avoid compatibility issues and is important for ensuring we have access to a suitable version of packages that work with PyTorch for example. Using containers means that a future external database could have a separate container, which promotes security and scalability. Using Nginx and Gunicorn will help a deployed web application handle multiple requests. Overall, this set-up can be used and further maintained to change scalability, performance and security of the web server.

# 6 Evaluation

## 6.1 Evaluation goal

Our goal is to evaluate how well different deep neural network architectures and approaches used in deep learning perform against each other when predicting where IDRs are within protein sequences. We have produced and trained six different models. The models are built using three different types of architectures: CNN (2D), CNN (1D), and RNN (LSTM) (Section 5.2). Each architecture is trained using two types of feature inputs: one-hot encoding and position-specific scoring matrix (PSSM) (Section 5.1.2). We now discuss these model's performance at protein disorder prediction with questions considering the *"effect of architecture on performance"*, the *"effect of input features on performance"* and how this overall has an effect to the classification of disorder.

## 6.2 How we will assess our deep neural networks

We will use metrics from Section 4.4, as these are often used in machine learning. The MCC is especially useful to ensure all our classes have been accurately classified as we have an imbalanced dataset. This MCC will prevent a model that predicts the majority class (ordered amino acids) from being more accurate due to simply predicting the most popular class and not learning meaningful features.

With the metrics, we evaluate the overall performance of each model given unseen protein sequences. As we are assessing multiple sequences, each of which has multiple labels, we will first calculate the micro-average of these metrics for each sequence, meaning that we count the total number of true positives, true negatives, false positives and false negatives for the predictions made for a single sequence, and calculate our metrics. We then take an average of these calculated metric scores over all the sequences to report our final scores. This was done because this evaluation should be thought of as the model's performance given a new sequence, therefore it is appropriate to take the average score over these sequences. Furthermore, this approach is used in the literature as protein sequences length and disorder content can vary greatly, meaning it would be difficult to compare metrics directly across different sequences. For example, the metrics could be skewed from an extremely large sequence that is predicted well.

It can be noted that we did calculate these metrics using the sum of the true positives, true negatives, false positives and false negatives over all of the sequences, instead of calculating it for each sequence, but similar conclusions were made about what architectures and approaches performed better. Calculating metrics in this fashion was valuable as with the total true positive, true negative, false positive and false negative values we constructed confusion matrices to see how often models predicted ordered or disordered labels correctly.

Finally, we will evaluate our model performance using the probability-based classifications discussed in our design (Section 4.4.1). With our probability values, we can plot receiver operating characteristic (ROC) curves and calculate the area under the ROC Curve (AUC). This was done so we can evaluate how different thresholds will perform at classifying disorder and how this will affect the true positive rate (TPR) and false positive rate (FPR). This gives us a further understanding about the models' performances and how we can modify the disorder threshold value to predict a disordered amino acid without creating unwanted behaviour where a high false positive rate occurs.

### 6.2.1 Statistical significance

To identify statistical significance between models, we conduct hypothesis tests, where we can state that there is a significant difference between two model's performances if our statistical test returns a p-value less than the significance level. We set this significance level to 0.05 as this is a standard cut-off. For our first hypothesis test, we will consider using the MCC values of each sequence in our test dataset, so we must use a statistical test for paired data. We focus on the significance of the MCC value as it is a strong metric for demonstrating the performance of classification on an imbalanced dataset, and this metric has been used for hypothesis testing in the relevant literature (Zhao and Kurgan 2022). An appropriate statistical test we can use is the paired t-test. We first validated the assumptions before conducting the paired t-test. We tested the assumption of normality using the Shapiro-Wilk test (Wikipedia contributors 2023*f*) before carrying out the paired t-test. If the data's underlying distribution is not normal, we conduct the non-parametric Wilcoxon signed-rank test (Wikipedia contributors 2023*g*). After all of our evaluations were conducted, we found that we could not satisfy this normality requirement, therefore we used the Wilcoxon signed-rank test. We can validate the assumptions for this test by ensuring the median of the differences is around zero, satisfying the requirement that the distribution must be symmetric. Furthermore, we plotted these distributions for visual inspection of this as well. These datasets also satisfy the assumption of independence as each sequence in a test dataset is unseen during training and unique.

With the Wilcoxon signed-rank test, we evaluate the statistical difference between feature inputs and neural network architectures. With these experimental factors we test how our input features affect the MCC outcome measurement, comparing this for each architecture. Then, we test how our neural network architectures affect the MCC metric, comparing them when they have the same input features. From this, we get many p-values that demonstrate the significance of different feature inputs and neural network architectures.

We also used the same statistical tests and followed the same method as above when evaluating the AUC value for each sequence. This will allow us to draw conclusions about the significance of the differences between models, using the probability-based predictions.

## 6.3 Classification of DisProt data

The withheld testing dataset we created from DisProt (Quaglia et al. 2022) was used to evaluate our different model performances first. Table 6.1 presents the evaluation metrics of these different models with this test set. From the table, we can see that the performance of the models does vary depending on the architecture and feature input used. Between the CNN models, the CNN using 1D convolutions within its architecture outperforms the 2D architecture for both types of feature inputs. The LSTM architecture also performs competitively given our models; giving similar results to the better CNN (1D).

In terms of feature input, we observe that PSSMs outperform one-hot encoding for every model. This suggests that the PSSM feature input may be more informative at expressing features about disordered regions within protein sequences. Furthermore, Table 6.2 evidences that the MCC of models using PSSM input is statistically significantly better; therefore, we conclude that PSSM feature input has helped to improve protein disorder predictions for this dataset.

Overall, the highest performing models are the CNN (1D) and LSTM architectures with PSSM feature input. This CNN has an accuracy of 0.6939, the highest accuracy for this test dataset and an MCC of 0.2422. This LSTM has an accuracy of 0.6934 and an MCC of 0.2490 which is the

***Table 6.1:*** *This table shows different models and their performance predicting protein disorder using different evaluation metrics. It is clear PSSM feature input produces better predictions than one-hot encoding. Furthermore, the model architectures of CNN (1D) and the LSTM are competitive against each other, whereas CNN (2D) (which has the simplest architecture) does not perform as well with this test dataset.*

| Model architecture | Feature input | Accuracy | Precision | Recall | Specificity | F1-score | MCC |
|---|---|---|---|---|---|---|---|
| CNN (2D) | One-hot | 0.4277 | 0.2737 | **0.9281** | 0.2493 | 0.3603 | 0.1380 |
| | PSSM | 0.4953 | 0.2814 | 0.8360 | 0.3595 | 0.3582 | 0.1433 |
| CNN (1D) | One-hot | 0.6831 | 0.3541 | 0.6109 | 0.6369 | 0.3748 | 0.2061 |
| | PSSM | **0.6939** | 0.3669 | 0.6514 | **0.6436** | 0.3980 | 0.2422 |
| RNN (LSTM) | One-hot | 0.6258 | 0.3328 | 0.7648 | 0.4977 | 0.4033 | 0.2041 |
| | PSSM | 0.6934 | **0.3734** | 0.6759 | 0.6285 | **0.4101** | **0.2490** |

***Table 6.2:*** *Results of the Wilcoxon signed-rank test on sequence MCC values for evaluating the significance of input features. A one-tailed test was used to see if the PSSM feature input had greater scores than one-hot feature input. We reject all null hypotheses as all p-values are lower than alpha (0.05).*

| Model architecture | Feature comparison | Wilcoxon Statistic | P-value |
|---|---|---|---|
| CNN (2D) | PSSM, One-hot | 53729.0 | $1.5272 \cdot 10^{-03}$ |
| CNN (1D) | PSSM, One-hot | 54035.0 | $1.9047 \cdot 10^{-03}$ |
| RNN (LSTM) | PSSM, One-hot | 56711.0 | $2.6974 \cdot 10^{-08}$ |

highest MCC score on this test dataset. The LSTM architecture also has the highest F1-score (0.4101). However, Table 6.3 shows that the improvement of MCC between the LSTM and CNN (1D) architecture is not statistically significant, therefore we accept the null hypothesis that there is no difference between these architectures. Furthermore, we do have strong statistical evidence to reject the null hypothesis when comparing these two architectures to our other CNN (2D): Table 6.3 demonstrates there is a significant difference between the quality of predictions made.

***Table 6.3:*** *Results of the Wilcoxon signed-rank test on sequence MCC values for evaluating different model architectures. A two-tailed test was used to see if models were statistically different. We could not reject the null hypothesis stating the LSTM and CNN (1D) have no difference, as these p-values are not less than alpha (0.05).*

| Feature input | Architecture comparison | Wilcoxon Statistic | P-value |
|---|---|---|---|
| One-hot encodings | CNN (1D), CNN (2D) | 30744.0 | $3.2158 \cdot 10^{-09}$ |
| | LSTM, CNN (2D) | 23735.0 | $9.8264 \cdot 10^{-13}$ |
| | LSTM, CNN (1D) | 44539.0 | $7.7522 \cdot 10^{-01}$ |
| PSSMs | CNN (1D), CNN (2D) | 25119.0 | $7.6701 \cdot 10^{-17}$ |
| | LSTM, CNN (2D) | 23241.0 | $2.1073 \cdot 10^{-19}$ |
| | LSTM, CNN (1D) | 43391.0 | $3.2658 \cdot 10^{-01}$ |

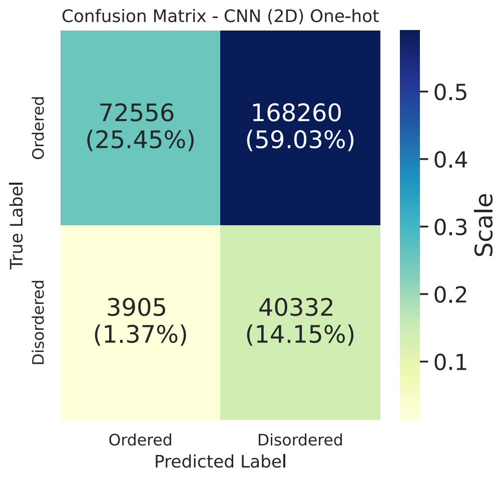
### 6.3.1 Model confusion

We can look at each model with more granularity by evaluating the classification report of each model, given by the scikit-learn library (Pedregosa et al. 2011). This looks at the overall true positives, true negatives, false positives and false negatives rather than regarding them for each sequence. This gives us similar information to Table 6.1, and we can also generate a confusion matrix to see how much order and disorder was misclassified by different models.

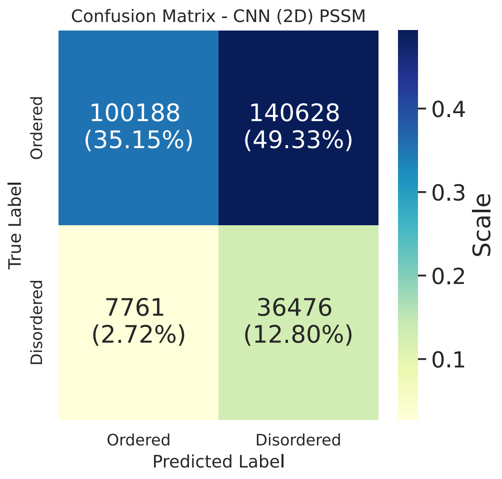
**CNNs**

For our CNNs using the 2D convolutions within its architecture, our confusion matrices (Figure 6.1a and Figure 6.1b) show that this model is often classifying many ordered residues as disordered. Using the PSSM features lowers this misclassification, but there is still mostly disordered amino acids being predicted despite the dataset containing a majority of ordered amino acids. This classification distribution helps us understand why the recall for this model is so high (Table 6.1), as it is much less likely to misclassify a disordered amino acid because it attempts to classify more amino acids as disordered.

For our CNNs using the 1D convolutions within its architecture, our confusion matrices (Figure 6.2a and Figure 6.2b) show there are better classifications being made than the previous CNN architecture. We can see a better balance between the two classes, with a lot more ordered amino acids being classified correctly than incorrectly. We can also see a slight improvement to the predictions made when we use the PSSMs as our feature input. This agrees with our analysis of the test set (Section 6.3), demonstrating using PSSM features improves the CNN models predictions.



*(a) One-hot encoding confusion matrix.*

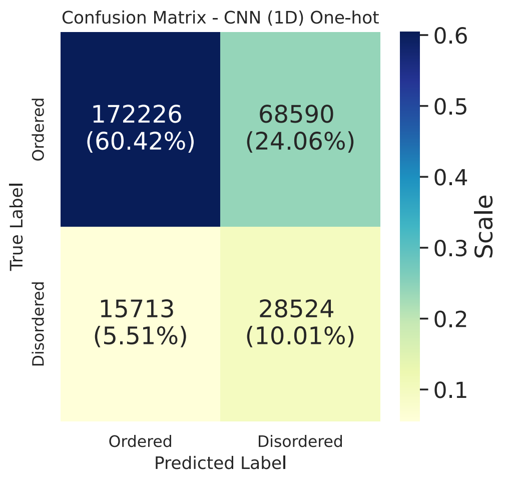


*(b) PSSM confusion matrix.*

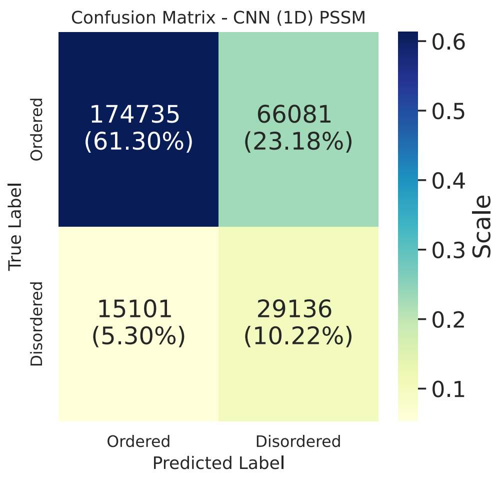
***Figure 6.1:*** *Confusion matrices for CNN with 2D convolution kernels within its architecture. (a) shows the confusion matrix for the model using one-hot encoding feature input. (b) shows the confusion matrix for the model using PSSM feature input. We see this models architecture performs poorly; misclassifying many ordered amino acids. It is clear that despite a high recall when predicting disordered regions, this architecture is not very precise, therefore it is not very sensible.*

**RNNs**

Finally, our RNNs, using the LSTM architecture can be regarded as the best model architecture for predicting IDRs in the protein sequences of our test dataset. Our confusion matrices (Figure 6.3a and Figure 6.3b) help demonstrate how many ordered and disordered amino acids were misclassified by models using the LSTM architecture. Considering the different feature inputs, our one-hot encoding method tends to classify more disordered amino acids correctly at the cost of misclassifying more ordered amino acids as disordered. Whereas, the PSSM inputs have less misclassified ordered amino acids, but the model has a weaker recall and has predicted less disordered amino acids correctly.

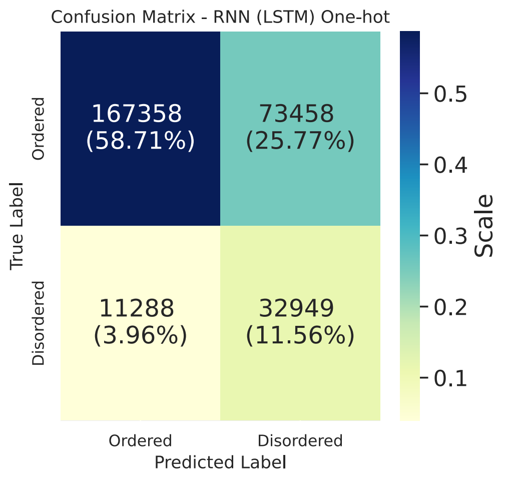


*(a) One-hot encoding confusion matrix.*

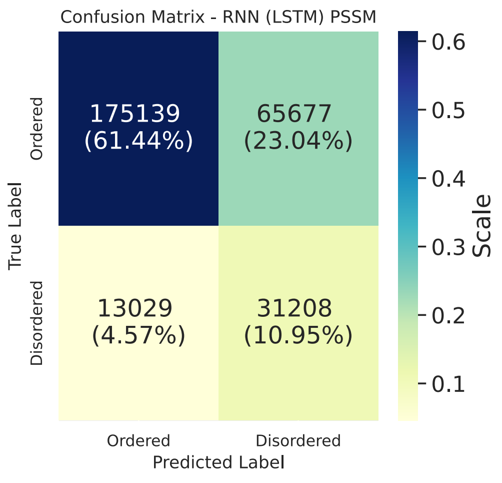


*(b) PSSM confusion matrix.*

***Figure 6.2:*** *Confusion matrices for CNN with 1D convolution kernels within its architecture. (a) shows the confusion matrix for the model using one-hot encoding feature input. (b) shows the confusion matrix for the model using PSSM feature input. We see this models architecture has a sensible precision and recall. Furthermore, using PSSM features reduces misclassification, however only marginally. Despite this, the model using PSSMs is significantly better (Table 6.2).*



*(a) One-hot encoding confusion matrix.*



*(b) PSSM confusion matrix.*

***Figure 6.3:*** *Confusion matrices for RNN using an LSTM architecture. (a) shows the confusion matrix for the model using one-hot encoding feature input. (b) shows the confusion matrix for the model using PSSM feature input. Using this LSTM architecture results in the lowest number of misclassifications. The different feature inputs are seen to misclassify different classes.*

### 6.3.2 Analysis of probability-based predictions

Using the DisProt test dataset, Figure 6.4 demonstrates that our LSTM with one-hot encoding feature input is the best model in terms of trade-off between the true positive rate (TPR) and false positive rate (FPR). The two models using the LSTM architecture have a higher area under the ROC curve (AUC) (Table 6.4), showing the LSTM is better at sensibly identifying IDRs within

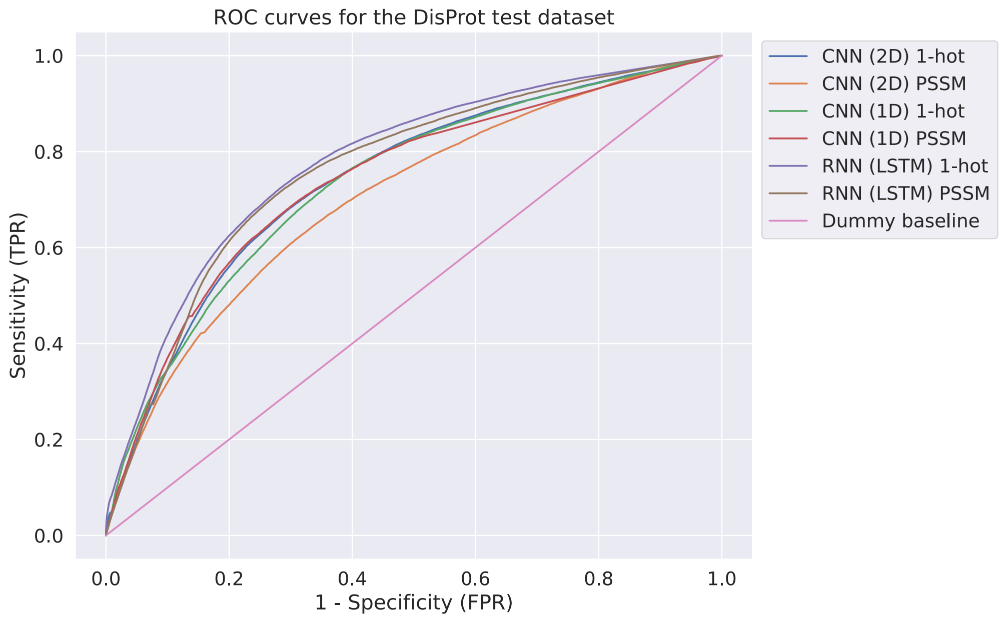


***Figure 6.4:*** *ROC curves of probability-based IDR predictions for our 6 models on the withheld DisProt test data. It is clear the models with LSTM architectures have the best performance given a TPR and FPR trade off. Furthermore, every model performs above the dummy baseline.*

***Table 6.4:*** *This table shows different models and their performance regarding AUC. These scores are calculated from the ROC curves generated in Figure 6.4 using the DisProt test data.*

| Model architecture | CNN (2D) | | CNN (1D) | | RNN (LSTM) | |
|---|---|---|---|---|---|---|
| Feature input | One-hot | PSSM | One-hot | PSSM | One-hot | PSSM |
| AUC | 0.7426 | 0.7043 | 0.7379 | 0.7403 | 0.**7794** | 0.7635 |

protein sequences. These improved metrics were seen for the LSTM in our evaluation of other metrics as well, such as MCC and F1-score. The CNN models have a slightly lower performance than the LSTMs when using this metric; with the 2D CNN with PSSM input having the worst performance at correctly classifying disordered amino acids confidently.

From conducting a Wilcoxon signed-rank test with these model's AUC scores (Table 6.5 and Table 6.6), we reach a number of conclusions. We cannot conclude there is a significant improvement between the one-hot encoding and PSSM input for the LSTM architecture, but for the CNN (1D) model there is a significant improvement when using different feature inputs, concluding PSSM feature input improves this metric for this model architecture. Furthermore, different architectures have different performances dependent on the feature input when using this metric. For example the LSTM and CNN (2D) having no significant difference when using one-hot encodings, but a clear statistical difference when using these architectures with PSSM input. Therefore, we will not make a conclusion about the best performing architectures with the AUC metric.

Finally, all of these classifiers perform better than a random dummy baseline. This can be seen in Figure 6.4 since all the ROC curves are above the dummy baseline.

***Table 6.5:*** *Results of the Wilcoxon signed-rank test on sequence AUC values for evaluating the significance of input features. A one-tailed test was used to see if the PSSM feature input had greater scores than one-hot feature input. We reject the null hypothesis for the CNN (1D) model architecture as this p-value is lower than alpha (0.05).*

| Model architecture | Feature comparison | Wilcoxon Statistic | P-value |
|---|---|---|---|
| CNN (2D) | PSSM, One-hot | 32404.0 | $1.0000 \cdot 10^{+00}$ |
| CNN (1D) | PSSM, One-hot | 54441.0 | $1.1425 \cdot 10^{-03}$ |
| RNN (LSTM) | PSSM, One-hot | 50071.0 | $1.5568 \cdot 10^{-01}$ |

***Table 6.6:*** *Results of the Wilcoxon signed-rank test on sequence AUC values for evaluating different model architectures. A two-tailed test was used to see if models were statistically different. We could not reject the null hypothesis stating the LSTM and CNN (1D) have no difference when PSSM input is used, or the hypothesis stating the LSTM and CNN (2D) have no difference when one-hot encoding input is used, as these p-values are not less than alpha (0.05).*

| Feature input | Architecture comparison | Wilcoxon Statistic | P-value |
|---|---|---|---|
| One-hot encodings | CNN (1D), CNN (2D) | 35839.0 | $3.4921 \cdot 10^{-05}$ |
| | LSTM, CNN (2D) | 43252.0 | $1.3128 \cdot 10^{-01}$ |
| | LSTM, CNN (1D) | 37561.0 | $3.9353 \cdot 10^{-04}$ |
| PSSM | CNN (1D), CNN (2D) | 33839.0 | $1.6086 \cdot 10^{-07}$ |
| | LSTM, CNN (2D) | 31908.0 | $2.3268 \cdot 10^{-09}$ |
| | LSTM, CNN (1D) | 44209.0 | $2.8746 \cdot 10^{-01}$ |

## 6.4 Classification of CASP10 data

The CASP10 dataset has been used to assess many protein disorder prediction tools. There are 94 targets (protein sequences) in this dataset that were used for analysis, as some other sequences proposed in this dataset were removed by the organisers before or during the competition (Moult et al. 2014). For this dataset, we generate one-hot encoded and PSSM features similarly to our DisProt dataset. Furthermore, no sequences in this dataset were invalid for our vectorisation process and model input; therefore, no sequences had to be removed.

We performed a similar analysis on the CASP dataset as we have done using the DisProt withheld test dataset. We used the same model architectures and features as above but trained our models using our entire DisProt dataset before making predictions on the CASP dataset. Table 6.7 presents the evaluation metrics of our different models' performances on the CASP10 dataset.

***Table 6.7:*** *This table shows different models and their performance predicting protein disorder using different evaluation metrics on the CASP10 dataset.*

| Model architecture | Feature input | Accuracy | Precision | Recall | Specificity | F1-score | MCC |
|---|---|---|---|---|---|---|---|
| CNN (2D) | One-hot | 0.6440 | 0.1403 | 0.4980 | 0.6466 | 0.1862 | 0.1296 |
| | PSSM | 0.4309 | 0.1108 | 0.6715 | 0.3967 | 0.1691 | 0.0977 |
| CNN (1D) | One-hot | **0.7503** | 0.1362 | 0.3489 | **0.7647** | 0.1714 | 0.1098 |
| | PSSM | 0.7148 | **0.1771** | 0.5255 | 0.7096 | **0.2376** | **0.1828** |
| RNN (LSTM) | One-hot | 0.5296 | 0.1265 | 0.6450 | 0.4869 | 0.1936 | 0.1191 |
| | PSSM | 0.6590 | 0.1707 | **0.5640** | 0.6479 | 0.2260 | 0.1688 |

From this table, we see a similar performance difference between the architectures and feature inputs as the previous unseen test dataset. Again, the CNN using 2D convolutions within its architecture has the worst performance. This may be due to it having the simplest architecture. For the CNN using 1D convolutions and the LSTM it is clear again that using the PSSMs as feature input does improve the prediction quality of protein disorder; however this improvement is not statistically different to one-hot encoding features as shown in Table 6.8. Nevertheless, this lack of significance is most likely due to the small number of sequences in this benchmark dataset.

The LSTM and CNN (1D) models are once again comparably competitive, with the CNN model achieving the highest MCC of 0.1828 and the highest F1-score of 0.2376. It is also interesting to note that the better CNN architecture, a model using one-hot encoding has the highest accuracy and specificity measure. Overall, we claim the CNN (1D) using PSSM features has the leading performance as it has the strongest distribution of metrics. However, we can see that both the CNN (1D) and LSTM perform similarly with the CASP10 dataset, and Table 6.9 shows there is no significant difference between them. This makes it more difficult to justify a best architecture and has demonstrated the capability of different network architectures at the task of disorder prediction.

***Table 6.8:*** *Results of the Wilcoxon signed-rank test on CASP10 sequence MCC values for evaluating the significance of input features. A one-tailed test was used to see if the PSSM feature input had greater scores than one-hot feature input. We cannot reject the null hypotheses as all p-values are greater than alpha (0.05).*

| Model architecture | Feature comparison | Wilcoxon Statistic | P-value |
|---|---|---|---|
| CNN (2D) | PSSM, One-hot | 2158.0 | $6.1062 \cdot 10^{-01}$ |
| CNN (1D) | PSSM, One-hot | 2464.0 | $1.0284 \cdot 10^{-01}$ |
| RNN (LSTM) | PSSM, One-hot | 2182.0 | $5.0535 \cdot 10^{-01}$ |

***Table 6.9:*** *Results of the Wilcoxon signed-rank test on CASP10 sequence MCC values for evaluating different model architectures. A two-tailed test was used to see if models were statistically different. We reject the null hypothesis stating the CNN (1D) and CNN (2D) with PSSM features have no difference, as this p-value is less than alpha (0.05).*

| Feature input | Architecture comparison | Wilcoxon Statistic | P-value |
|---|---|---|---|
| One-hot encodings | CNN (1D), CNN (2D) | 2120.0 | $6.7140 \cdot 10^{-01}$ |
| | LSTM, CNN (2D) | 1990.0 | $5.6179 \cdot 10^{-01}$ |
| | LSTM, CNN (1D) | 1893.0 | $5.3417 \cdot 10^{-01}$ |
| PSSM | CNN (1D), CNN (2D) | 1598.0 | $1.6727 \cdot 10^{-02}$ |
| | LSTM, CNN (2D) | 1720.0 | $1.0278 \cdot 10^{-01}$ |
| | LSTM, CNN (1D) | 2033.0 | $6.7979 \cdot 10^{-01}$ |

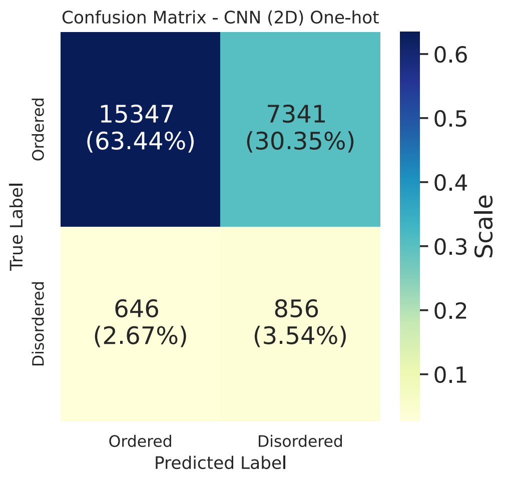
### 6.4.1 Model confusion (external CASP10 data)

Using our different model architectures and feature inputs to make predictions on the CASP dataset, we have seen similarities and differences to the confusion shown about predictions on our test dataset (Section 6.3.1).

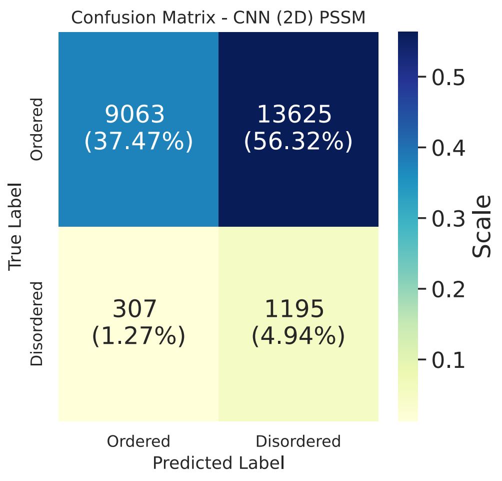
#### CNNs

For the CNN architecture with 2D convolution kernels, the two models that use different feature inputs have very different confusion matrices. With one-hot encoding features, our model classifies 57% of the disordered amino acids correctly (Figure 6.5a), whereas the model using PSSM features classifies 80% of these disordered amino acids correctly at the expense of classifying more than half of the ordered amino acids incorrectly (Figure 6.5b). This is similar to this architecture's performance on the DisProt test dataset (Section 6.3.1), where we saw that it often over-predicted disordered amino acids. This means this model has likely been unable to capture the unique intricate features that represent IDRs.

The CNN architecture with 1D convolution kernels classifies only approximately half of the disordered amino acids correctly for both feature inputs, which can be seen by Figure 6.6a and Figure 6.6b. We can see the CNN with one-hot encoding features, which is our most accurate model (Table 6.7), performs worse at classifying disordered amino acids, and the CNN with PSSM features performs worse at classifying ordered amino acids. Table 6.8 demonstrates this difference is not significant enough to determine the least confused model.



*(a) One-hot encoding confusion matrix (CASP10).*

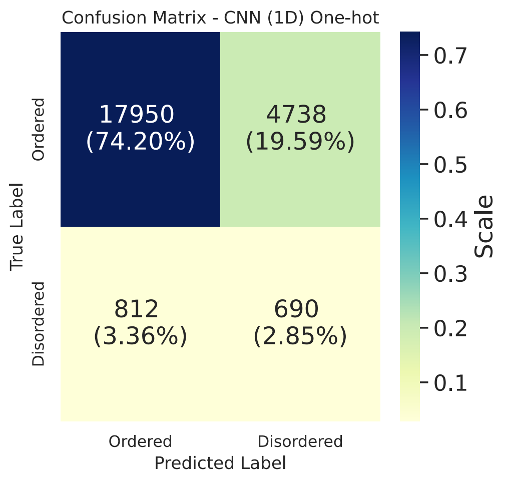


*(b) PSSM confusion matrix. (CASP10)*

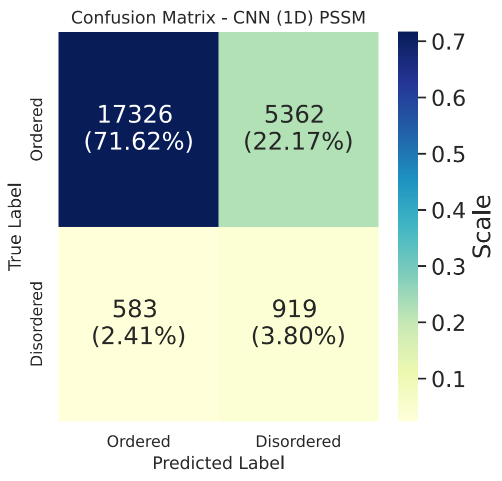
*Figure 6.5: Confusion matrices for CNN with 2D convolution kernels within its architecture. (a) shows the confusion matrix for the model using one-hot encoding feature input. (b) shows the confusion matrix for the model using PSSM feature input. We see this models architecture performs reasonably well with one-hot encoded features, but misclassifies many ordered amino acids when using PSSMs.*

**RNNs**

Our RNN models have less confusion. For the LSTM when one-hot encoding is used, we can see slight overestimations of disorder again. However, despite overestimating disordered labels, this model is often a lot less confused about ordered amino acids (Figure 6.7a) when we compare this model to our other model that overestimates disorder (the 2D CNN). From our confusion matrix of the LSTM with PSSM features (Figure 6.7b) we can see that it can correctly predict 63% of the disordered amino acids, which is better than the competitive CNN (1D) model (with 61%). However, the proportion of correctly predicted disordered amino acids is still lower than our DisProt dataset for this model. Overall, the LSTM does get more confused when classifying ordered amino acids when we compare it to our CNN (1D) models; however, in general we can see the LSTM architecture can correctly predict more disordered amino acids while still reasonably predicting ordered amino acids correctly. This is important in our prediction task, and this is useful to see the LSTM architecture exhibits the lowest level of confusion in the CASP10 dataset task.

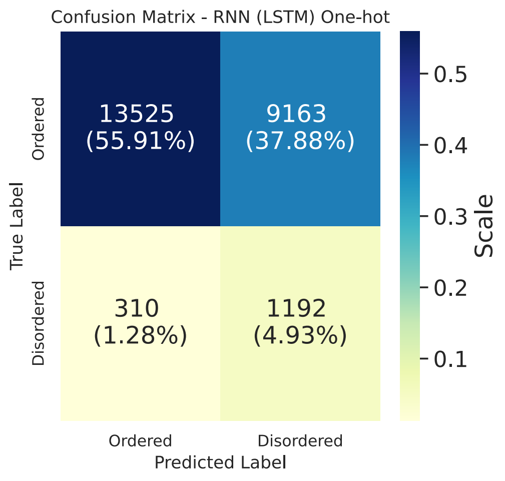


*(a) One-hot encoding confusion matrix (CASP10).*

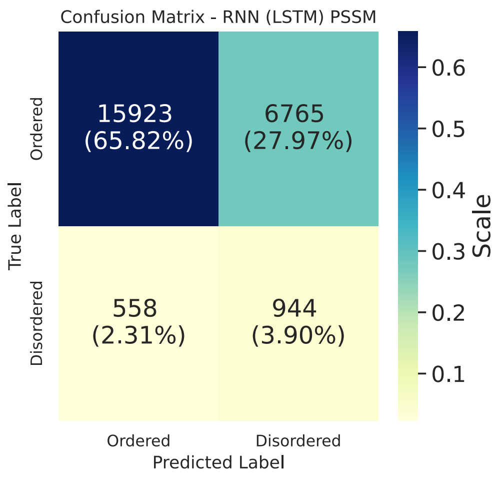


*(b) PSSM confusion matrix (CASP10).*

***Figure 6.6:*** *Confusion matrices for CNN with 1D convolution kernels within its architecture. (a) shows the confusion matrix for the model using one-hot encoding feature input. (b) shows the confusion matrix for the model using PSSM feature input. Using the PSSM feature input has decreased overall confusion, however we cannot conclude the difference between these two models is statistically significant (Table 6.8).*



*(a) One-hot encoding confusion matrix (CASP10).*



*(b) PSSM confusion matrix (CASP10).*

***Figure 6.7:*** *Confusion matrices for RNN using an LSTM architecture. (a) shows the confusion matrix for the model using one-hot encoding feature input. (b) shows the confusion matrix for the model using PSSM feature input. We see this models architecture experiencing the least overall confusion again.*

### 6.4.2 Analysis of probability-based predictions (external CASP10 data)

When analysing ROC curves with our CASP10 dataset, we see the LSTM using one-hot encoding features achieve the best AUC (Table 6.10). This is interesting because it is a different conclusion compared to the other results showing PSSMs improve predictions, but also agrees with the AUC analysis of our DisProt test dataset (Section 6.3.2). From inspection of the ROC curve graph (Figure 6.8), we see that the CNN with 2D convolutions in its architecture has the worst performance on this dataset. Using our Wilcoxon signed-rank test, we could not reject our null

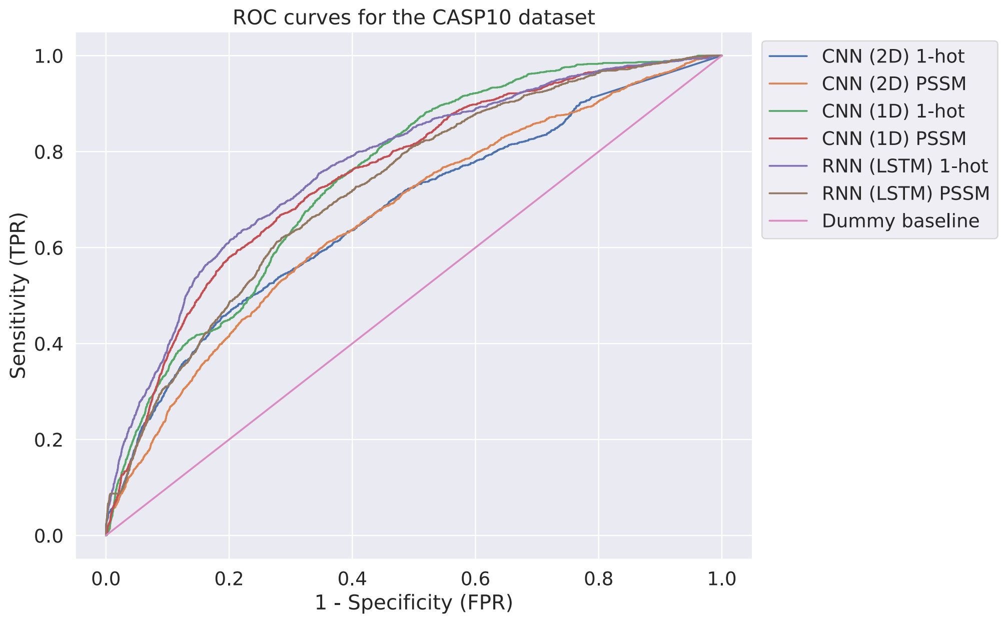


***Figure 6.8:*** *ROC curves of probability-based IDR predictions for our 6 models on the CASP10 dataset. The LSTM model using one-hot encoding has the best performance given a TPR and FPR trade off. Both CNN (2D) models perform the poorest. Furthermore, every model does perform above the dummy baseline.*

***Table 6.10:*** *This table shows different models and their performance regarding AUC. These scores are calculated from the ROC curves generated in Figure 6.8 using the CASP10 dataset.*

| Model architecture | CNN (2D) | | CNN (1D) | | RNN (LSTM) | |
|---|---|---|---|---|---|---|
| Feature input | One-hot | PSSM | One-hot | PSSM | One-hot | PSSM |
| AUC | 0.6707 | 0.6636 | 0.7458 | 0.7518 | **0.7701** | 0.7232 |

hypotheses that feature input or model architecture resulted in significantly different AUC scores, and the table of these test results are included in the appendix (Appendix A.2.1).

Finally, we can conclude that all of our models outperform a random classifier from inspection of the ROC curve graph (Figure 6.8), where each models ROC curve is above the dummy baseline.

### 6.4.3 Comparison of other solutions

Monastyrskyy et al. (2014) released an assessment of protein disorder region predictions in CASP10, highlighting the performance of state-of the art solutions. Table 6.11 shows our top performing models, LSTM and CNN (1D) with PSSM features, against top performing models that use different architectures. We see from this table that our models grossly overpredict disorder, which can be seen by a much larger false positive value. This has had an effect on other metrics such as the precision and MCC scores. These state-of-the-art models have a common use of SVMs, and some use ensemble approaches alongside other machine learning methods. These outperform our deep learning strategies. However, from this comparison it is clear that we could increase regularisation to prevent our models likelihood of predicting disorder. Furthermore, our upweighted loss function could be trailled with different weights instead of the ratio difference of labels to assess if this could lower overprediction of disorder when given unseen test data.

*Table 6.11: This table is adapted from Table 2 in the 'Assessment of model performance on the CASP10 dataset' (Monastyrskyy et al. 2014). It compares our top two models (CNN (1D) PSSM and RNN (LSTM) PSSM) with some of the top entries in the CASP10 competition.*

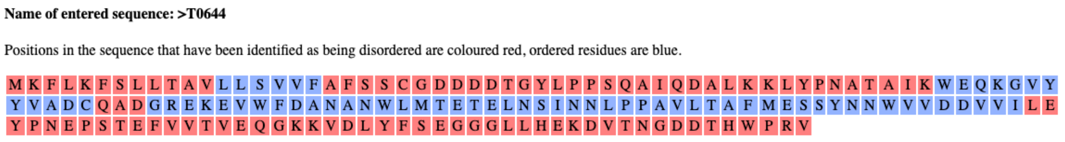
| Name | TP | FP | TN | FN | Precision | Accuracy | MCC | AUC (ROC) |
|---|---|---|---|---|---|---|---|---|
| Prdos-CNF | 657 | 287 | 22401 | 845 | 0.696 | 0.712 | 0.529 | **0.907** |
| DISOPRED3 | 607 | 201 | 22487 | 895 | **0.751** | 0.698 | **0.531** | 0.897 |
| biomine_dr_mixed | 628 | 368 | 22320 | 874 | 0.631 | 0.701 | 0.488 | 0.890 |
| POODLE | 980 | 2064 | 20624 | 522 | 0.322 | **0.781** | 0.409 | 0.875 |
| CNN (1D) PSSM | 919 | 5362 | 17326 | 583 | 0.177 | 0.715 | 0.183 | 0.752 |
| RNN (LSTM) PSSM | 944 | 6765 | 15923 | 558 | 0.171 | 0.660 | 0.169 | 0.723 |

## 6.5 Evaluation of the web server

With our web server, we have fulfilled the minimum viable product by assessing our proposed MoSCoW requirements (Section 3.7). The web interface follows from the designed wireframes (Section 4.5.1) and can be used locally. Predictions made on this web server use our model with the LSTM architecture.

We evaluate our web server's prediction graphic in comparison to the DISOPRED3 prediction method (Jones and Cozzetto 2015), using the PSIPRED server (Buchan and Jones 2019). We compare these using the first protein sequence in the CASP10 dataset (T0644.dr, which is found in Appendix A.2.2 alongside its true disorder label).

Our server's classifier predicts many of these amino acids as disordered (Figure 6.9), whereas DISOPRED3 misclassifies only a few amino acids (Figure 6.10). This misclassification of ordered amino acids is apparent through our earlier evaluation (Section 6.4.3). This informs us that we should refine our models in the future to increase the quality of disorder predictions, which will benefit bioscience researchers. Furthermore, we believe our highlighting of disorder using full colour accentuates the disorder, improving readability of the disorder visualisation.


**Name of entered sequence: >T0644**

Positions in the sequence that have been identified as being disordered are coloured red, ordered residues are blue.

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

*Figure 6.9: Prediction made for CASP target T0644 using the LSTM model on our web server. We see many disordered amino acids at both the start and end of this sequence highlighted in red. A full screenshot of this web page can be found in Appendix A.2.2.*

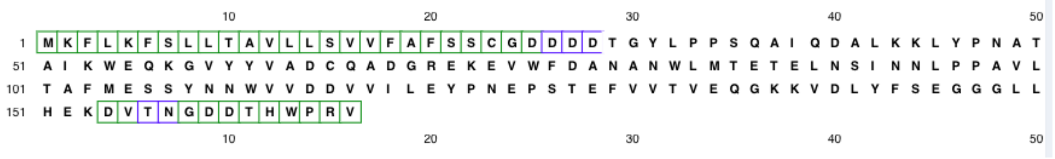

```
                         10                        20                        30                        40                        50
  1   M K F L K F S L L T A V L L S V V F A F S S C G D D D D T G Y L P P S Q A I Q D A L K K L Y P N A T
 51   A I K W E Q K G V Y Y Y V A D C Q A D G R E K E V W F D A N A N W L M T E T E L N S I N N L P P A V L
101   T A F M E S S Y N N W V V D D V V I L E Y P N E P S T E F V V T V E Q G K K V D L Y F S E G G G L L
151   H E K D V T N G D D T H W P R V
                         10                        20                        30                        40                        50
```

*Figure 6.10: Prediction made for CASP target T0644 using DISOPRED3. We see disordered amino acids at the start and end of this sequence highlighted with coloured borders. Full web page found in Appendix A.2.2.*

# 7 Conclusion

## 7.1 Summary

This project explored how different deep learning approaches and architectures perform at predicting protein disorder, given a protein sequence. We considered different feature encoding methods to represent protein sequences: one-hot encoding, and PSSMs which provides information about the evolution of a protein. We developed CNNs that used either 1D convolution kernels or 2D convolution kernels throughout the network, as well as a RNN that used the LSTM architecture. These were trained using different feature encoding approaches. We put our trained model within a web server, which makes it easier for bioscience researchers to evaluate protein sequences without interacting directly with the underlying technology. The predictions made by the model are displayed on a web page, which highlights disordered regions for a given sequence using a sequence-based visualisation.

Our evaluation determined the effect different feature inputs and model architectures had on correctly identifying protein disorder. We found that models using PSSM feature input perform significantly better than models using one-hot encoding. We also demonstrated that LSTM and CNN (1D) architectures are similarly competitive, which highlights the importance of considering multiple architectures for the task. Furthermore, these architectures performed significantly better than our other CNN (2D) that had a simpler architecture. We used the Wilcoxon signed-rank test with MCC and ROC AUC values, to support these findings.

However, we found that even our better models struggled to precisely classify disorder within the sequence. This was apparent as our deep neural networks had a much higher false positive rate than other state-of-the-art methods when classifying sequences in the CASP10 benchmark dataset. This could be due to our weighted loss function that was used to neutralise the imbalance of disordered amino acid labels. This weighting may have been too high of a penalty, resulting in more misclassifications of the ordered class.

Overall, we have not achieved the same performance as other prediction tools created for protein disorder prediction; however, we have showed that deep learning methods can be applied to the task. In addition, we demonstrated our models using PSSM feature input have a significant performance improvement, and both LSTM and CNN (1D) architectures can capture some features well that are used to represent IDRs.

## 7.2 Future work

From our evaluation, we could first investigate more modifications to the loss function and consider alternative weighting strategies with this loss function. Moreover, focusing on carefully learning disordered features so that the disorder class was precisely predicted in future models when given more unseen test data. This may make our solution more competitive against state-of-the-art methods.

As we saw an improvement using more complex features, we could consider including other input features. Some state-of-the-art methods have a significant number of complex features; for example, SPOT-DISORDER2 has 73 input features, including evolutionary PSSMs alongside other structural properties such as secondary structure and protein contact map information (Hanson et al. 2019).

As well as further modifications to our model architectures, we could use the models that complement each other in an ensemble approach. This ensemble would combine our different deep neural networks, and let us analyse if increasing the complexity of our solution by using different networks together could provide significant change in the performance of our models.

Finally, we aim to extend the web server, satisfying more requirements. The web server interface could be improved by adding more advanced visualisation tools, such as graphs demonstrating the probability-based value predictions. A more technical improvement to the web server would be to include an external database of generated PSSMs, to allow for efficiency when using models requiring PSSM input. The BLAST query servers are shared; therefore, our server cannot use these resources constantly. By creating an efficient store of PSSM data given previous users' queries, we can lower the computational expense generated by the web server.

# A | Appendices

## A.1 Implementation

### A.1.1 Training epochs

***Table A.1:*** *Training epochs used for each model. The CNNs required much less training epochs than the LSTM before overfitting. These values were selected from inspection of validation loss not improving during training.*

| Model architecture | Feature input | Number of training epochs |
|---|---|---|
| CNN (2D) | One-hot | 40 |
| | PSSM | 30 |
| CNN (1D) | One-hot | 40 |
| | PSSM | 20 |
| RNN (LSTM) | One-hot | 100 |
| | PSSM | 100 |

## A.2 Evaluation

### A.2.1 Evaluation Tables

**Wilcoxon-signed rank test for CASP10 AUC values.** This Wilcoxon-signed rank test was performed on our CASP10 AUC values, comparing feature inputs and model architectures, but no null hypotheses stating there was no difference between these approaches or architectures could be rejected.

***Table A.2:*** *Results of the Wilcoxon signed-rank test on CASP10 sequence AUC values for evaluating the significance of input features. A one-tailed test was used to see if the PSSM feature input had greater scores than one-hot feature input. We reject the null hypothesis for the CNN (1D) model architecture as this p-value is lower than alpha (0.05).*

| Model architecture | Feature comparison | Wilcoxon Statistic | P-value |
|---|---|---|---|
| CNN (2D) | PSSM, One-hot | 519.0 | $2.9313 \cdot 10^{+01}$ |
| CNN (1D) | PSSM, One-hot | 561.0 | $1.4696 \cdot 10^{-01}$ |
| RNN (LSTM) | PSSM, One-hot | 469.0 | $5.2144 \cdot 10^{-01}$ |

***Table A.3:*** *Results of the Wilcoxon signed-rank test on sequence AUC values for evaluating different model architectures. A two-tailed test was used to see if models were statistically different. We could not reject the null hypothesis stating the LSTM and CNN (1D) have no difference when PSSM input is used, or the hypothesis stating the LSTM and CNN (2D) have no difference when one-hot encoding input is used, as these p-values are not less than alpha (0.05).*

| Feature input | Architecture comparison | Wilcoxon Statistic | P-value |
|---|---|---|---|
| One-hot encodings | CNN (1D), CNN (2D) | 419.0 | $5.2204 \cdot 10^{-01}$ |
| | LSTM, CNN (2D) | 440.0 | $6.9760 \cdot 10^{-01}$ |
| | LSTM, CNN (1D) | 418.0 | $5.1427 \cdot 10^{-01}$ |
| PSSM | CNN (1D), CNN (2D) | 467.0 | $9.4760 \cdot 10^{-01}$ |
| | LSTM, CNN (2D) | 461.0 | $8.9069 \cdot 10^{-01}$ |
| | LSTM, CNN (1D) | 452.0 | $8.0645 \cdot 10^{-01}$ |

### A.2.2 Web server evaluation

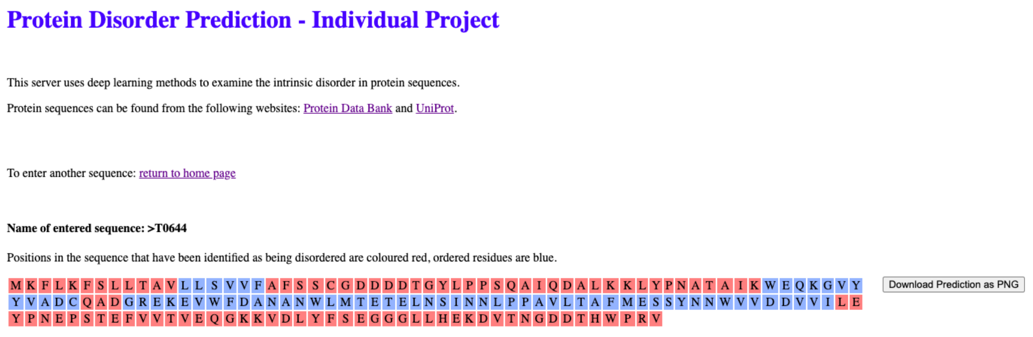
**Test protein sequence true label** Below is the protein sequence and disorder label for target T0644 from the CASP10 disordered protein dataset. This file T0644.dr can be obtained from the online zipped folder of CASP10 targets.

```
PFRMAT      DR
TARGET      T0644
M     D     1
K     D     2
F     D     3
L     D     4
K     D     5
F     D     6
S     D     7
L     D     8
L     D     9
T     D     10
A     D     11
V     D     12
L     D     13
L     D     14
S     D     15
V     D     16
V     D     17
F     D     18
A     D     19
F     D     20
S     D     21
S     D     22
C     D     23
G     D     24
D     D     25
D     O     26
D     O     27
D     O     28
T     O     29
G     O     30
Y     O     31
L     O     32
P     O     33
P     O     34
S     O     35
Q     O     36
A     O     37
I     O     38
Q     O     39
D     O     40
A     O     41
L     O     42
K     O     43
K     O     44
L     O     45
Y     O     46
P     O     47
N     O     48
A     O     49
T     O     50
A     O     51
I     O     52
K     O     53
W     O     54
E     O     55
Q     O     56
K     O     57
```

| | | |
|---|---|---|
| G | O | 58 |
| V | O | 59 |
| Y | O | 60 |
| Y | O | 61 |
| V | O | 62 |
| A | O | 63 |
| D | O | 64 |
| C | O | 65 |
| Q | O | 66 |
| A | O | 67 |
| D | O | 68 |
| G | O | 69 |
| R | O | 70 |
| E | O | 71 |
| K | O | 72 |
| E | O | 73 |
| V | O | 74 |
| W | O | 75 |
| F | O | 76 |
| D | O | 77 |
| A | O | 78 |
| N | O | 79 |
| A | O | 80 |
| N | O | 81 |
| W | O | 82 |
| L | O | 83 |
| M | O | 84 |
| T | O | 85 |
| E | O | 86 |
| T | O | 87 |
| E | O | 88 |
| L | O | 89 |
| N | O | 90 |
| S | O | 91 |
| I | O | 92 |
| N | O | 93 |
| N | O | 94 |
| L | O | 95 |
| P | O | 96 |
| P | O | 97 |
| A | O | 98 |
| V | O | 99 |
| L | O | 100 |
| T | O | 101 |
| A | O | 102 |
| F | O | 103 |
| M | O | 104 |
| E | O | 105 |
| S | O | 106 |
| S | O | 107 |
| Y | O | 108 |
| N | O | 109 |
| N | O | 110 |
| W | O | 111 |
| V | O | 112 |
| V | O | 113 |
| D | O | 114 |
| D | O | 115 |
| V | O | 116 |

```
V      O      117
I      O      118
L      O      119
E      O      120
Y      O      121
P      O      122
N      O      123
E      O      124
P      O      125
S      O      126
T      O      127
E      O      128
F      O      129
V      O      130
V      O      131
T      O      132
V      O      133
E      O      134
Q      O      135
G      O      136
K      O      137
K      O      138
V      O      139
D      O      140
L      O      141
Y      O      142
F      O      143
S      O      144
E      O      145
G      O      146
G      O      147
G      O      148
L      O      149
L      O      150
H      O      151
E      O      152
K      O      153
D      O      154
V      O      155
T      O      156
N      O      157
G      O      158
D      O      159
D      O      160
T      O      161
H      O      162
W      O      163
P      O      164
R      O      165
V      O      166
END
```

**Web page prediction.** Given the above sequence (translated to FASTA format), Figure A.1 shows the prediction made by our web server, and Figure A.2 shows the prediction made on the PSIPRED server using the DISOPRED3 method (Jones and Cozzetto 2015).



**Protein Disorder Prediction - Individual Project**

This server uses deep learning methods to examine the intrinsic disorder in protein sequences.

Protein sequences can be found from the following websites: Protein Data Bank and UniProt.

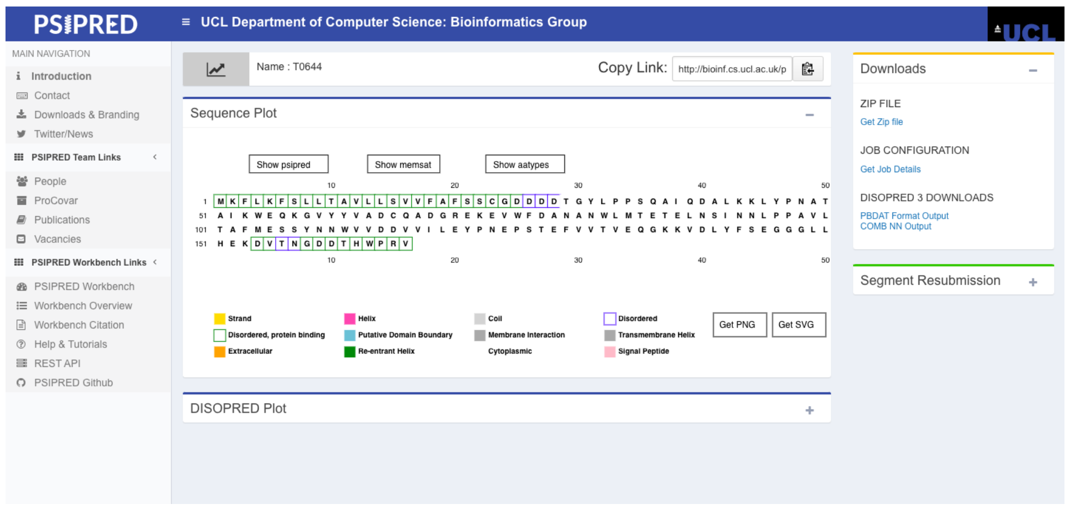
To enter another sequence: return to home page

**Name of entered sequence: >T0644**

Positions in the sequence that have been identified as being disordered are coloured red, ordered residues are blue.

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

*Figure A.1: Prediction made for CASP target T0644 using the LSTM model on our web server. This full web page shows users can also navigate back to the home page, and save an image of the sequence disorder prediction.*



Name : T0644

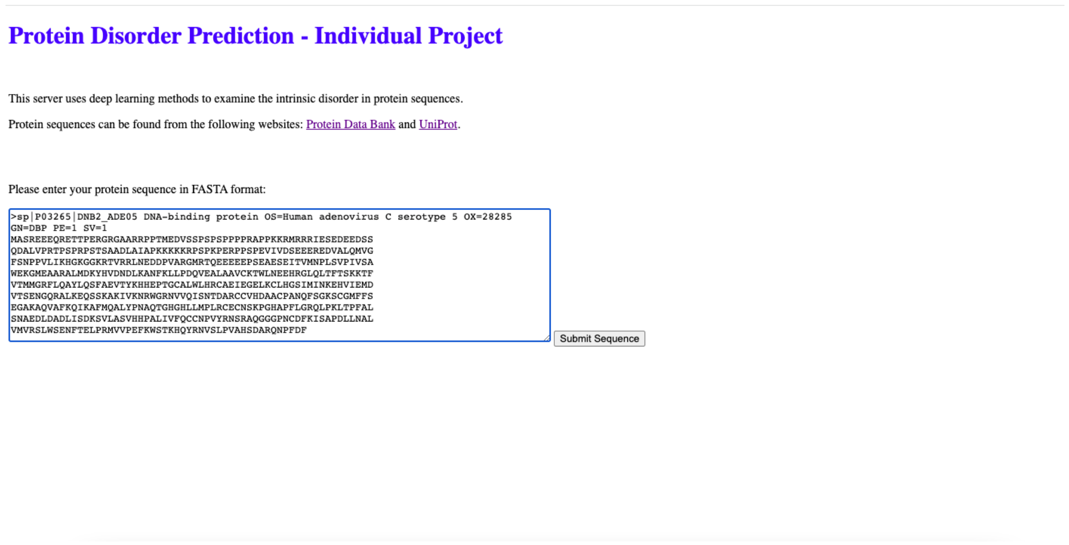
```
          10        20        30        40        50
  1  M K F L K F S L L T A V L L S V V F A F S S C G D D D D T G Y L P P S Q A I Q D A L K K L Y P N A T
 51  A I K W E Q K G V Y Y V A D C Q A D G R E K E V W F D A N A N W L M T E T E L N S I N N L P P A V L
101  T A F M E S S Y N N W V V D D V V I L E Y P N E P S T E F V V T V E Q G K K V D L Y F S E G G G L L
151  H E K D V T N G D D T H W P R V
          10        20        30        40        50
```

| | | | |
|---|---|---|---|
| 🟨 Strand | 🟪 Helix | ⬜ Coil | ☐ Disordered |
| ☐ Disordered, protein binding | 🟦 Putative Domain Boundary | ⬛ Membrane Interaction | ⬛ Transmembrane Helix |
| 🟧 Extracellular | 🟩 Re-entrant Helix | Cytoplasmic | 🟥 Signal Peptide |

*Figure A.2: Prediction made for CASP target T0644 using DISOPRED3. This full web page demonstrates the additional features the PSIPRED server has around the sequence disorder prediction.*

## A.3 Walkthrough of Web Server

We now present a brief walkthrough of the pages in our web server, demonstrating the form input (Figure A.3) to sequence-based visualisation (Figure A.4). We use the first protein in the DisProt dataset: Human adenovirus C serotype 5 (HAdV-5) (Human adenovirus 5), found on UniProt (The UniProt Consortium 2023).



## Protein Disorder Prediction - Individual Project

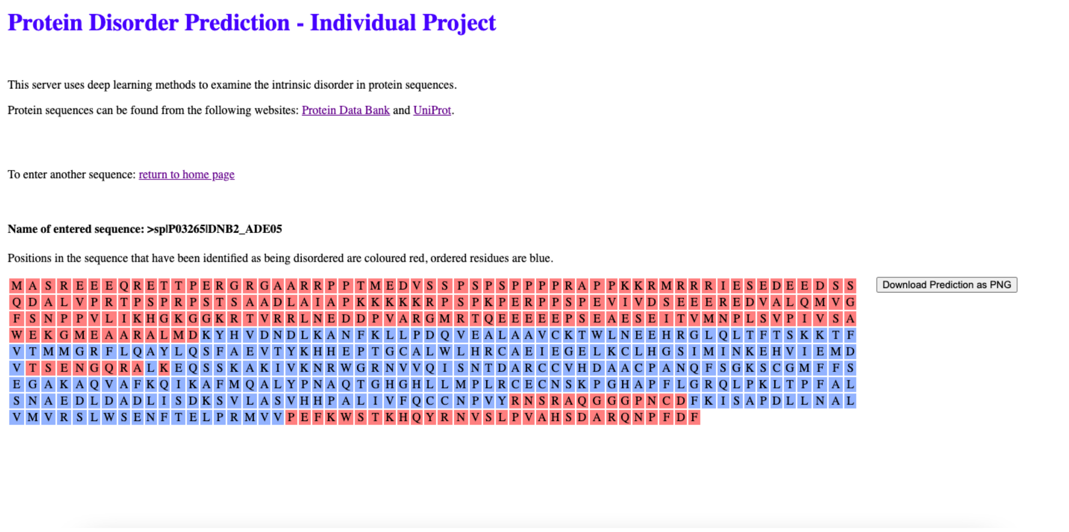
This server uses deep learning methods to examine the intrinsic disorder in protein sequences.

Protein sequences can be found from the following websites: Protein Data Bank and UniProt.

Please enter your protein sequence in FASTA format:

```
>sp|P03265|DNB2_ADE05 DNA-binding protein OS=Human adenovirus C serotype 5 OX=28285
GN=DBP PE=1 SV=1
MASREEEQRETTPERGRGAARRPPTMEDVSSPSPSPPPPRAPPKKRMRRRIESEDEEDSS
QDALVPRTPSPRPSTSAADLAIAPKKKKKRPSPKPERPPSPEVIVDSEEEREDVALQMVG
FSNPPVLIKHGKGGKRTVRRLNEDDPVARGMRTQEEEEEPSEAESEITVMNPLSVPIVSA
WEKGMEAARALMDKYHVDNDLKANFKLLPDQVEALAAVCKTWLNEEHRGLQLTFTSKKTF
VTMMGRFLQAYLQSFAEVTYKHHEPTGCALWLHRCAEIEGELKCLHGSIMINKEHVIEMD
VTSENGQRALKEQSSKAKIVKNRWGRNVVQISNTDARCCVHDAACPANQFSGKSCGMFFS
EGAKAQVAFKQIKAFMQALYPNAQTGHGHLLMPLRCECNSKPGHAPFLGRQLPKLTPFAL
SNAEDLDADLISDKSVLASVHHPALIVFQCCNPVYRNSRAQGGGPNCDFKISAPDLLNAL
VMVRSLWSENFTELPRMVVPEFKWSTKHQYRNVSLPVAHSDARQNPFDF
```

*Figure A.3:* *The home page of our web server. We have entered the protein sequence in FASTA format into our form, and we are about to click the submit sequence button.*



## Protein Disorder Prediction - Individual Project

This server uses deep learning methods to examine the intrinsic disorder in protein sequences.

Protein sequences can be found from the following websites: Protein Data Bank and UniProt.

To enter another sequence: return to home page

**Name of entered sequence: >sp|P03265|DNB2_ADE05**

Positions in the sequence that have been identified as being disordered are coloured red, ordered residues are blue.

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

*Figure A.4:* *The disorder prediction page. The prediction produced by our deep neural network is displayed using a sequence-based visualisation. Here red highlights disordered regions, and blue highlights ordered regions.*

# Bibliography

Agile Business Consortium (14), 'Moscow prioritisation', https://www.agilebusiness.org/page/ProjectFramework_10_MoSCoWPrioritisation. Last accessed: 2023-03-24.

Alshehri, M. A., Manee, M. M., Al-Fageeh, M. B. and Al-Shomrani, B. M. (2020), 'Genomic analysis of intrinsically disordered proteins in the genus camelus', *International Journal of Molecular Sciences* **21**(11), 4010.

Altschul, S. F., Madden, T. L., Schäffer, A. A., Zhang, J., Zhang, Z., Miller, W. and Lipman, D. J. (1997), 'Gapped blast and psi-blast: a new generation of protein database search programs', *Nucleic acids research* **25**(17), 3389–3402.

Arbel, N. (2018), 'How lstm networks solve the problem of vanishing gradients', https://medium.datadriveninvestor.com/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577. Last accessed: 2023-03-24.

Ardila, D., Kiraly, A. P., Bharadwaj, S., Choi, B., Reicher, J. J., Peng, L., Tse, D., Etemadi, M., Ye, W., Corrado, G. et al. (2019), 'End-to-end lung cancer screening with three-dimensional deep learning on low-dose chest computed tomography', *Nature medicine* **25**(6), 954–961.

BBC (2023), 'National 5 biology: Cell biology: Proteins', https://www.bbc.co.uk/bitesize/guides/zcr74qt/revision/1. Last accessed: 2023-03-24.

BerenLuthien, A. (2016), 'What are the advantages of relu over sigmoid function in deep neural networks?', https://stats.stackexchange.com/questions/126238/what-are-the-advantages-of-relu-over-sigmoid-function-in-deep-neural-networks. Last accessed: 2023-03-24.

Berman, H. M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T. N., Weissig, H., Shindyalov, I. N. and Bourne, P. E. (2000), 'The protein data bank', *Nucleic acids research* **28**(1), 235–242.

Bhattacharyya, M. (20), 'Gradient accumulation: Overcoming memory constraints in deep learning', https://towardsdatascience.com/gradient-accumulation-overcoming-memory-constraints-in-deep-learning-36d411252d01. Last accessed: 2023-03-24.

Bisong, E. (2019), 'Google colaboratory', *Building machine learning and deep learning models on google cloud platform: a comprehensive guide for beginners* pp. 59–64.

Brownlee, J. (2017), 'How to train a final machine learning model', https://machinelearningmastery.com/train-final-machine-learning-model/. Last accessed: 2023-03-24.

Brownlee, J. (2019), 'How to use learning curves to diagnose machine learning model performance', https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/. Last accessed: 2023-03-24.

Buchan, D. W. and Jones, D. T. (2019), 'The psipred protein analysis workbench: 20 years on', *Nucleic acids research* **47**(W1), W402–W407.

Cheriyedath, S. (2019), 'Protein folding', `https://www.news-medical.net/life-sciences/Protein-Folding.aspx`. Last accessed: 2023-03-24.

Chicco, D. and Jurman, G. (2020), 'The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation', *BMC genomics* **21**, 1–13.

Cook, A. (2022), 'Data leakage', `https://www.kaggle.com/code/alexisbcook/data-leakage`. Last accessed: 2023-03-24.

Devlin, J., Chang, M.-W., Lee, K. and Toutanova, K. (2018), 'Bert: Pre-training of deep bidirectional transformers for language understanding', *arXiv preprint arXiv:1810.04805* .

Django (2005), 'Django project', `https://www.djangoproject.com/`. Last accessed: 2023-03-24.

Dosztányi, Z. (2018), 'Prediction of protein disorder based on iupred', *Protein Science* **27**(1), 331–340.

Ema, R. R. and Adnan, N. (2022), 'Protein secondary structure prediction based on cnn and machine learning algorithms', *International Journal of Advanced Computer Science and Applications* **13**(11), 74–81.

Emenecker, R. J., Griffith, D. and Holehouse, A. S. (2021), 'Metapredict: a fast, accurate, and easy-to-use predictor of consensus disorder and structure', *Biophysical Journal* **120**(20), 4312–4319.

Fawcett, A. (2021), 'Data science in 5 minutes: What is one hot encoding?', `https://www.educative.io/blog/one-hot-encoding`. Last accessed: 2023-03-24.

Godoy, D. (2018), 'Understanding binary cross-entropy / log loss: a visual explanation', `https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a`. Last accessed: 2023-03-24.

Goodfellow, I., Bengio, Y. and Courville, A. (2016), *Deep Learning*, MIT Press. `http://www.deeplearningbook.org`.

Google Developers (22), 'Classification: True vs. false and positive vs. negative', `https://developers.google.com/machine-learning/crash-course/classification/true-false-positive-negative`. Last accessed: 2023-03-24.

Graves, A. and Schmidhuber, J. (2005), 'Framewise phoneme classification with bidirectional lstm and other neural network architectures', *Neural networks* **18**(5-6), 602–610.

Gundersen, G. (20), 'The log-sum-exp trick', `https://gregorygundersen.com/blog/2020/02/09/log-sum-exp/`. Last accessed: 2023-03-24.

Hanson, J., Paliwal, K. K., Litfin, T. and Zhou, Y. (2019), 'Spot-disorder2: improved protein intrinsic disorder prediction by ensembled deep learning', *Genomics, proteomics & bioinformatics* **17**(6), 645–656.

Hanson, J., Yang, Y., Paliwal, K. and Zhou, Y. (2017), 'Improving protein disorder prediction by deep bidirectional long short-term memory recurrent neural networks', *Bioinformatics* **33**(5), 685–692.

Hawkins, J. and Bodén, M. (2005), 'The applicability of recurrent neural networks for biological sequence analysis', *IEEE/ACM Transactions on Computational Biology and Bioinformatics* **2**(3), 243–253.

Hu, G., Katuwawala, A., Wang, K., Wu, Z., Ghadermarzi, S., Gao, J. and Kurgan, L. (2021), 'fldpnn: Accurate intrinsic disorder prediction with putative propensities of disorder functions', *Nature communications* **12**(1), 4438.

Jones, D. T. and Cozzetto, D. (2015), 'Disopred3: precise disordered region predictions with annotated protein-binding activity', *Bioinformatics* **31**(6), 857–863.

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A. et al. (2021), 'Highly accurate protein structure prediction with alphafold', *Nature* **596**(7873), 583–589.

Keerthi, A. (2022), 'Fantastic activation functions and when to use them', `https://towardsdatascience.com/fantastic-activation-functions-and-when-to-use-them-481fe2bb2bde`. Last accessed: 2023-03-24.

Klausen, M. S., Jespersen, M. C., Nielsen, H., Jensen, K. K., Jurtz, V. I., Soenderby, C. K., Sommer, M. O. A., Winther, O., Nielsen, M., Petersen, B. et al. (2019), 'Netsurfp-2.0: Improved prediction of protein structural features by integrated deep learning', *Proteins: Structure, Function, and Bioinformatics* **87**(6), 520–527.

Kumar, S. (2020), 'Feature engineering — deep dive into encoding and binning techniques', `https://towardsdatascience.com/feature-engineering-deep-dive-into-encoding-and-binning-techniques-5618d55a6b38`. Last accessed: 2023-03-24.

Liaw, R., Liang, E., Nishihara, R., Moritz, P., Gonzalez, J. E. and Stoica, I. (2018), 'Tune: A research platform for distributed model selection and training', *arXiv preprint arXiv:1807.05118* .

Long, J., Shelhamer, E. and Darrell, T. (2015), Fully convolutional networks for semantic segmentation, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 3431–3440.

Martinelli, A. H., Lopes, F. C., John, E. B., Carlini, C. R. and Ligabue-Braun, R. (2019), 'Modulation of disordered proteins with a focus on neurodegenerative diseases and other pathologies', *International journal of molecular sciences* **20**(6), 1322.

McGuffin, L. J., Bryson, K. and Jones, D. T. (2000), 'The psipred protein structure prediction server', *Bioinformatics* **16**(4), 404–405.

McKinney, W. (2010), Data structures for statistical computing in python, *in* 'Proceedings of the 9th Python in Science Conference', Austin, TX, pp. 56–61.

Merkel, D. (2014), 'Docker: lightweight linux containers for consistent development and deployment', *Linux journal* **2014**(239), 2.

Mizianty, M. J., Peng, Z. and Kurgan, L. (2013), 'Mfdp2: Accurate predictor of disorder in proteins by fusion of disorder probabilities, content and profiles', *Intrinsically disordered proteins* **1**(1), e24428.

Mohammadi, A., Zahiri, J., Mohammadi, S., Khodarahmi, M. and Arab, S. S. (2022), 'Pssmcool: a comprehensive r package for generating evolutionary-based descriptors of protein sequences from pssm profiles', *Biology Methods and Protocols* **7**(1), bpac008.

Monastyrskyy, B., Kryshtafovych, A., Moult, J., Tramontano, A. and Fidelis, K. (2014), 'Assessment of protein disorder region predictions in casp10', *Proteins: Structure, Function, and Bioinformatics* **82**, 127–137.

Morgan, I. and Saleh, O. (2017), Entropic elasticity in the giant muscle protein titin, *in* 'APS March Meeting Abstracts', Vol. 2017, pp. A4–003.

Moult, J., Fidelis, K., Kryshtafovych, A., Schwede, T. and Tramontano, A. (2014), 'Critical assessment of methods of protein structure prediction (casp)—round x', *Proteins: Structure, Function, and Bioinformatics* **82**(S2), 1–6.

Moult, J. et al. (1994), 'Protein structure prediction center', `https://predictioncenter.org/`. Last accessed: 2023-03-24.

National Center for Biotechnology Information (1988), 'Fasta format for nucleotide sequences', `https://www.ncbi.nlm.nih.gov/genbank/fastaformat/`. Last accessed: 2023-03-24.

Necci, M., Piovesan, D. and Tosatto, S. C. (2021), 'Critical assessment of protein intrinsic disorder prediction', *Nature methods* **18**(5), 472–481.

Nickolls, J., Buck, I., Garland, M. and Skadron, K. (2008), 'Scalable parallel programming with cuda: Is cuda the parallel programming model that application developers have been waiting for?', *Queue* **6**(2), 40–53.

O'Connor, C. M., Adams, J. U. and Fairman, J. (2010), 'Essentials of cell biology', *Cambridge, MA: NPG Education* **1**, 54.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L. et al. (2019), 'Pytorch: An imperative style, high-performance deep learning library', *Advances in neural information processing systems* **32**.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V. et al. (2011), 'Scikit-learn: Machine learning in python', *the Journal of machine Learning research* **12**, 2825–2830.

Prilusky, J., Felder, C. E., Zeev-Ben-Mordehai, T., Rydberg, E. H., Man, O., Beckmann, J. S., Silman, I. and Sussman, J. L. (2005), 'Foldindex©: a simple tool to predict whether a given protein sequence is intrinsically unfolded', *Bioinformatics* **21**(16), 3435–3438.

Quaglia, F., Hatos, A., Salladini, E., Piovesan, D. and Tosatto, S. C. (2022), 'Exploring manually curated annotations of intrinsically disordered proteins with disprot', *Current Protocols* **2**(7), e484.

Reese, W. (2008), 'Nginx: the high-performance web server and reverse proxy', *Linux Journal* **2008**(173), 2.

Remmert, M., Biegert, A., Hauser, A. and Söding, J. (2012), 'Hhblits: lightning-fast iterative protein sequence searching by hmm-hmm alignment', *Nature methods* **9**(2), 173–175.

Romero, P., Obradovic, Z., Li, X., Garner, E. C., Brown, C. J. and Dunker, A. K. (2001), 'Sequence complexity of disordered protein', *Proteins: Structure, Function, and Bioinformatics* **42**(1), 38–48.

Steinegger, M. and Söding, J. (2017), 'Mmseqs2 enables sensitive protein sequence searching for the analysis of massive data sets', *Nature biotechnology* **35**(11), 1026–1028.

Tenchov, R. (2022), 'Are intrinsically disordered proteins the key to treating covid-19?', `https://www.cas.org/resources/cas-insights/drug-discovery/intrinsically-disordered-proteins-covid-19`. Last accessed: 2023-03-24.

The UniProt Consortium (2023), 'Uniprot: the universal protein knowledgebase in 2023', *Nucleic Acids Research* **51**(D1), D523–D531.

Walsh, I., Martin, A. J., Di Domenico, T. and Tosatto, S. C. (2012), 'Espritz: accurate and fast prediction of protein disorder', *Bioinformatics* **28**(4), 503–509.

Wang, S., Sun, S., Li, Z., Zhang, R. and Xu, J. (2017), 'Accurate de novo prediction of protein contact map by ultra-deep learning model', *PLoS computational biology* **13**(1), e1005324.

Wang, S., Weng, S., Ma, J. and Tang, Q. (2015), 'Deepcnf-d: predicting protein order/disorder regions by weighted deep convolutional neural fields', *International journal of molecular sciences* **16**(8), 17315–17330.

Ward, J. J., McGuffin, L. J., Bryson, K., Buxton, B. F. and Jones, D. T. (2004), 'The disopred server for the prediction of protein disorder', *Bioinformatics* **20**(13), 2138–2139.

Wikipedia contributors (2022), 'Position weight matrix — Wikipedia, the free encyclopedia', `https://en.wikipedia.org/w/index.php?title=Position_weight_matrix&oldid=1120683380`. Last accessed: 2023-03-24.

Wikipedia contributors (2023*a*), 'Amino acid — Wikipedia, the free encyclopedia', `https://en.wikipedia.org/w/index.php?title=Amino_acid&oldid=1146210544`. Last accessed: 2023-03-24.

Wikipedia contributors (2023*b*), 'Evaluation of binary classifiers — Wikipedia, the free encyclopedia', `https://en.wikipedia.org/w/index.php?title=Evaluation_of_binary_classifiers&oldid=1146372896`. Last accessed: 2023-03-24.

Wikipedia contributors (2023*c*), 'Gunicorn — Wikipedia, the free encyclopedia', `https://en.wikipedia.org/w/index.php?title=Gunicorn&oldid=1132680247`. Last accessed: 2023-03-24.

Wikipedia contributors (2023*d*), 'Intrinsically disordered proteins — Wikipedia, the free encyclopedia', `https://en.wikipedia.org/w/index.php?title=Intrinsically_disordered_proteins&oldid=1137357647`. Last accessed: 2023-03-24.

Wikipedia contributors (2023*e*), 'Protein engineering — Wikipedia, the free encyclopedia', `https://en.wikipedia.org/w/index.php?title=Protein_engineering&oldid=1139746054`. Last accessed: 2023-03-24.

Wikipedia contributors (2023*f*), 'Shapiro–wilk test — Wikipedia, the free encyclopedia', `https://en.wikipedia.org/w/index.php?title=Shapiro%E2%80%93Wilk_test&oldid=1143103925`. Last accessed: 2023-03-24.

Wikipedia contributors (2023*g*), 'Wilcoxon signed-rank test — Wikipedia, the free encyclopedia', `https://en.wikipedia.org/w/index.php?title=Wilcoxon_signed-rank_test&oldid=1136398620`. Last accessed: 2023-03-24.

Xie, S., Girshick, R., Dollár, P., Tu, Z. and He, K. (2017), Aggregated residual transformations for deep neural networks, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 1492–1500.

Xue, B., Dunbrack, R. L., Williams, R. W., Dunker, A. K. and Uversky, V. N. (2010), 'Pondr-fit: a meta-predictor of intrinsically disordered amino acids', *Biochimica et Biophysica Acta (BBA)-Proteins and Proteomics* **1804**(4), 996–1010.

Zeng, H., Edwards, M. D., Liu, G. and Gifford, D. K. (2016), 'Convolutional neural network architectures for predicting dna–protein binding', *Bioinformatics* **32**(12), i121–i127.

Zhang, A., Lipton, Z. C., Li, M. and Smola, A. J. (2021), 'Dive into deep learning', *arXiv preprint arXiv:2106.11342* .
**URL:** *https://github.com/d2l-ai/d2l-en*

Zhao, B. and Kurgan, L. (2022), 'Deep learning in prediction of intrinsic disorder in proteins', *Computational and Structural Biotechnology Journal* **20**, 1286–1294.

Zhou, P., Feng, J., Ma, C., Xiong, C., Hoi, S. C. H. et al. (2020), 'Towards theoretically understanding why sgd generalizes better than adam in deep learning', *Advances in Neural Information Processing Systems* **33**, 21285–21296.