3-10-2021

DeePSLiM: A Deep Learning Approach to Identify Predictive Short-linear Motifs for Protein Sequence Classification

Alexandru Filip
University of Windsor

DeePSLiM: A Deep Learning Approach to Identify Predictive Short-linear Motifs for Protein Sequence Classification

By

Alexandru Filip

A Thesis
Submitted to the Faculty of Graduate Studies
through the School of Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Master of Science
at the University of Windsor

Windsor, Ontario, Canada

2020

DeePSLiM: A Deep Learning Approach to Identify Predictive Short-linear Motifs for Protein Sequence Classification

by

Alexandru Filip

APPROVED BY:

M. Belalia
Department of Mathematics and Statistics

A. Mukhopadhyay
School of Computer Science

L. Rueda, Co-Advisor
School of Computer Science

A. Ngom, Co-Advisor
School of Computer Science

December 14, 2020

DECLARATION OF CO-AUTHORSHIP / PREVIOUS PUBLICATION

I. Co-Authorship

I hereby declare that this thesis incorporates material that is result of joint research, as follows:

Chapter 2 of the thesis was co-authored with Luis Rueda and Alioune Ngom. L. Rueda and A. Ngom contributed with initial thoughts about this research area and the main ideas, and assisted in elaborating on the new novel approach implemented in this work. Both authors contributed in finalizing the idea and follow-up discussions. A. Filip implemented the method, data pre-processing, experimental design, and the data analysis. A. Filip wrote the contents of the chapter. L. Rueda and A. Ngom assisted in writing and reviewing the content of the chapter.

I am aware of the University of Windsor Senate Policy on Authorship and I certify that I have properly acknowledged the contribution of other researchers to my thesis, and have obtained written permission from each of the co-author(s) to include the above material(s) in my thesis. I certify that, with the above qualification, this thesis, and the research to which it refers, is the product of my own work.

II. Previous Publication

This thesis includes an original paper that has been previously submitted for publication in the following IEEE journal:

Thesis chapter	**Publication title**	**Publication Status**
Chapter 2	Alexandru Filip, Alioune Ngom and Luis Rueda, DeePSLiM: A Deep Learning Approach to Identify Predictive Short-linear Motifs for Protein Sequence Classification, IEEE/ACM Transactions on Computational Biology and Bioinformatics, 2020	Submitted

I certify that I have obtained a written permission from the copyright owner(s) to include the above published material(s) in my thesis. I certify that the above material describes work completed during my registration as a graduate student at the University of Windsor.

III. General

I declare that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution. I understand that my thesis may be made electronically available to the public.

ABSTRACT

With the increasing quantity of biological data, it is important to develop algorithms that can quickly find patterns in large databases of DNA, RNA and protein sequences. Previous research has been very successful at applying deep learning methods to the problems of motif detection as well as classification of biological sequences. There are, however, limitations to these approaches. Most are limited to finding motifs of a single length. In addition, most research has focused on DNA and RNA, both of which use a four letter alphabet. A few of these have attempted to apply deep learning methods on the larger, twenty letter, alphabet of proteins.

We present an enhanced deep learning model, called DeePSLiM, capable of detecting predictive, short linear motifs (SLiM) in protein sequences. The model is a shallow network that can be trained quickly on large amounts of data. The SLiMs are predictive because they can be used to classify the sequences into their respective families. The model was able to reach scores of 94.5% on accuracy, precision, recall, F1-Score and Matthews-correlation coefficient, as well as 99.9% area under the receiver operator characteristic curve (AUROC).

DEDICATION

I dedicate this thesis to my mother. Your love, patience and kindness are the reason I was able to complete this work. Thank you for teaching me never to give up.

I also dedicate this thesis to the memory of my father. Without the lessons you have taught me and the example you have set, I would not be where I am today. Thank you.

AKNOWLEDGEMENTS

I would like to thank my supervisor, Dr. Luis Rueda, and co-supervisor, Dr. Alioune Ngom. Their collective experience and guidance have helped make this process a gratifying one. It was a pleasure being one of their students.

I would also like to thank my external reader, Dr. Mohamed Belalia, and internal reader, Dr. Asish Mukhopadhyay, for taking time out of their schedules to help in the final phase of this work.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

DNA	Deoxyribonucleic acid
RNA	Ribonucleic acid
AA	Amino acid
SLiM	Short linear motif
PFM	Position frequency matrix
PPM	Position probability matrix
PWM	Position weight matrix
MSA	Multiple sequence alignment
ANN	Artificial neural network
ReLU	Rectified linear unit
CNN	Convolutional neural network
EM	Expectation Maximization

CHAPTER 1

Introduction

1.1 Introduction to Molecular Biology

1.1.1 Proteins

Inside of every living cell, basic functions, such as metabolism, cell structure, monitoring of intra and extracellular conditions, movement and defence, are carried out in large part by proteins [12, 17]. Proteins are the most abundant macromolecules in any living organism due to the large number of functions they perform. The basic building blocks of proteins are amino acids (AA). AAs are synthesized into long chains to form the primary structure of a protein. There are 20 different amino acids that can be used to form the primary structure and they are shown in Fig. 1.1.1. Simply put, proteins are sequences with a 20 letter alphabet.

Each protein is encoded in the DNA of the cell. DNA, like proteins, is a long chain formed of more basic subunits called nucleic acids. There are 5 nucleic acids, 4 of which are used in DNA (Adenine, Thymine, Guanine and Cytosine) and 4 of which are used in another macromolecule, called RNA (Adenine, Uracil, Guanine and Cytosine). The nucleic acids are shown in Fig. 1.1.2. To create a single protein, a cell uses two processes, called transcription and translation, collectively known as the central dogma of molecular biology. The central dogma is shown in Fig. 1.1.3. First, the desired protein encoding is found in a gene, located on the DNA. The gene is then written to a short strand of RNA through transcription, which is then translated from the 4 letter alphabet of nucleic acids into the 20 letter alphabet of

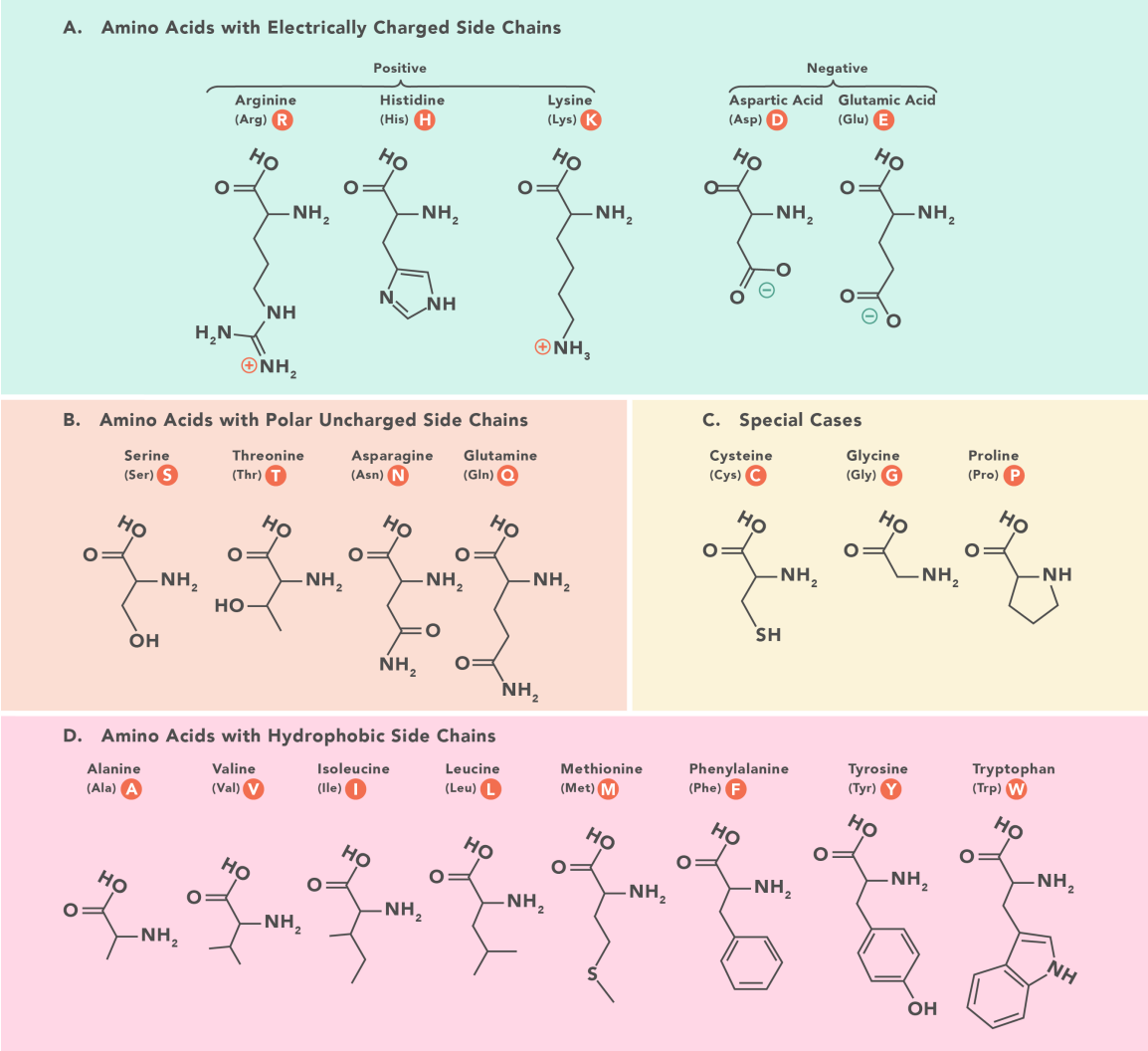

Fig. 1.1.1: The molecular structures of the 20 amino acids.

proteins. Following translation the protein will fold into its most stable configuration, forming the secondary and tertiary structures. The shape as well as the AAs present in a protein will determine its function. Further post-translational modifications may occur as well as the formation of larger complexes with other proteins to form the quaternary structure.

1.1.2 Motifs

A motif is a common pattern observed among a group of macromolecules such as DNA, RNA or proteins. When applied to proteins, the term motif may refer to one

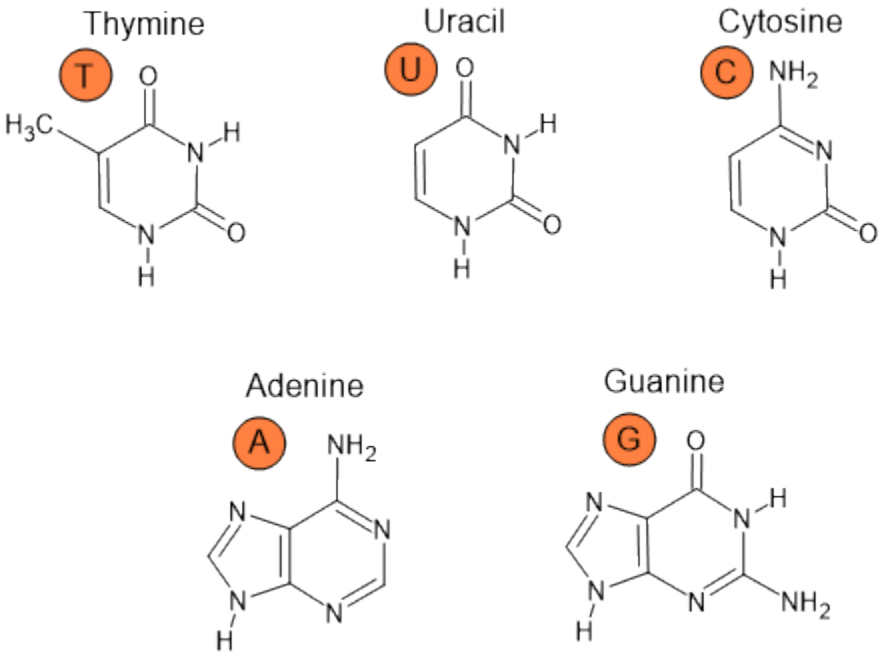

Fig. 1.1.2: The molecular structures of the 5 nucleic acids.

of three different types: sequence motif, structural motif or functional motif [14]. We will primarily focus on sequence motifs in this work. These are subsequences in the primary structure of a protein ranging from 3-20 AAs in length. A *subsequence* of a string S is defined as a contiguous subset of characters in S. More formally, a subsequence of a string S of length n is specified by two indices i_1 and i_2 defining the beginning and end of the subsequence in S, where $1 \leq i_1 < i_2 \leq n$. The subsequence specified by the range $[i_1, i_2]$ is the string $S(i_1)S(i_1+1)S(i_1+2)\ldots S(i_2)$.

The three types of motifs are related to one another. A sequence motif gives rise to structural or functional motifs, however, this is not a simple relationship. The same sequence motif present in many proteins may be involved different structures or functions depending on other sequences present on the protein. Conversely, many proteins that perform the same function may make use of different sequence motifs. Motifs are conserved during the evolution of a species and are encoded in the genes of every organism in the species, that is to say in the parts of their DNA that encode proteins. For this reason, this work focuses on the detection of sequence motifs within protein families.

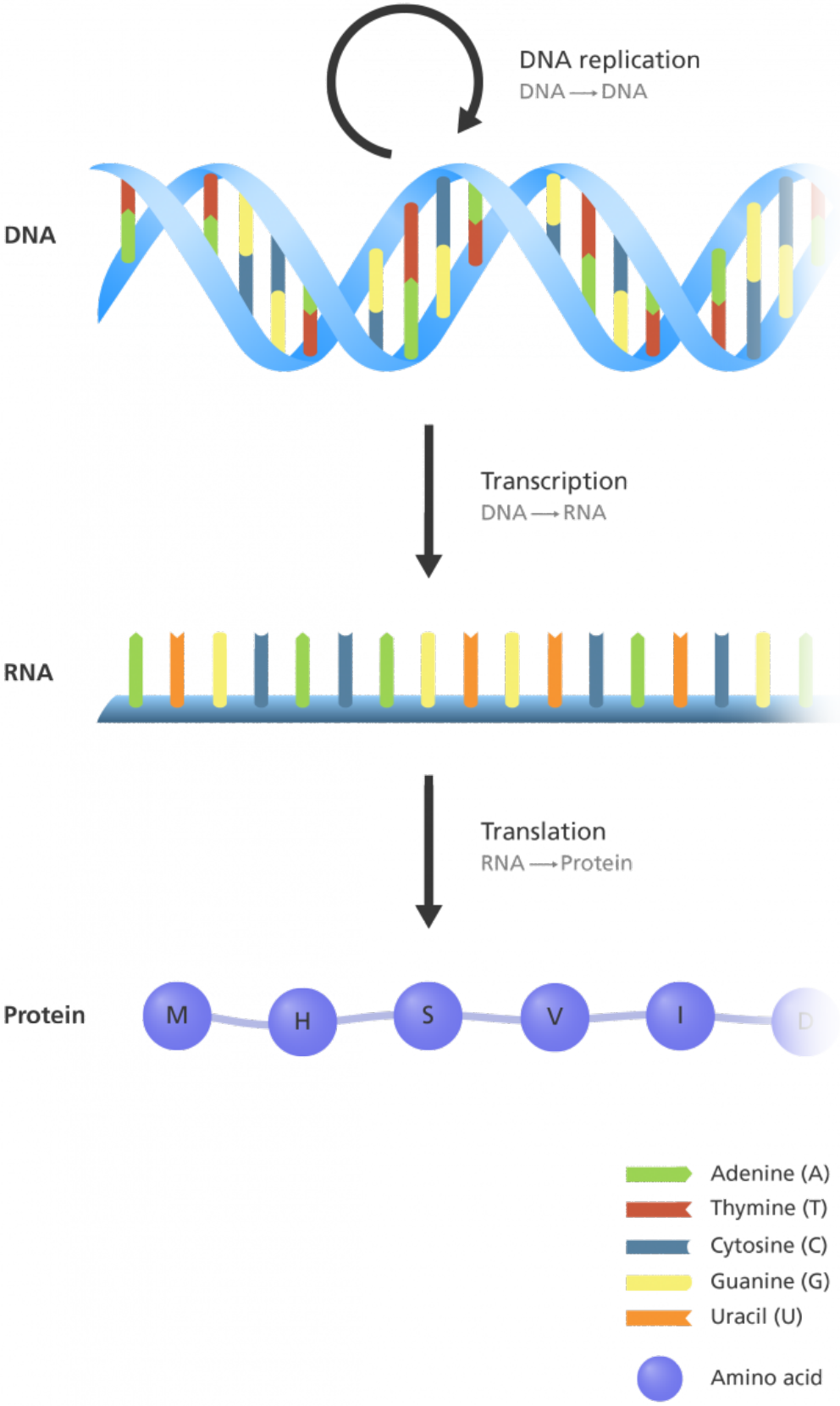

Fig. 1.1.3: The central dogma of molecular biology.

1.1.3 Short Linear Motifs

A short linear motif (SLiM) is a sequence motif consisting of 3-10 contiguous amino acids [6]. SLiMs are conserved in cell lines for their effects on structure and function. These may include signaling, degradation and cleavage. A SLiM can be represented in many different ways including a logo, regular expression, position frequency matrix (PFM), position probability matrix (PPM) and position weight matrix (PWM). Representing a SLiM as a logo or regular expression, as shown in Fig. 1.1.4, make it easily readable by beginners and professionals alike. A matrix representation, as shown in Fig. 1.1.5 and 1.1.6, is less readable but provides a method of evaluating a sequence of the appropriate length for the probability that the sequence belongs to the given motif. Evaluation of a sequence with a PPM is performed by multiplying the probability of each amino acid in the sequence at its corresponding location in the PPM. An example is shown in Fig. 1.1.7. Evaluating a PWM is done by adding the values instead of multiplying and produces a log-probability. A PWM can be calculated from a given PFM using the following equation [8]

$$w_{i,j} = \ln \frac{n_{i,j} + p_i}{(N+1)p_i}\,, \tag{1}$$

where $w_{i,j}$ is the value at index i, j in the PWM, $n_{i,j}$ is the corresponding value in the PFM, p_i is the probability of AA i in the dataset, also called the background probability, and N is the number of sequences in the database.

1.2 Problem Statement

1.2.1 Planted (l, d)-Motif Problem

Given a set of t sequences, each of length n, find a common subsequence M of length l, called a motif, or a variant of M that is present in most or all of the sequences. Each planted variant of M contains exactly d point substitutions [9]. We also refer to this problem as the motif finding problem and any algorithm that is meant to solve

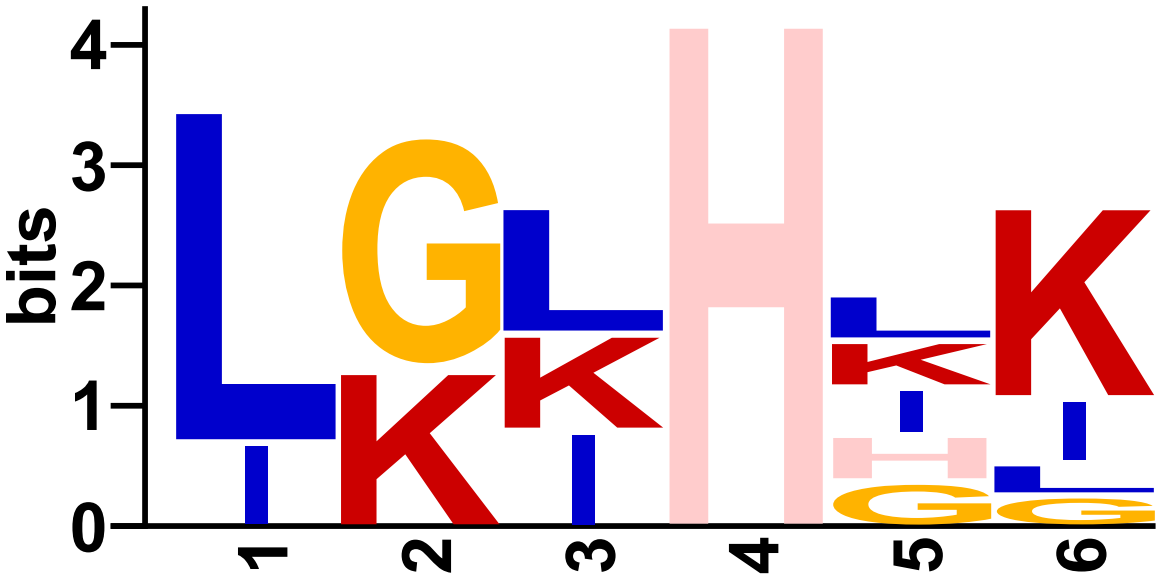

Fig. 1.1.4: Example of a motif logo. The regular expression of this motif is [LI][GK][LKI]H[LKIHG][KILG].

$$\begin{array}{c} \\ L \\ H \\ I \\ G \\ K \end{array} \begin{bmatrix} & & & & & \\ 0.8 & 0.0 & 0.4 & 0.0 & 0.2 & 0.1 \\ 0.0 & 0.0 & 0.0 & 1.0 & 0.2 & 0.0 \\ 0.2 & 0.0 & 0.3 & 0.0 & 0.2 & 0.2 \\ 0.0 & 0.6 & 0.0 & 0.0 & 0.2 & 0.1 \\ 0.0 & 0.4 & 0.3 & 0.0 & 0.2 & 0.6 \end{bmatrix}$$

Fig. 1.1.5: PPM of the logo shown in Fig. 1.1.4. Note that in this example, the probabilities of some amino acids are shown for brevity. When representing a PPM in code, values are stored for all amino acids.

$$\begin{array}{c} \\ L \\ H \\ I \\ G \\ K \end{array} \begin{bmatrix} 1.399 & -2.996 & 0.718 & -2.996 & 0.049 & -0.598 \\ -2.996 & -2.996 & -2.996 & 1.619 & 0.049 & -2.996 \\ 0.049 & -2.996 & 0.438 & -2.996 & 0.049 & 0.049 \\ -2.996 & 1.115 & -2.996 & -2.996 & 0.049 & -0.598 \\ -2.996 & 0.718 & 0.438 & -2.996 & 0.049 & 1.115 \end{bmatrix}$$

Fig. 1.1.6: PWM of the logo shown in Fig. 1.1.4. The matrix is calculated from the PPM in Fig. 1.1.5 and Eq. 1 after multiplying the values in the PPM by 10 to create a PFM. We assumed a background probability of 0.1 and 20 sequences in the dataset.

$$0.8 \times 0.4 \times 0.3 \times 1.0 \times 0.2 \times 0.6 = 0.01152$$

Fig. 1.1.7: Calculation of the probability that the sequence LKIHGK belongs to the motif shown in Fig. 1.1.4 using the PPM in Fig. 1.1.5.

this problem as a motif finding algorithm.

A motif finding algorithm may not be applicable to results obtained in a wet lab. Protein sequences are rarely of the same length and the parameters that describe motifs are unknown upon obtaining a set of protein sequences. To increase the applicability to wet lab results, algorithm designers relax the parameters n, l and d as follows: sequences are allowed to vary in length, a range of lengths may be specified for l and d is treated as a parameter to be minimized by the algorithm. Experimental results may also contain multiple motifs, therefore motif finding algorithms are designed to find multiple motifs in a set of sequences, not one as is stated in the problem.

One may notice that the planted (l, d)-motif problem is similar to the multiple sequence alignment (MSA) problem [2]. The difference between the two is that the MSA problem asks for a global comparison of the input sequences whereas motif finding asks for a local comparison. The difference becomes obvious when considering how one motif may be present at multiple locations throughout a sequence. An algorithm that performs MSA may find one occurrence of a motif in each input

sequence whereas an algorithm that performs motif finding may find all occurrences. Additionally, if many motifs are shared between two sequences but do not occur in the same order, a MSA algorithm will be able to align the occurrence of some motifs but a motif finding algorithm is expected to find all occurrences.

1.3 Motivation

As stated previously, finding a SLiM does not necessarily imply the presence of a structure or function. However, it can provide likely candidates without the need to compare a protein with every other protein in a database [16]. This would allow researchers to more quickly determine properties of novel proteins.

SLiMs have also been found to have a role in disease. For example, mutations in proteins that are thought to be responsible for cancer cell lines have been found in functionally important SLiMs. Some believe that the study of these SLiMs may lead to the development of targeted treatments [15]. Moreover, viruses are believed to infect cells and hijack their internal mechanisms through SLiM mimicry [6]. Some of the cell mechanisms that are hijacked by viruses are cellular transport, signal transduction, control of protein levels and transcriptional regulation. Each is a potential site that can used as a target for antiviral drugs.

1.4 Existing Tools

Previous motif finding algorithms can generally be placed into two categories; supervised and unsupervised. Supervised algorithms consider a set of input sequences and a set of labels. The algorithm must find motifs in the sequences that help when predicting the labels. Unsupervised algorithms generally take only the sequences as input. Their goal is to find motifs that optimize a chosen metric such as information content or p-value. The supervised motif finding algorithms that we we discuss here are DREME, DeepBind, DanQ and DeepSea. The unsupervised motif finding algorithms that we will discuss are MEME and Consensus.

MEME applies the expectation maximization (EM) algorithm to a set of DNA or protein sequences to find SLiMs [4]. The tool can be installed locally or used online in the MEME Suite [5]. The MEME algorithm works in three phases. First, it generates initial guesses for the number of motifs requested. Next, it will use the EM algorithm to maximize the likelihood of the motifs in the input sequences. Finally, the algorithm will evaluate the relative entropy of the discovered motifs and output those with the highest scores. MEME represents motifs as PWMs, meaning that motifs are of constant length and do not contain gaps. If a motif containing a variable length gap exists, MEME will discover the two ends and treat them as separate motifs.

The CONSENSUS algorithm is a greedy method for finding motifs in a dataset [8]. Internally, it represents a motif as a PFM and tries to maximize each PFMs information content. The algorithm takes a set of sequences, a motif width and the maximum number of motifs as input. First, the algorithm will iterate through the first sequence in the dataset in a sliding window manner. Each window is of the specified motif width. A PFM is generated for each window and is placed into a set of the most significant motifs found so far. The algorithm will then consider the remaining sequences one at a time. As before, a sliding window is used on each sequence. A PFM is generated for each of the windows and is added to the most significant PFMs, at first, those generated in the first step. The information content of each PFM is calculated and those with the highest information content are chosen as the next most significant motifs for future iterations. This continues until each sequence has contributed once to each PFM. Finally, identical motifs are removed from the result, meaning that the algorithm may return less than the desired number of motifs. Variations on the CONSENSUS algorithm exist, for example, one that will estimate the motif width, where the user has to first specify a bias that is subtracted from the information content so that the average score is negative.

DREME is an algorithm designed to find discriminative motifs in two sets of sequences [3]. The authors define discriminative motifs as motifs present in one of the datasets but not in the other. The dataset containing the motifs is called the positive set and the other is called the negative set. DREME is also available online

through the MEME suite [5]. DREME can be used to find motifs that are likely to be relevant in a set of DNA sequences that share a common property. The negative set is used to eliminate motifs that may be present in the positive set but are not relevant to the property of interest. The negative set will usually contain sequences that are related to those in the positive set (perhaps ones that belong to the same species) but that are known not to share the property of interest. DREME internally represents motifs as regular expressions (RE). DREME takes two DNA sequence datasets and a significance threshold as input. The first step is to generate seed REs by finding short (3-8 letter) sequences containing no wildcards that occur in many sequences of the positive set and few sequences of the negative set. Next, the algorithm goes on to select the RE with the highest statistical significance using Fisher's Exact Test. Wildcards are added to the seed REs one at a time until no new wildcard can be added. Next, the most significant RE in the dataset is chosen, converted to a PPM and its E-value is calculated [8]. Every occurrence of the motif is then erased from the input. The algorithm will continue this process until no new motif with an E-value higher than the significance threshold is found. The result is the set of PPMs discovered by the search.

DeepBind is an approach that uses deep learning to predict the binding specificity of proteins to DNA or RNA [1]. In their experiments, the researchers used a ChIP-seq dataset to predict the binding specificity of DNA to proteins, taking a step toward finding the regulatory mechanisms of the cell. They were able to extract known motifs from the convolutional neural network (CNN) filters in the first layer of the network. The model that we propose expands upon DeepBind by applying the concepts to protein sequences.

DeepSEA [18] is another deep learning model that looks for the effects of mutations in noncoding regions of DNA. The model finds regulatory regions in DNA by predicting 919 chromatin features, including transcription factor binding, DNase activity and histone binding. A model that improves on the results of DeepSEA is DanQ [13]. By adding a bidirectional recurrent neural network between the CNN and the fully connected layers the researchers were able to further improve recognition and

motif discovery.

The architectures presented above use a single filter length in the CNN layer, limiting the network to detect SLiMs of a single length. The classification tasks that many of these models are trained to perform provide limited information from binary classification data. DanQ demonstrated that the use of multiple labels can provide more information to the network, allowing for better classification and discovery of higher-quality motifs. Finally, researchers were able to successfully discover SLiMs in DNA sequences, which raises the question of the applicability of their methods to the twenty-letter alphabet of protein sequences. The effectiveness of neural networks on protein sequence classification has been demonstrated by other architectures, such as DeepSol [11]. Their aim, however, was not to discover motifs with their model and thus did not make an attempt to do so.

1.5 Artificial Neural Networks

Artificial neural networks (ANN) are a popular machine learning model inspired by the cellular structure of the human brain. The brain is made up of cells, called neurons, that are connected in a complex, yet organized, web as shown in Fig. 1.5.1. Each neuron sends electrical signals down a long tail, called an axon, to nearby neurons each of which sends the signal further when the total sum of electrical signals received passes a threshold. The complexity of this network leads to every one of the brain's functions from breathing to speech and cognition.

The earliest attempt at mimicking neurons in software was the perceptron, shown in Fig. 1.5.2. Perceptrons take a weighted sum of their inputs and emit a 1 when their threshold is surpassed and 0 otherwise. Eventually, the threshold function was replaced with any function that produced a real number to allow for more flexibility in the output. Any function that modifies the output of a perceptron is called an activation function.

Real brains are extremely complex and a simplified structure was needed to perform machine learning tasks. In an ANN, neurons are grouped together into layers.

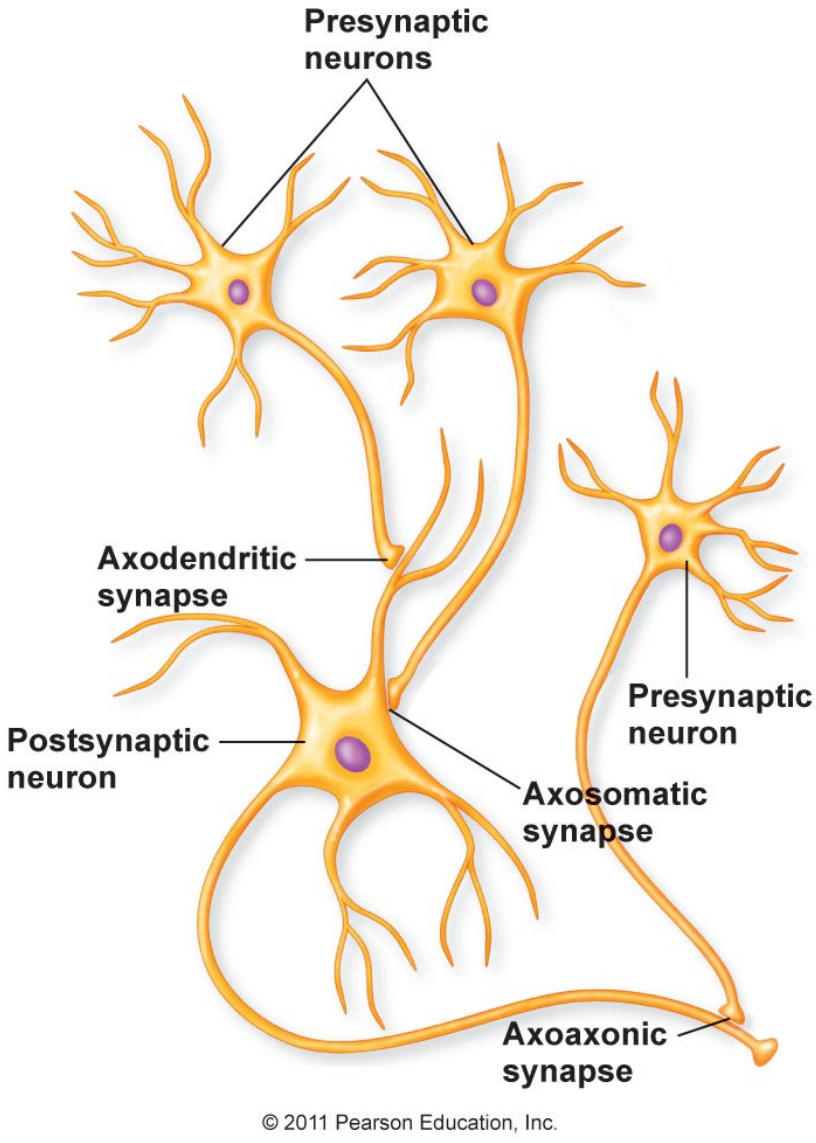

Fig. 1.5.1: The structure of a neuron and how it connects to neighboring neurons.

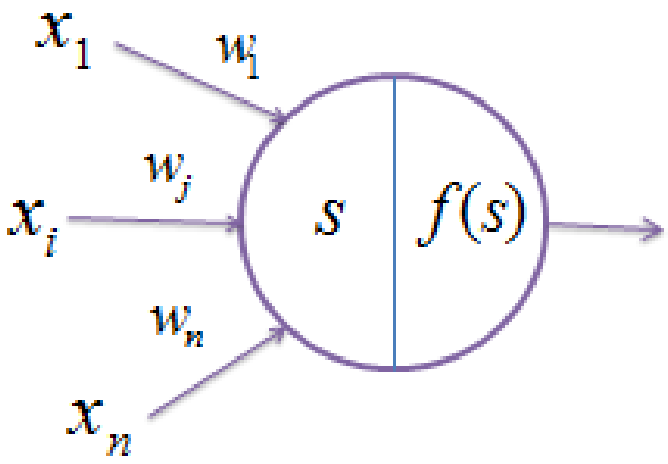

Fig. 1.5.2: The basic structure of a perceptron.

Each layer processes the output of the previous layer and produces a new output that can be further processed or act as the result of the network. Each layer contains a set of weights that are adjusted in order to train the network to perform its desired task well. This task may be either classification (labelling or categorizing points) or regression (approximating a function from a set of points).

Neurons in an ANN are grouped together into a multi-dimensional array called a tensor. Briefly, the simplest tensor is a list of numbers called a vector. Placing the numbers in a 2-dimensional grid is called a matrix. The tensor can be further extended into any number of dimensions. To avoid confusion, the number of dimensions that a tensor contains is called its rank. Thus, a vector is a rank-1 tensor, a matrix has rank-2 and any higher rank is simply referred to as a rank-n tensor, where n is a non-negative integer.

The layers of an ANN are functions that act on tensors. Layers can be arranged in any way but are typically stacked on top of one another, each processing the output of the one before. More complex architectures can have a branching structure where multiple layers process the same input. The outputs of these layers can be merged together using one of many operations. Some examples are concatenation, element-wise addition and element-wise multiplication. Finally, the output of the network is evaluated to by measuring how far it diverges from the expected output. The

function used to measure the difference from the expected output is call the error or loss function.

In regression, the loss function measures the distance between the predicted and expected point. An example of this may be the L^2 norm, which is defined as

$$\sqrt{\sum_{i=1}^{n} x_i^2}\,, \tag{2}$$

where $\mathbf{x}$ is the vector output by the network and n is the length of $\mathbf{x}$.

In classification, the output is a set of probabilities, one for each class. The loss function must measure the difference between two probability distributions. A common loss function used for classification is the cross-entropy, which is defined as

$$\sum_{i=1}^{n} y_i \log(\hat{y}_i)\,, \tag{3}$$

where $\mathbf{x}$ is a list of probabilities corresponding to each class, n is the length of $\mathbf{x}$ or the number of classes, y_i is the true label of the sample (1 for the correct class and 0 for all other classes) and $\hat{y}_i$ is the prediction that the network made.

An ANN can generalize well to a given dataset and learn complex relationships between the input and output. Part of the strength of an ANN comes from the use of activation functions, which allow the network to learn non-linear relationships. Some commonly used activation functions are sigmoid, rectified linear unit (ReLU), leaky ReLU and softmax, which are defined as

$$\text{sigmoid}(x) = \frac{1}{1+e^{-x}}\,, \tag{4}$$

$$\text{ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}\,, \tag{5}$$

$$\text{Leaky-ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases}\,, \tag{6}$$

$$\text{softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}\,, \tag{7}$$

equations 4, 5 and 6 are defined for real numbers, not vectors. Thus, they are applied to each component of a vector. In equation 6 the term α is a parameter that can be set by the user, typically the number used is 0.01. Softmax, equation 7, is the only activation function listed here that is defined on a vector. When creating a classifier on a neural network the softmax function is commonly used as the final activation function and ensures that the output can be treated as a list of probabilities.

1.5.1 One-Hot Encoding

One-hot encoding is a common scheme used in neural networks to represent categorical data in the input and output. If a value can be in one of n classes, a vector of length n is used to represent the value. This vector has zeros at all positions except the one that represents the desired class. At this position, a one is placed.

1.5.2 Training

Training of the neural network is performed using the backpropagation algorithm. Backpropagation updates the individual weights by first calculating the derivative of each weight with respect to the network's error. The derivative gives the direction of steepest ascent in the error function. Subtracting the derivative moves the weights in the direction of steepest descent, i.e. the direction in which the error is reduced. Before subtracting, the derivative is multiplied by a small number, called the learning rate, to avoid overshooting the target. This is the update step of backpropagation and is shown in Eq. 8.

The "back" in backpropagation comes from the fact that the algorithm calculates the derivative of each layer using the derivative of the layer that follows, thus propagating the error backward through the network. To perform this calculation, the chain rule is applied as shown in Eq. 9. For this reason, all functions in a neural network must be differentiable, including activation functions. One may notice that

ReLU and leaky ReLU are not differentiable at $x = 0$. Here, the derivative is defined to be 0.

$$\Delta w_i^{(l)} = -\eta \frac{\partial E}{\partial w_i^{(l)}}\,, \tag{8}$$

where $w_i^{(l)}$ is weight i of layer l, E is the error of the network for the training step and η is a small number called the learning rate.

$$\frac{\partial E}{\partial w_i^{(l)}} = \frac{\partial E}{\partial y_i^{(l)}} \cdot \frac{\partial y_i^{(l)}}{\partial w_i^{(l)}}\,, \tag{9}$$

where, as before, $w_i^{(l)}$ is weight i of layer l and E is the error of the network. Additionally, $y_i^{(l)}$ is the ith element of the output of layer l.

1.5.3 Fully Connected Layer

A fully connected layer is the simplest operation with trainable weights that is performed in a neural network. The layer takes a vector as input and provides a vector as output. The size of input and output vectors do not have to be the same. A linear transformation is performed on the input vector x using a weight matrix W and bias vector b as follows

$$y = Wx + b\,, \tag{10}$$

to produce the output vector y. The parameters W and b are trainable parameters specific to the layer and are adjusted using backpropagation (Eq. 8) to fit the input to the desired output. Initially, the values of W are randomly sampled from a normal distribution and b is initialized to the zero vector.

A fully connected layer is also called a dense layer or linear layer. The latter term is used because, on its own, it can only learn linear relationships between the input and output. To overcome this limitation, one of the activation functions mentioned previously can be applied. The most common functions used after a linear layer are sigmoid (Eq. 4) and softmax (Eq. 7), if the it is the final layer in the network, or ReLU (Eq. 5) and leaky-ReLU (Eq. 6) if it is not the last layer.

1.5.4 Convolutional Layer

Convolutional layers were created to analyze images with a neural network, commonly referred to as a convolutional neural network (CNN). They were later generalized to handle any data with samples that have a spatial structure, meaning that information within a sample should be considered within its local context. This includes pixels in an image, pieces of a sound waveform and letters in a word. Some common tasks that CNNs are used to perform are object detection within images and speech recognition.

Consider, for example, the CIFAR-10 dataset, which contains images of size $32 \times 32 \times 3$ (width, height and channels). A fully connected layer that takes one of these images as input must first flatten it into a vector of size $32 \times 32 \times 3 = 3072$. If the layer produces a vector of size n it would require $(3072+1) \times n$ weights. Furthermore, when performing a task like object detection the layer must learn how to detect objects at any location in an image. This results in redundancy in the learned weights and calls for a new method to reduce a layer's size while also considering a sample's spatial structure.

A convolutional layer contains a small set of weights, called a kernel or filter, that is evaluated at every location in a sample. The rank of a filter is the same as that of a sample in the dataset with an additional bias term. Filters always contain at least one channel dimension. Two-dimensional data, such as images, have rank 3 (width, height and channel), whereas one-dimensional data, such as sound or natural language, has rank 2 (length and channel). The size of a filter along the variable dimensions (width, height, length, etc.) can be any number that the user desires and is typically chosen to be small. The number of channels in the filter must match the number of channels in the input. For example, colour images have three channels for red, green and blue, thus filters used on such images must also have three channels but can have any size along the width and height. For the remainder of this work we will consider one-dimensional convolutions due to the fact that we are working with protein sequences. The theory, however, can be extended to any number of dimensions.

When applying a filter to a sample, the filter is placed at the beginning of the sequence. A weighted sum is performed by first multiplying each weight in the filter with the corresponding value in the sample, then adding the results to the bias. The filter is then shifted over to the next location in the sequence and the operation is performed again. This repeats until the entire sequence has been evaluated. An activation function is applied on the output of each convolutional layer to allow it to perform non-linear operations. A single filter can detect one feature in the input, for this reason convolution layers apply many filters to the same sequence. Each filter considers every channels of the input and produces one output channel. The convolution operation for a single layer p is defined as follows

$$y_{h,i}^{(p+1)} = b_i + \sum_{j=1}^{W_p} \sum_{k=1}^{C_p} w_{j,k}^{(p)} x_{j+h,k}^{(p)} \,, \tag{11}$$

where W_p is the width of the kernel along the variable dimension, C_p is the number of input channels, h is the location in the output and i is the index of the filter being evaluated as well as the index of the corresponding channel in the output. The operation is performed from $h = 1$ up to $L - W_p + 1$ where L is the length of the input sequence. An example of a convolutional operation is shown in Fig. 1.5.3.

Each convolutional layer can also have a stride parameter to indicate how far to move the filters between evaluations. The length of the output given input of length L can be calculated as follows

$$\left\lfloor \frac{L - W_p}{s} + 1 \right\rfloor \,, \tag{12}$$

where s is the stride.

1.5.5 Pooling

Pooling is similar to a convolution in that an operation is performed on small windows of the input data. Unlike convolutional layers, pooling layers do not have trainable weights. Instead, they perform simple operations on each window that only rely on the

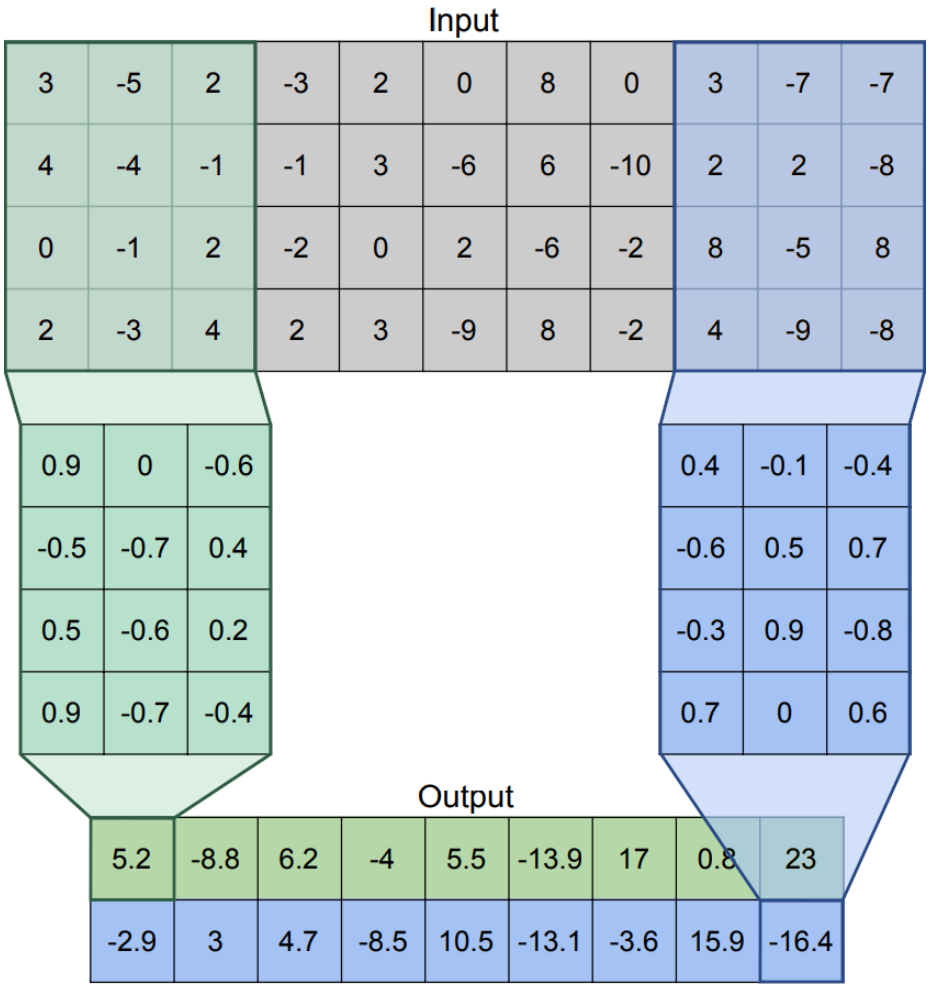

Fig. 1.5.3: An example of a convolution performed along a one-dimensional sequence.

input data. The most common kinds of pooling are max-pooling and average-pooling. As the names suggest, max-pooling takes the maximum value for every channel in the window and average pooling takes their average. In their most common form, pooling operations are performed using strides that are equal to the window's width. This means that overlapping windows are not considered.

Typically, pooling operations are performed after the activation function of each convolutional layer to reduce spatial size. This is meant to reduce redundancy in the features detected by the network and decrease computational load. An example of max and average pooling is shown in Fig. 1.5.4

Each pooling strategy has advantages and disadvantages. Max-pooling considers the strongest feature detected in each window resulting in more certainty in the network's decision. The disadvantage of this strategy is that during training, the network learns from one detection of a feature in each window, meaning that other detections are ignored. Average pooling considers every occurrence of a feature within a window, allowing the layer to learn from many more instances. The disadvantage occurs when the data in a window is sparse, i.e. it contains many zeros, which may occur after applying ReLU. Taking an average would result in a weaker signal and more uncertainty in the output.

Global pooling is another operation that is performed following the final convolutional layer. Global pooling is performed along all variable dimensions, reducing their lengths to 1 and creating a vector with length equal to the number of channels. The resulting vector can then be passed through a stack of fully connected layers to perform classification. It is worth noting here that another common operation used in-between convolutional and fully connected layers is the flatten operation. This is where the input data is treated as if it were sequential. In this work, however, we focus on global pooling. The advantages and disadvantages of max and average pooling, discussed previously, are exaggerated when performing the operation globally. In Section 1.6.4, we will discuss a trade-off strategy that attempts to cover the weaknesses of each strategy with the strengths of the other.

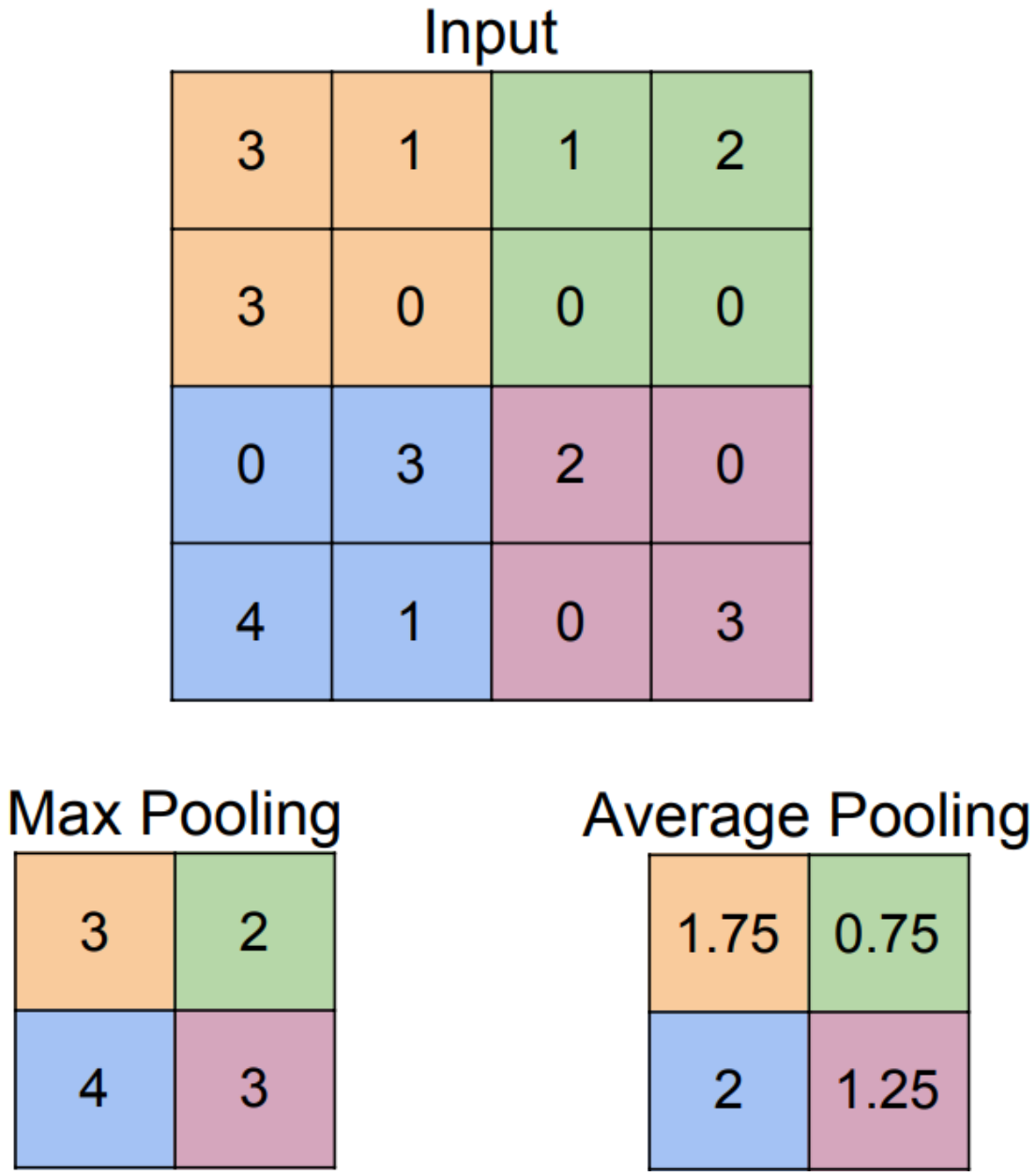

Fig. 1.5.4: An example of Max and Average Pooling on a two-dimensional grid.

1.5.6 Batch Normalization

Each layer of an ANN is dependent on all of the previous layers. This means that small changes to the parameters of one layer can result in large changes to the inputs of deeper layers. Each layer must continually learn a new distribution of its input, a problem called internal covariate shift. This results in long training times as the learning rate needs to be made very small. Batch normalization (BatchNorm) is a technique used to overcome internal covariate shift [10]. To achieve this, a BatchNorm layer learns a new distribution and scales each minibatch accordingly.

In short, a minibatch is a set of samples randomly selected from the dataset, which is used as the input to the network rather than using individual samples. Minibatches are helpful because the average gradient over a minibatch is a good estimate of the gradient of the full dataset. Modern hardware allows us to perform many computations in parallel making algorithms designed around using minibatches more efficient. A BatchNorm layer will find the mean and variance of a minibatch and use these values to normalize the input. The layer then re-adjusts the input using learned parameters for mean and standard deviation. The result is shorter training times and more stable gradients. The equations to perform batch normalization are

$$\begin{aligned} \mu_B &= \tfrac{1}{m}\textstyle\sum_{i=1}^{m} x_i \\ \sigma_B^2 &= \tfrac{1}{m}\textstyle\sum_{i=1}^{m} (x_i - \mu_B)^2 \\ \hat{x_i} &= \tfrac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \\ y_i &= \gamma \hat{x_i} + \beta \end{aligned} \quad , \tag{13}$$

where m is the size of a minibatch, x_i is the ith sample in the minibatch, μ_B is the average of the samples, σ_B^2 is the variance of samples, $\hat{x_i}$ is the normalized input, ϵ is a small number used to prevent division by zero, y_i is the output of the layer and β and γ are the learned mean and standard deviation respectively.

1.6 Proposed Method

1.6.1 Full Network Architecture

Here, we will describe the architecture of the neural network in detail. The full network is shown in Fig. 2.2.1.

1.6.2 One-hot Encoding of Input Sequence

We use one-hot encoding as the representation for both the input and output of our network. The simplest case is the output. Each position of the output vector represents a protein family and the network produces a number between 0 and 1 at every position, representing the probability that the sequence is of the desired class. The families are one-hot encoded to represent a 100% chance of the true family and a 0% chance of every other family. The input is a sequence of one-hot encoded vectors, representing the amino acids present at each position. The length of each of these vectors is 20, each position representing one amino acid.

Representing the input as a sequence of one-hot encoded vectors and considering a convolutional filter to be a PWM makes the convolution operation similar to finding the log-probability of a motif at each position in the sequence, similar to the method described in Section 1.1.3. This is the case because each position in the PWM is multiplied by 1 for the corresponding AA and 0 otherwise. When the values are added, the result is the same as adding only the values in the PWM of each amino acid.

1.6.3 Multi-Length Convolution

To allow the network to learn motifs of multiple lengths, multiple convolutional layers are applied to the same input sequence. Each layer contains many filters of the same length, representing motifs of that length. However, the lengths of filters that belong to different layers are not the same. In our experiments, the first layer contained filters of length 3, the next of length 4 and so on until the final layer of length 24.

This step results in multiple tensors, each with a different length given by Eq. 12. It is for this reason that the convolution operations must be performed separately.

1.6.4 Trade-Off Pooling Strategy

Following the convolution operation, a global pooling is applied to take all occurrences of a motif into consideration. As mentioned in Section 1.5.5, applying average pooling, especially global average pooling, to sparse data would result in a weak signal that makes it difficult for the network to perform classification. In this section, we will introduce our novel trade-off pooling strategy to take advantage of the strengths of both average and max-pooling.

Our strategy first finds the locations where a motif was detected by looking for all values greater than zero, creating a rank-2 tensor with ones where the values are greater than zero and zeros everywhere else. The sum of these values along the variable dimension is the number of occurrences of each motif. By taking the sum of the original input along the variable dimension we obtain another vector where each element is simply the total of the scores for the corresponding motif along the length of the sequence. Finally, the values in the total score vector are divided element-wise by the number of occurrences of the motif plus one, resulting in an average score along the length of the sequence, which considers only detected motifs. This operation is shown in Eq. 14.

$$\mathbf{e_{w,i}} = \frac{\sum_{j=1}^{l-w+1} M_{i,j}}{\sum_{j=1}^{l-w+1} c(M_{i,j}) + 1}\,, \tag{14}$$

where

$$c(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}\,, \tag{15}$$

1.6.5 Sequence Classification

After pooling, the vectors are concatenated depth-wise. The result is a vector with its length equal to the total number of motifs recognized by the network. This vector

is first passed through a batch normalization layer then classified using a shallow neural network consisting of two fully connected layers separated by a leaky ReLU with $\alpha = 0.01$. The number of neurons in the final layer is equal to the number of protein families. A softmax activation function is used to turn the vector into a list of probabilities and cross-entropy is used as the loss function.

1.6.6 Contributions

In this work we propose a deep learning model, DeePSLiM, for quickly finding predictive SLiMs in a set of proteins. The SLiMs are predictive because they can be used to determine the family that a given protein belongs to. The network has few weights making it fast to train on large quantities of data. Unlike other neural network models, DeePSLiM can learn SLiMs of multiple width due to the use of filters being evaluated in parallel. In addition, we have also proposed a novel pooling method that aims to be a tradeoff between max-pooling and average-pooling. DeePSLiM is available as a Python package on github [7].

The contributions of this thesis can be summarized as follows:

- Proposed a neural network model, called DeePSLiM, that is able to differentiate proteins into their respective families based on their amino acid sequences.
- Proposed a method to detect SLiMs of different lengths by use of multiple convolutional neural network layers evaluating the input sequence in parallel.
- Proposed a novel pooling method that serves as a tradeoff between average-pooling and max-pooling to capture the strengths of each method and decrease training time.
- Implemented DeePSLiM as a Python package, which can be found at [7].

References

[1] Babak Alipanahi et al. "Predicting the sequence specificities of DNA-and RNA-binding proteins by deep learning". In: *Nature Biotechnology* 33.8 (2015), pp. 831–838.

[2] Thiru S Arthanari and Hoai An Le Thi. "New formulations of the multiple sequence alignment problem". In: *Optimization Letters* 5.1 (2011), pp. 27–40.

[3] Timothy L Bailey. "DREME: motif discovery in transcription factor ChIP-seq data". In: *Bioinformatics* 27.12 (2011), pp. 1653–1659.

[4] Timothy L Bailey, Charles Elkan, et al. "Fitting a mixture model by expectation maximization to discover motifs in bipolymers". In: (1994).

[5] Timothy L Bailey et al. "MEME SUITE: tools for motif discovery and searching". In: *Nucleic Acids Research* 37.suppl_2 (2009), W202–W208.

[6] Norman E Davey, Gilles Travé, and Toby J Gibson. "How viruses hijack cell regulation". In: *Trends in biochemical sciences* 36.3 (2011), pp. 159–169.

[7] Alexandru Filip. *Github Repo: DeePSLiM*. `https://github.com/AlexFilip/DeePSLiM`. 2020.

[8] Gerald Z Hertz and Gary D. Stormo. "Identifying DNA and protein patterns with statistically significant alignments of multiple sequences." In: *Bioinformatics (Oxford, England)* 15.7 (1999), pp. 563–577.

[9] Daniel Huson. "Algorithms in Bioinformatics II". In: *PNAS* (2002).

[10] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *arXiv preprint arXiv:1502.03167* (2015).

[11] Sameer Khurana et al. "DeepSol: a deep learning framework for sequence-based protein solubility prediction". In: *Bioinformatics* 34.15 (2018), pp. 2605–2613.

[12] David L Nelson, Albert L Lehninger, and Michael M Cox. *Lehninger principles of biochemistry*. Macmillan, 2008.

[13] Daniel Quang and Xiaohui Xie. "DanQ: a hybrid convolutional and recurrent deep neural network for quantifying the function of DNA sequences". In: *Nucleic Acids Research* 44.11 (2016), e107–e107.

[14] Luis Rueda and Manish Pandit. "A model based on minimotifs for classification of stable protein-protein complexes". In: *2014 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology*. IEEE. 2014, pp. 1–6.

[15] Bora Uyar et al. "Proteome-wide analysis of human disease mutations in short linear motifs: neglected players in cancer?" In: *Molecular BioSystems* 10.10 (2014), pp. 2626–2642.

[16] Kim Van Roey et al. "Short linear motifs: ubiquitous and functionally diverse protein interaction modules directing cell regulation". In: *Chemical reviews* 114.13 (2014), pp. 6733–6778.

[17] Donald Voet, Judith G Voet, and Charlotte W Pratt. *Fundamentals of biochemistry: life at the molecular level*. 577.1 VOE. 2013.

[18] Jian Zhou and Olga G Troyanskaya. "Predicting effects of noncoding variants with deep learning–based sequence model". In: *Nature Methods* 12.10 (2015), pp. 931–934.

CHAPTER 2

DeePSLiM: A Deep Learning Approach to Identify Predictive Short-linear Motifs for Protein Sequence Classification

2.1 Introduction

Motifs are patterns present in macromolecules, these being DNA, RNA and proteins. They can be found in the sequences of these molecules or they can be higher level patterns of structures and functions. SLiMs are subsequences present in larger biological sequences. A *subsequence* of a string S is defined as a contiguous subset of characters in S. More formally, a subsequence of a string S of length n is specified by two indices i_1 and i_2 defining the beginning and end of the subsequence in S, where $1 \leq i_1 < i_2 \leq n$. The subsequence specified by the range $[i_1, i_2]$ is the string $S(i_1)S(i_1+1)S(i_1+2)\ldots S(i_2)$.

SLiMs can be used to represent exact sets of symbols (nucleic acids or amino acids) but can also allow for the presence of different symbols with common properties at the same site, for example, amino acids with the same charge. A site is defined as the location or locations in a sequence where a SLiM is present. This means that interactions between distant parts of a single sequence are not taken into consideration. In a biological system, SLiMs act as cleavage sites for enzymes, sites of protein-protein

interaction, and serve a role in structural stability or enzymatic activity, among others. Many of these sequences are highly conserved in the evolution of an organism, leading to the presence of protein families carrying similar functions. This makes SLiMs good candidates for knowledge discovery, making machine learning algorithms to identify SLiMs highly desirable.

SLiMs can be represented in many ways, including regular expressions and matrix forms, each having it's own set of limitations. For instance, regular expressions do not show the probability of each symbol that can be present at a particular site, though they can be used to represent SLiMs of varying length. The opposite is true for the matrix representation, which can be used to show the probability of every symbol at each site, however, the matrix represents a fixed length SLiM. In this work, we represent a SLiM as a position weight matrix (PWM) as it is the simplest to emulate a motif in a neural network [10].

Many algorithms have been developed to discover new motifs. These include the use of expectation maximization [3, 2], hidden Markov models [21, 18], and deep learning approaches [1, 27, 14, 22]. SLiMs can be used in classifying protein, RNA, or DNA sequences and in retrieving information associated with structural or functional features. Conversely, finding protein, RNA, or DNA sequences that do not belong to the same class may indicate a fundamental difference in these features. For example, proteins found in the cytoplasm of a cell will likely contain similar patterns of amino acids to make them water-soluble, while their shapes are typically globular. They differ from transmembrane proteins, which are typically rod-shaped and contain long sections of non-polar amino acids, followed by short sections of polar amino acids to allow for the embedding of the protein in the cell membrane.

SLiMs are used in sequence analysis (i.e., classification, clustering, etc.) as an effective way of representing or translating biological sequences into vectors (i.e., points) in some space, and hence, allowing for the use of efficient classification algorithms such as support vector machines (SVMs) [16] and neural networks [13, 1]. However, sequence motif discovery and sequence analysis are usually performed separately. Recent studies, however, have shown that shallow neural networks can be

used to perform both steps simultaneously [1, 27, 22, 14]; i.e., discovering the SLiMs in a dataset of sequences while also finding a classifier model for the sequences.

DeepBind is an approach that uses deep learning to predict the binding specificity of proteins to DNA or RNA [1]. In their experiments, the researchers used a ChIP-seq dataset to predict the binding specificity of DNA to proteins, taking a step toward finding the regulatory mechanisms of the cell. They were able to extract known motifs from the convolutional neural network (CNN) filters in the first layer of the network. The model that we propose expands upon DeepBind by applying the concepts to protein sequences.

DeepSEA [27] is another deep learning model that looks for the effects of mutations in noncoding regions of DNA. The model finds regulatory regions in DNA by predicting 919 chromatin features, including transcription factor binding, DNase activity and histone binding. A model that improves on the results of DeepSEA is DanQ [22]. By adding a bidirectional recurrent neural network between the CNN and the fully connected layers the researchers were able to further improve recognition and motif discovery.

The architectures presented above use a single filter length in the CNN layer, limiting the network to detect SLiMs of a single length. The classification tasks that many of these models are trained to perform provide limited information from binary classification data. DanQ demonstrated that the use of multiple labels can provide more information to the network, allowing for better classification and discovery of higher-quality motifs. Finally, researchers were able to successfully discover SLiMs in DNA sequences, which raises the question of the applicability of their methods to the twenty-letter alphabet of protein sequences. The effectiveness of neural networks on protein sequence classification has been demonstrated by other architectures, such as DeepSol [14]. Their aim, however, was not to discover motifs with their model and thus did not make an attempt to do so.

In this thesis, we propose a novel deep learning model that allows neural networks to (1) discover multiple variable-length SLiMs in a large dataset of protein sequences, and (2) use the discovered SLiMs to classify the protein sequences according to their

families. This model is designed to be independent of sequence length and makes use of the information found in multi-class labels. Our work builds on top of existing models in literature such as DeepBind [1] and DeepSEA [27], to demonstrate the applicability to sequences with larger alphabets, such as proteins. We also demonstrate that it is possible to perform multi-class classification on these sequences and achieve meaningful results. Finally, we explore the use of a tradeoff strategy between average-pooling and max-pooling to eliminate the problems introduced by a sparse response from the motif scan, while also allowing the network to learn valuable information from multiple sites in each sequence.

2.2 Materials and Methods

In our experiments, we used the Uniprot database [6], a large dataset of manually curated protein sequences, along with labels provided by the Pfam database [8] to perform classification. Our model is a neural network consisting of multiple CNNs that are evaluated in parallel, pooling layers, vector concatenation and two fully-connected layers, which act as a classifier of the sequences. The first layer of the network is meant to be used to detect the sites where SLiMs are present in the sequences. For this reason, filter length and motif length are synonymous in this thesis and will be used interchangeably based on the context. The Pfam database places proteins into one of many families, making the problem one of multi-label classification. This means that our model will detect SLiMs that may be present in multiple families. The association between families and the SLiMs they contain can be determined once the neural network is trained by examining SLiMs detected when the network is presented with sequences from each family. For the purposes of our experiments we have applied filters of each length between 5 and 24. However, it is possible to make a model which uses any combination of filter lengths. Some of the SLiMs detected by the network will also be similar as there is no way to guarantee that the filters of a CNN will be unique.

2.2.1 Data Pre-processing

We first downloaded the protein sequences available from the Uniprot database [6]. Each sequence was presented in FASTA format and contained a collection of labels from many other databases such as Ensembl [7], PROSITE [23], PDB [4] and Pfam [8]. The Pfam labels were used as class labels for classification in this work. After selecting only those proteins with a Pfam label, we obtained 560,118 proteins, each belonging to one of 11,450 families. Some of the sequences contained letters that are not part of the standard 20-letter amino acid alphabet. These letters are B, O, U, X and Z. Each of these letters, except U, represents the possibility of multiple amino acids, for example, B represents Aspartic Acid (D) or Asparagine (N). All proteins sequences containing any of the non-standard letters were removed from the study. In addition, proteins that were present in multiple families and were identical in sequence were removed. The edit distances between the remaining sequences were measured. The largest distance found between any two sequences that did not belong to the same class was 21%. We examined the number of proteins present in the remaining families. These contained between 1 and 2,551 proteins. To prevent class imbalance, and therefore bias in the dataset, only those families with more than 500 sequences were used. The presence of many classes with a low number of examples would prevent the model from generalizing well. As a result of the pre-processing step, the final dataset contained 106,680 protein sequences in 142 families.

2.2.2 Model

Our proposed model is shown in Fig. 2.2.1. The input is provided as a one-hot encoded sequence of amino acids in an $l \times 20$ matrix, where l is length of the protein sequence. This step allows the filter in the convolution to be used as a PWM.

The first layer of the network is composed of a multi-length motif scan performed by running multiple convolutional layers on the same input sequence, similar to the one used by DeepSol [14]. The output of each layer is a matrix of size $f_m \times (l - m + 1)$ where m is the length of the filter, f_m is the number of filters of length m, and l is the

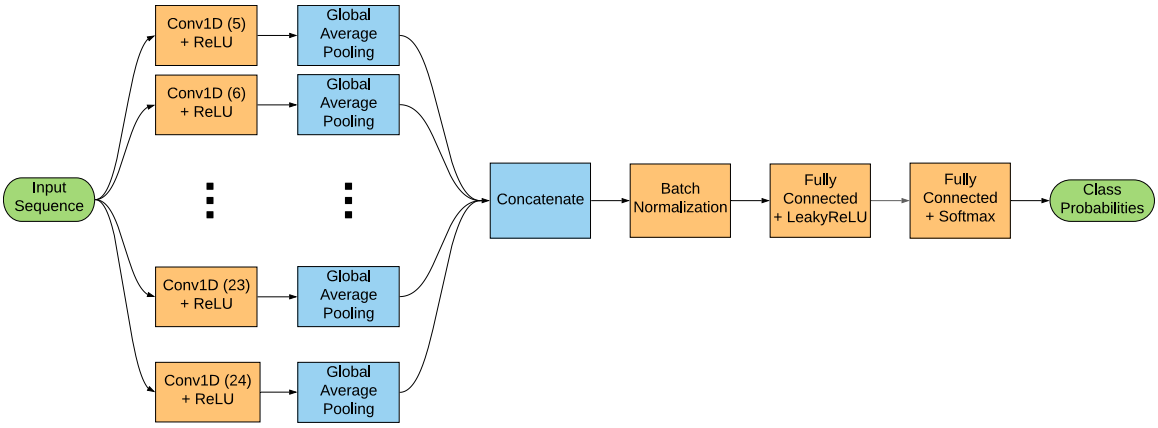

Fig. 2.2.1: Neural network architecture used for classification of protein sequences. Orange boxes represent layers with trainable parameters, blue boxes are functions without parameters, and green boxes represent the input and output. Each length of the filters used in each CNN layer is written in parentheses. Global average pooling takes only non-zero values into consideration, producing a vector for each filter length. A depth-wise concatenation is performed, resulting in a single vector which is used for classification in the final batch-normalization and linear layer combination.

length of the sequence. Each entry in the matrix represents the result of evaluating a subsequence for the presence of a motif. The evaluation result is a log-probability that the motif is present at the corresponding site in the sequence. Each convolution contains multiple filters and thus, will look for multiple motifs. In our experiments, each layer consisted of five filters resulting in a total of $(24-4)5 = 100$ motifs. The filters can be of any length and are not limited to being consecutive as shown in Fig. 2.2.1. However, for simplicity, in our work we used consecutive filter lengths.

A bias vector of size f_m is added to each column of the matrix, followed by a rectified linear unit (ReLU) operation that acts as a threshold and indicates the presence or absence of a motif at each site. The average log-probability of the motifs detected in each row is calculated resulting in a vector $\mathbf{e_w}$. When calculating the entries of e_w, the denominator only takes the detected motifs into consideration, i.e., all entries greater than zero. As such, the denominator is increased by one to avoid division by zero. The purpose of calculating an average is to evenly distribute the backpropagation of error across all detected motifs similar to the technique used by [26]. The effectiveness of taking the average of motif evaluations has also been demonstrated in [16]. Considering only the detected motifs provides the benefits of

average pooling without the weak response caused by sparsity in the data; i.e., when a motif has a weak response in a sequence and most or all results are zero. When the input to the operation is sparse, that function acts more like max-pooling. To calculate the i-th term of e_w starting from a matrix M the following operation is used.

$$\mathbf{e_{w,i}} = \frac{\sum_{j=1}^{l-w+1} M_{i,j}}{\sum_{j=1}^{l-w+1} c(M_{i,j}) + 1}\,, \tag{1}$$

where

$$c(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}. \tag{2}$$

At this stage, a vector $\mathbf{e_w}$ exists for each motif length w. These vectors are concatenated into a single large vector, $\mathbf{e}$. Afterwards, batch normalization is applied to $\mathbf{e}$ to increase the rate of convergence and reduce overfitting [12]. Finally, a two-layer neural network is used for classification. A fully connected layer applies a linear transformation ($Wx + b$) with an output of size 150. A leakyReLU activation with a slope of 0.01 is used to avoid the dying ReLU [17]. Dropout with a drop rate of 20% is applied to help the model generalize. A second linear layer is applied with an output of size 142, the number of families. Finally, a softmax activation is applied resulting in a list representing the probability of the sequence being in each class.

2.2.3 Training

The model was trained on the processed Uniprot 2019 database described in Subsection Data Pre-processing. We randomly split the dataset into training and validation sets using the model selection package in Scikit-Learn [20]. The training set contains 80% of the total dataset, 85,344 examples, and the validation set contained 20%, 21,337 examples. Before each epoch, the performance of the model was evaluated and recorded on the validation set. Validation was performed once before training because we observed that the network quickly converged to a local minimum within the first epoch, making the AUROC consistently high. For our first experiment, the

network contained five motifs of each length from 5 to 24 inclusive. Five-fold cross validation was used to ensure that the network did not overfit the data.

We used cross-entropy loss between the network's prediction and the class label. We then applied label smoothing [24] to the class labels with $\epsilon = 0.05$. To account for any remaining class imbalance, each loss value was multiplied by a weight equal to the reciprocal of the number of examples in the class. The final loss function used was

$$\sum_{i=1}^{C} \left((1-\epsilon)\delta_{k,i} + \frac{\epsilon}{C} \right) \frac{1}{N_i} log(p_i) \,, \tag{3}$$

where p_i is the predicted probability of class i, N_i is the number of sequences present in class i, $\delta_{k,i}$ is the kronecker delta function (i.e., 1 if $k = i$, 0 otherwise) and k is the true class of the sequence, ϵ is the label smoothing coefficient, and C is the number of classes. We trained the network using the Adam optimizer [15].

2.2.4 Grid Search

We performed grid search to determine the effect that filter length has on prediction ability. A range of filter lengths $[\min, \max]$ was selected for each version of the model, where $5 \leq \min \leq \max \leq 24$. Consecutive filter lengths were assigned to the CNN layers. To keep the evaluation fair between each round of grid search, the models used a total of 100 filters distributed equally along the layers. If an equal number of filters could not be assigned to the CNNs, one filter was removed from each of the shortest CNNs until the model used a total of 100. This meant that the shorter layers contained $\left\lfloor \frac{100}{\max - \min} \right\rfloor$ filters and the longer layers contained $\left\lceil \frac{100}{\max - \min} \right\rceil$ filters. We ran the training loop described above for every combination of min and max. Each model was trained on the same train-validation split.

2.2.5 Evaluation Metrics

The network was evaluated during training by creating and visualizing receiver operating characteristic (ROC) curves and confusion matrices on the validation set at the end of each epoch. These allowed for the calculation of the AUROC, Accuracy,

Precision, Recall, F1-Score and Matthews-Correlation Coefficient (MCC). The latter five evaluation metrics were calculated using a macro-average as follows:

$$\begin{aligned}
\text{Accuracy} &= \frac{N_c}{N_s}\,, \\
\text{Precision}_f &= \frac{C_{f,f}}{P_f}\,, \\
\text{Recall}_f &= \frac{C_{f,f}}{T_f}\,, \\
\text{F1-Score}_f &= \frac{2 \cdot \text{Precision}_f \cdot \text{Recall}_f}{\text{Precision}_f + \text{Recall}_f}\,, \\
\text{MCC} &= \frac{N_c N_s - \sum_{i=1}^{N_f} T_i \cdot P_i}{\sqrt{\left(N_s^2 - \sum_{i=1}^{N_f} T_i^2\right)\left(N_s^2 - \sum_{i=1}^{N_f} P_i^2\right)}}\,,
\end{aligned} \tag{4}$$

where f is the index of a family in the dataset, N_f is the number of families, and C is the confusion matrix. N_c is the number of correctly predicted samples, N_s is the total number of samples in the validation set, P_f is the number of samples predicted to be in family f and T_f is the number of samples in family f. They are calculated as follows:

$$\begin{aligned}
N_c &= \sum_{i=1}^{N_f} C_{i,i}\,, \\
N_s &= \sum_{i=1}^{N_f} \sum_{j=1}^{N_f} C_{i,j}\,, \\
P_f &= \sum_{i=1}^{N_f} C_{f,i}\,, \\
T_f &= \sum_{i=1}^{N_f} C_{i,f}\,.
\end{aligned} \tag{5}$$

The macro-average of each measure is calculated as follows:

$$\frac{\sum_f M_f}{N_f}\,, \tag{6}$$

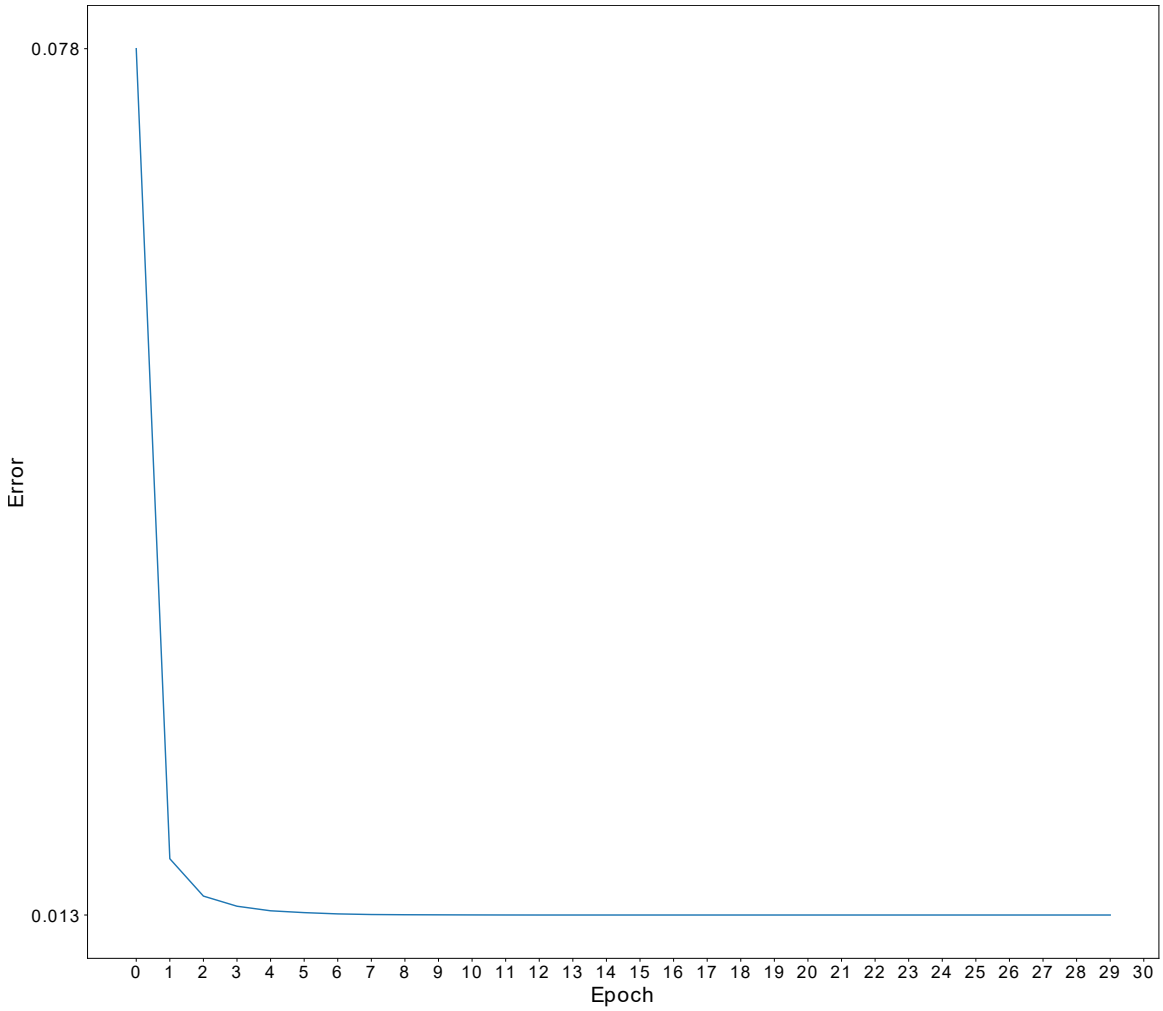

Fig. 2.3.1: Cross-entropy loss of the network evaluated on the validation set before each epoch of training.

where M_f is any metric in Eq. (4) that can be evaluated for a family f.

2.2.6 Implementation Details

DeePSLiM was implemented in PyTorch [19]. Statistical analysis was performed using Scikit-learn [20] and Numpy [25]. Plots were created using Matplotlib [11]. All experiments were performed on the Compute Canada Cedar server cluster using an NVIDIA P100 Pascal GPU (`http://www.computecanada.ca`).

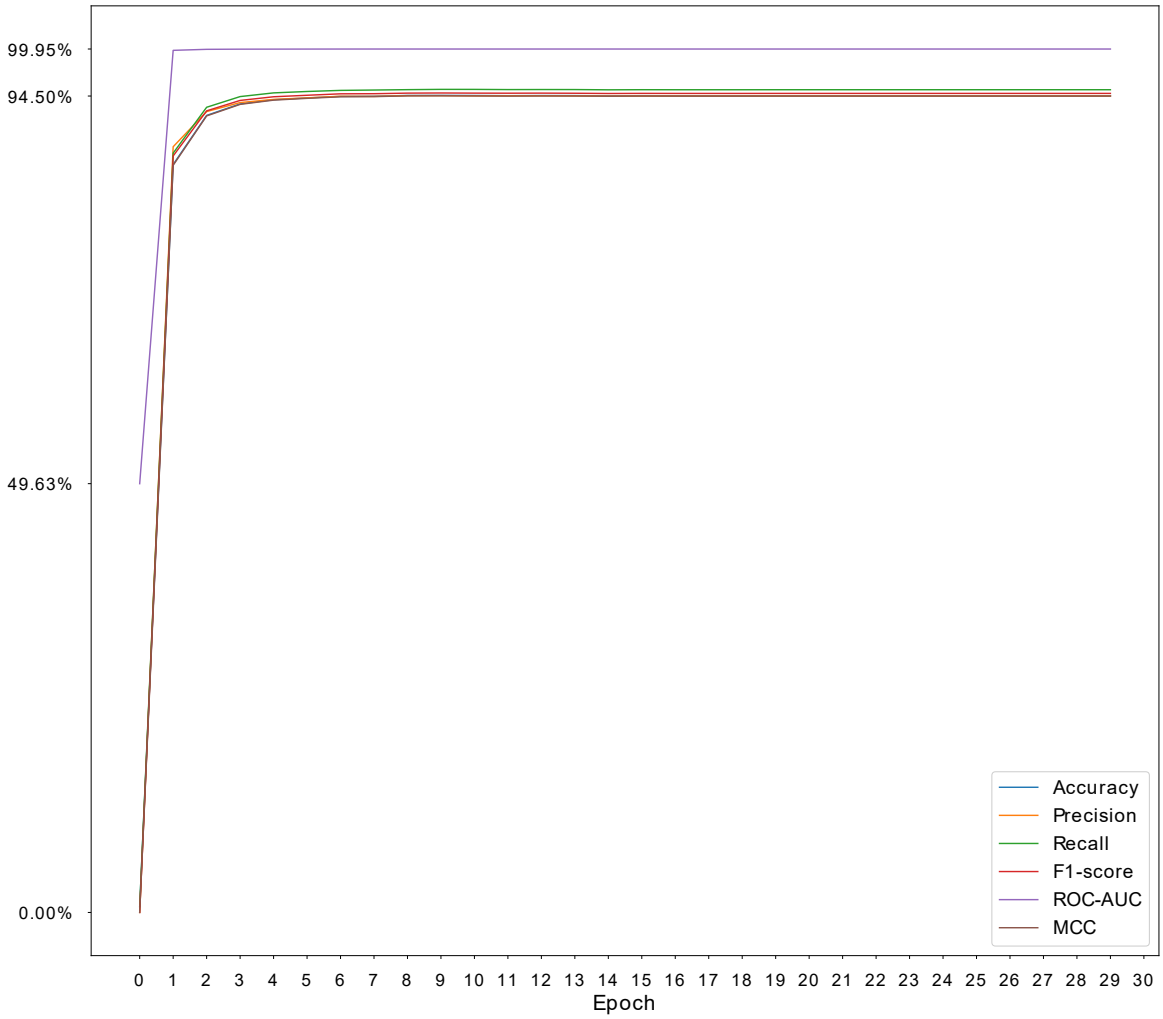

Fig. 2.3.2: Accuracy, precision, recall, F1-Score, MCC and AUROC of the model evaluated on the validation set before each epoch.

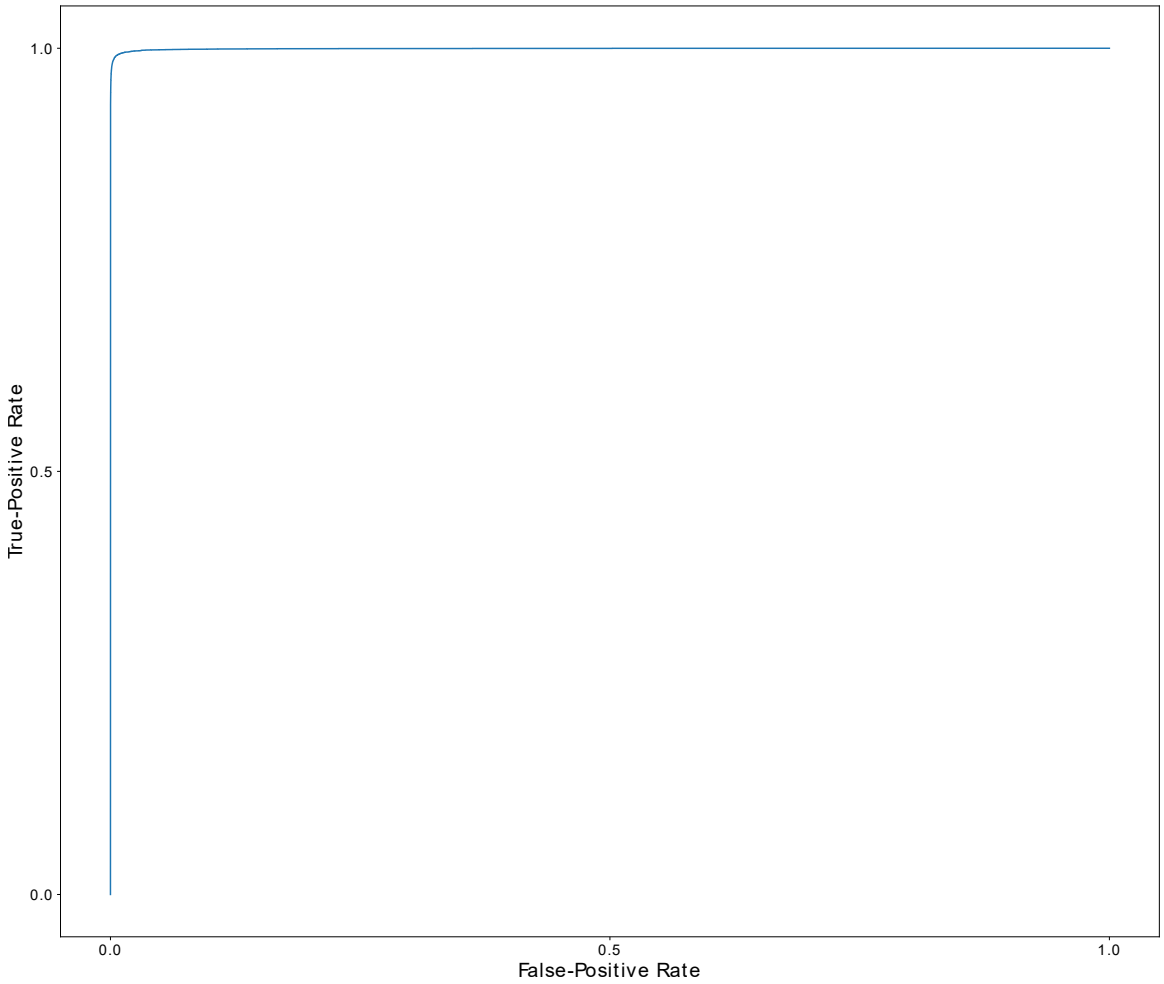

Fig. 2.3.3: ROC curve of the trained network.

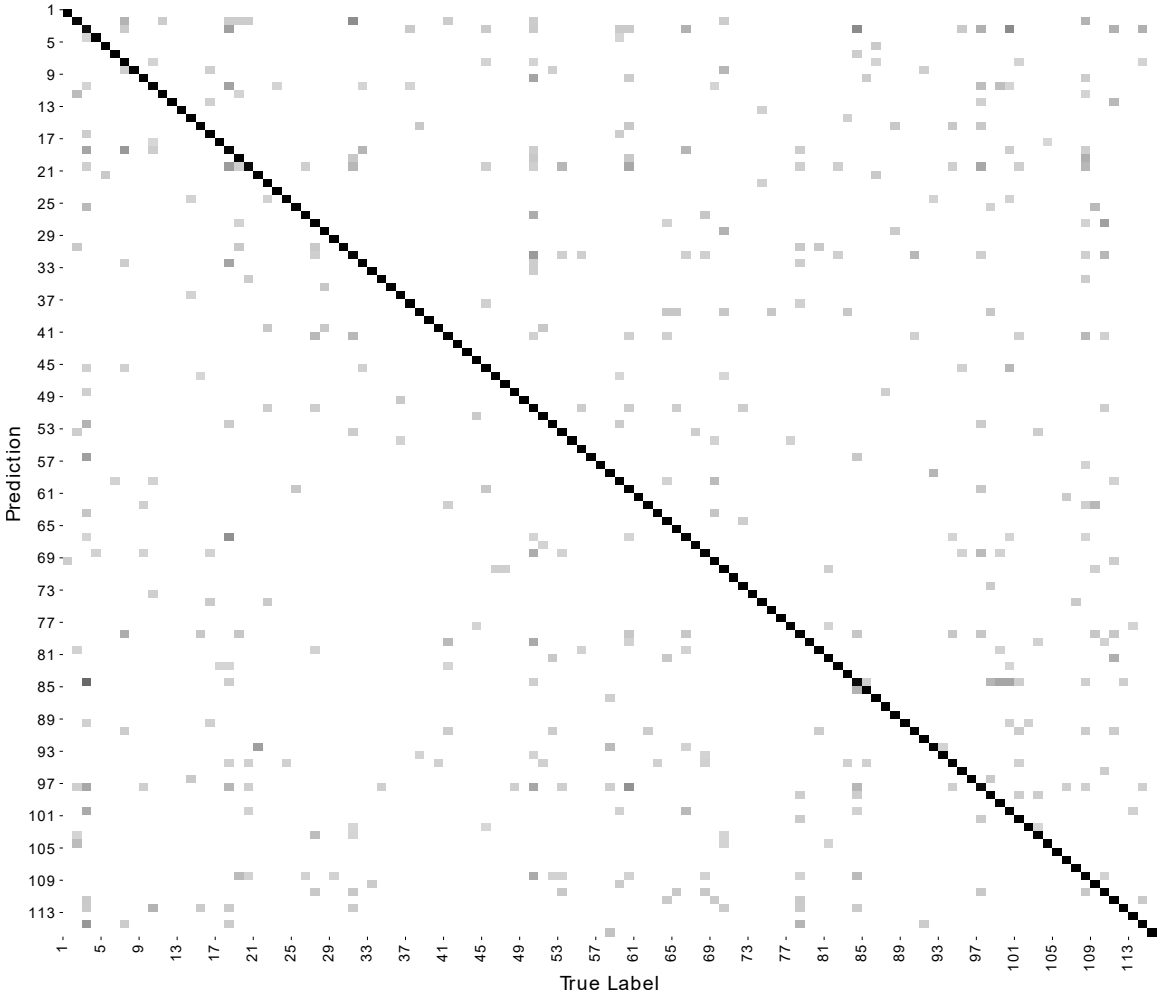

Fig. 2.3.4: Confusion matrix of the trained neural network. Darker cells represent a higher count than lighter cells. The cells are drawn on a logarithmic scale to make the lightest colored erroneous (off-diagonal) values more visible.

2.3 Results

We first trained our model with filters of lengths ranging between 5 and 24 inclusive. Each CNN layer contained five filters of it's respective length. The Adam optimizer used the following parameters: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$ and exponential learning rate decay of 0.5. A batch size of 64 was used, and the model was trained for 30 epochs. It is worth mentioning that we trained the model for over 100 at first. However, due to the small number of parameters, the model reached a plateau at which it stabilized for the remainder of training.

The training history is visualized in Fig. 2.3.1, and the history of metrics from Eq. (4) are shown in Fig. 2.3.2. The model quickly reached a stable local minimum after the first epoch resulting in a score close to 90% on all evaluation metrics. By the seventh epoch, the model converged and reached a plateau where it stayed for the remainder of the training phase. The value of the AUROC of the final model was 99.9% and the values accuracy, precision, recall, F1-Score, and MCC were all roughly the same, approximately 96%.

To evaluate the performance of the network during training, a ROC curve and confusion matrix were created for the validation set before the start of each epoch. The plots for the final network are shown in Figs. 2.3.3 and 2.3.4, respectively. These measures act as further evidence for the model's predictive power, despite the low number of parameters.

2.3.1 Grid Search

The heatmaps shown in Figs. 2.3.5 and 2.3.6 show the cross-entropy loss and Matthews correlation coefficient, respectively, of each neural network trained during grid search. Each cell in the heatmap represents a network with filters of length ranging from the corresponding number on the x-axis to the one on the y-axis. The remaining heatmaps for accuracy, precision, recall, F1-Score and AUROC are similar to Fig. 2.3.6 and can be found in the appendix.

We extracted motifs from the CNN filters of the trained model in position prob-

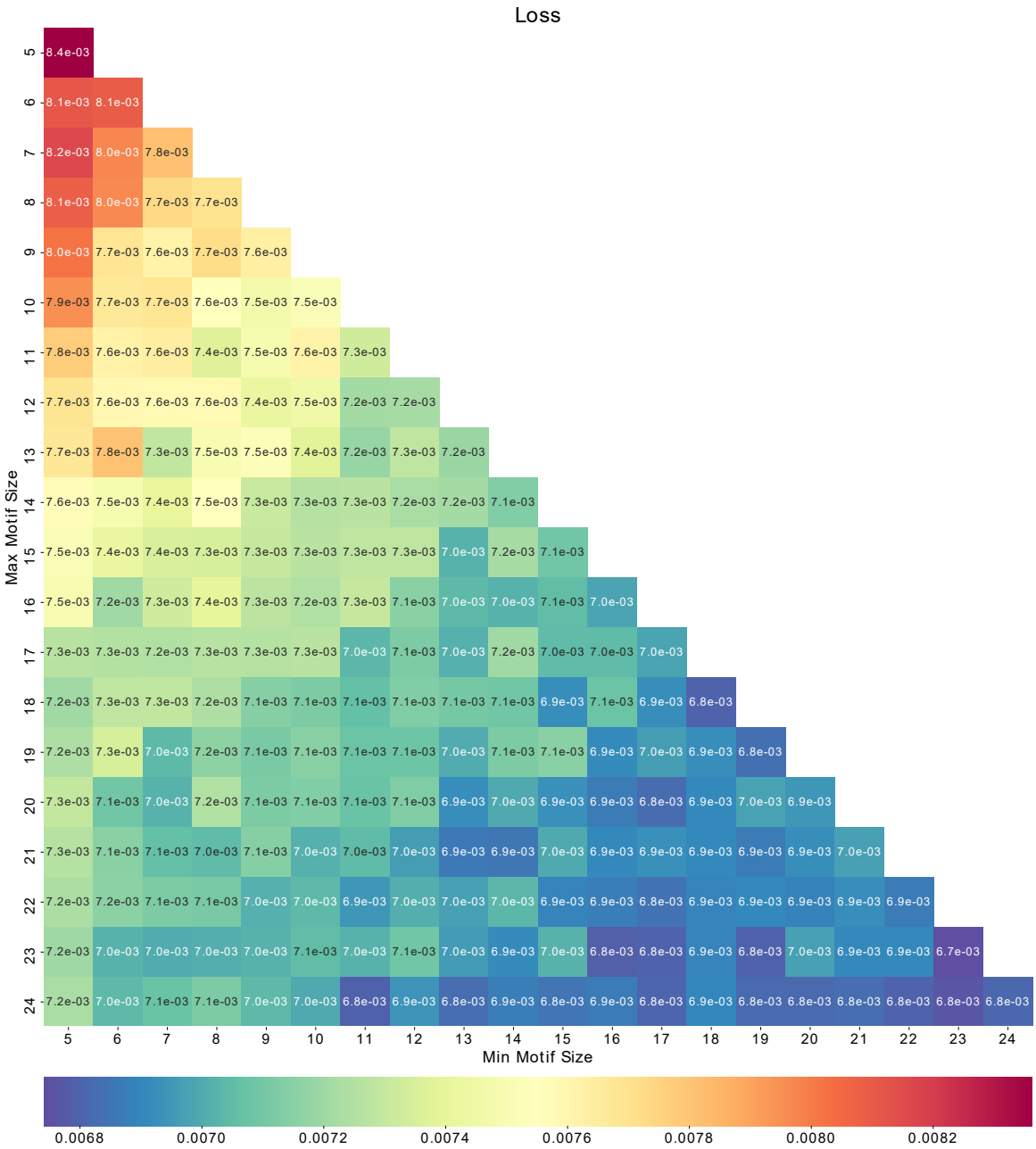

Fig. 2.3.5: Heatmap of the cross-entropy loss for each of the networks in the grid search. Each box represents a network with motif lengths within the range [min, max]. The diagonal represents networks, where all motifs are of the same length.

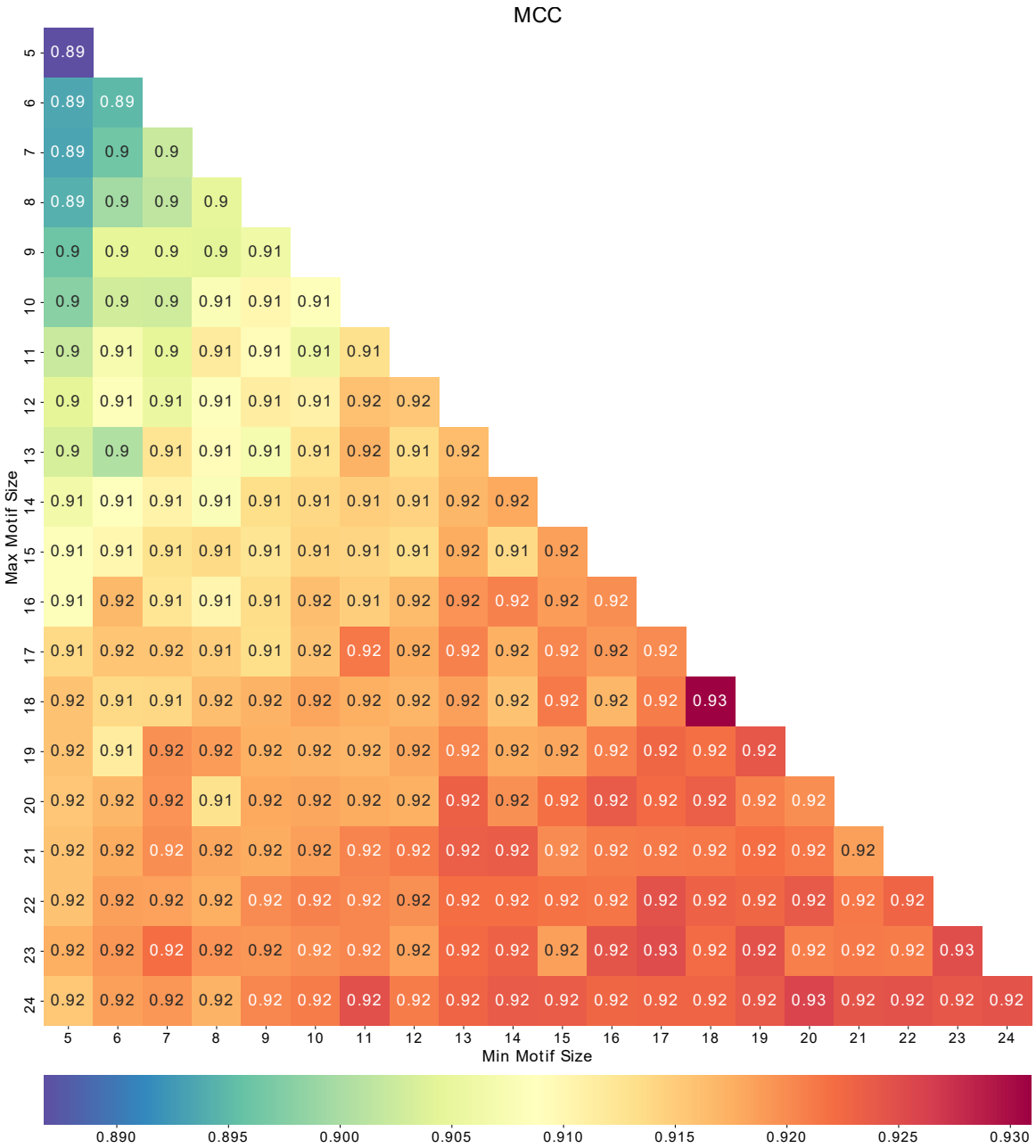

Fig. 2.3.6: Heatmap of the Matthews-Correlation coefficient for each of the networks in the grid search. Each box represents a network with motif lengths within the range $[\min, \max]$. The diagonal represents networks, where all motifs are of the same length.

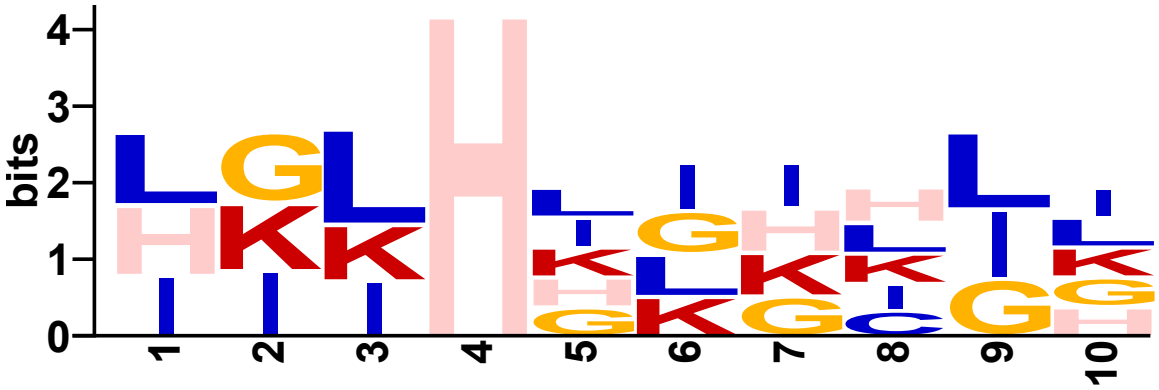

Fig. 2.3.7: Example of a motif of length 10 extracted from the model.

ability matrix (PPM) format using the method described in DeepBind [1]. We then used TomTom [9] to determine which of these motifs most closely resembled known motifs, calculate their E-values [10] and display motif logos. Fig. 2.3.7 shows one such motif, that was discovered by the network, with an E-value of 4.438×10^{-4}. The remaining motifs can be found in the appendix.

2.4 Discussion

While training the model, a minutious approach was followed to avoid overfitting during the training process. This included the use of batch normalization and dropout in the classifier, as well as splitting the dataset into training and validation sets. The validation set contained one fifth of the sequences in the entire dataset. During the training process, the model was evaluated on the validation dataset at the beginning each epoch. The model's loss did not increase in later epochs and all other evaluation metrics did not decrease, as shown in Figs. 2.3.1 and 2.3.2, respectively. Instead, the metrics reached a plateau and stayed at the same level for the remainder of training. This indicates that the model was able to generalize to the entire dataset.

The plot in Fig. 2.3.2 shows that accuracy, precision, recall, F1-Score and MCC had similar results throughout training, ending between 94% and 95%. The most important of these measures is MCC as it has been shown to be a better measure of a model's performance when dealing with imbalanced classes [5]. The implication

here is that the steps taken toward mitigating the effects of the imbalanced classes in the dataset were successful. These steps were filtering out classes with fewer than 500 examples and the use of class weights and label smoothing in the loss function.

We were not able to perform a direct comparison with any of the models mentioned previously due to incompatibilities with the code or the datasets. In some cases, the authors did not make their code available. Others used outdated versions of libraries that were unavailable to us or gave us many errors. We did, however, run our own model with modifications that made it similar to DeepBind. The grid search demonstrated the effect that filter lengths play in the classification criteria. Although using large filters did slightly improve the classification criteria, we demonstrated that shortening using filters of multiple lengths performed just as well in many cases and used less memory. Furthermore, the use of our tradeoff pooling strategy and batch normalization played a major role in the training of our network. First, we tried to use max pooling and average pooling after the convolutional layer. These methods did not perform as well as the tradeoff strategy, leading us to believe that there is a good compromise between max and average pooling, at least, when performing global pooling. Finally, in our tests on the network's classifier, we found that batch normalization played a crucial role in training the network. Without a BatchNorm layer the model performed very poorly, in some cases, not reaching any stable minimum. Only by adding BatchNorm to the classifier, before one of the two linear layers, were we able to consistently train the network with such good results shown in Figs. 2.3.1 and 2.3.2.

As shown in Fig. 2.3.6, networks with longer SLiMs performed better than those with shorter SLiMs. This may indicate that longer SLiMs in proteins are more highly conserved. Perhaps, this is due to the conservation of specific functions such as enzymatic activity, protein-protein interaction sites, cleavage sites, or structural stability. Being able to recognize these longer SLiMs would provide the model more information about each class and give other filters the freedom to recognize other SLiMs in the dataset. This may reduce the chance of different filters recognizing different parts of the same motif as well. On the other hand, the better performance may be due to

the fact that networks with longer filters contain more trainable weights giving them more degrees of freedom during the training process.

The largest performance difference observed during the grid search was approximately 4%, ranging from 93% to 97%. The limit of the model seems to be approximately 97% due to the fact that networks with shorter filters, and thus fewer trainable weights, performed as well as the model with filters of length 24. If it is necessary to achieve higher performance, the network can be enhanced by adding more layers to the classifier. This was avoided during our experiments for two reasons. The first is a common pattern that relatively shallow neural networks tend to perform very well as long as a large dataset is used [27, 13, 28]. The second is to prevent the classifier from doing the work that the motifs should perform, allowing for the network to find high-quality motifs that do not require a deep network with many parameters.

The SLiMs extracted from the model are shown in Fig. 2.3.7 and in Figs. S11 through S18 of the Supplementary Material. One challenge faced by our model is the detection of the same motif by separate motif detectors. This is a common problem faced in other motif detection algorithms and the convergence of multiple filters on the same feature is known to occur in other deep learning tasks as well. However, the way that our model is structured provides one large advantage over other motif detection algorithms. A neural network trained on a binary classification task needs to detect motifs present in each class but not present in both. Common motifs are left out, thus, losing meaningful information. By introducing multiple classes, our model has the potential to discover those motifs that different classes have in common as well as those they do not. These can be used to study both similarities and differences in proteins of different classes without the need to run multiple algorithms on the dataset.

References

[1] Babak Alipanahi et al. "Predicting the sequence specificities of DNA-and RNA-binding proteins by deep learning". In: *Nature Biotechnology* 33.8 (2015), pp. 831–838.

[2] Timothy L Bailey. "DREME: motif discovery in transcription factor ChIP-seq data". In: *Bioinformatics* 27.12 (2011), pp. 1653–1659.

[3] Timothy L Bailey et al. "MEME SUITE: tools for motif discovery and searching". In: *Nucleic Acids Research* 37.suppl_2 (2009), W202–W208.

[4] Helen M Berman et al. "The protein data bank". In: *Nucleic Acids Research* 28.1 (2000), pp. 235–242.

[5] Davide Chicco and Giuseppe Jurman. "The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation". In: *BMC Genomics* 21.1 (2020), p. 6.

[6] UniProt Consortium. "UniProt: a worldwide hub of protein knowledge". In: *Nucleic Acids Research* 47.D1 (2019), pp. D506–D515.

[7] Fiona Cunningham et al. "Ensembl 2019". In: *Nucleic Acids Research* 47.D1 (2019), pp. D745–D751.

[8] Sara El-Gebali et al. "The Pfam protein families database in 2019". In: *Nucleic Acids Research* 47.D1 (2019), pp. D427–D432.

[9] Shobhit Gupta et al. "Quantifying similarity between motifs". In: *Genome Biology* 8.2 (2007), R24.

[10] Gerald Z Hertz and Gary D. Stormo. "Identifying DNA and protein patterns with statistically significant alignments of multiple sequences." In: *Bioinformatics (Oxford, England)* 15.7 (1999), pp. 563–577.

[11] John D Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in science & engineering* 9.3 (2007), pp. 90–95.

[12] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *arXiv preprint arXiv:1502.03167* (2015).

[13] David R Kelley, Jasper Snoek, and John L Rinn. "Basset: learning the regulatory code of the accessible genome with deep convolutional neural networks". In: *Genome Research* 26.7 (2016), pp. 990–999.

[14] Sameer Khurana et al. "DeepSol: a deep learning framework for sequence-based protein solubility prediction". In: *Bioinformatics* 34.15 (2018), pp. 2605–2613.

[15] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[16] Yixun Li et al. "The predictive performance of short-linear motif features in the prediction of calmodulin-binding proteins". In: *BMC Bioinformatics* 19.14 (2018), p. 410.

[17] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. "Rectifier nonlinearities improve neural network acoustic models". In: *Proc. icml*. Vol. 30. 1. 2013, p. 3.

[18] Andrew F Neuwald and Jun S Liu. "Gapped alignment of protein sequence motifs through Monte Carlo optimization of a hidden Markov model". In: *BMC Bioinformatics* 5.1 (2004), p. 157.

[19] Adam Paszke et al. "Automatic differentiation in PyTorch". In: (2017).

[20] Fabian Pedregosa et al. "Scikit-learn: Machine learning in Python". In: *The Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[21] Roman Prytuliak et al. "HH-MOTiF: de novo detection of short linear motifs in proteins by Hidden Markov Model comparisons". In: *Nucleic Acids Research* 45.W1 (2017), W470–W477.

[22] Daniel Quang and Xiaohui Xie. "DanQ: a hybrid convolutional and recurrent deep neural network for quantifying the function of DNA sequences". In: *Nucleic Acids Research* 44.11 (2016), e107–e107.

[23] Christian JA Sigrist et al. "PROSITE: a documented database using patterns and profiles as motif descriptors". In: *Briefings in Bioinformatics* 3.3 (2002), pp. 265–274.

[24] Christian Szegedy et al. "Rethinking the inception architecture for computer vision". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2818–2826.

[25] Stéfan van der Walt, S Chris Colbert, and Gael Varoquaux. "The NumPy array: a structure for efficient numerical computation". In: *Computing in Science & Engineering* 13.2 (2011), pp. 22–30.

[26] Bolei Zhou et al. "Learning deep features for discriminative localization". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2921–2929.

[27] Jian Zhou and Olga G Troyanskaya. "Predicting effects of noncoding variants with deep learning–based sequence model". In: *Nature Methods* 12.10 (2015), pp. 931–934.

[28] James Zou et al. "A primer on deep learning in genomics". In: *Nature Genetics* 51.1 (2019), pp. 12–18.

CHAPTER 3

Conclusion and Future Work

3.1 Conclusion

This thesis presents DeePSLiM, an enhanced neural network model that is able to classify protein sequences and detect sequence motifs. Our model can be trained quickly on large quantities of data, learning to detect SLiMs of multiple lengths and simultaneously creating a classifier of the protein sequences that it was trained on. The model is able to differentiate between many families, performing well on many evaluation metrics, including AUROC and MCC, indicating that it is robust against imbalanced classes.

Potential areas of future work include the classification of proteins based on similar functions, structural patterns, cell localization or any other multi-class task. These may provide meaningful results relating to the chosen label. One of the challenges that we faced was the detection of the same motif by multiple motif detectors. A method to prevent motif detectors from being too similar to one another may provide additional benefits to our model and may have broader applications in the field of deep learning. Furthermore, determination of the best filter lengths to use when training the model is also a possible avenue for extending this work.

3.1.1 Contributions

The contributions of this thesis can be summarized as follows:

- Proposed a neural network model, called DeePSLiM, that is able to differentiate proteins into their respective families based on their amino acid sequences.

- Proposed a method to detect SLiMs of different lengths by use of multiple convolutional neural network layers evaluating the input sequence in parallel.
- Proposed a novel pooling method that serves as a tradeoff between average-pooling and max-pooling to capture the strengths of each method and decrease training time.
- Implemented DeePSLiM as a Python package, which can be found at [1].

3.2 Future Work

In this work, we demonstrated how DeePSLiM can be used to find motifs of multiple lengths by training on a labeled set of protein sequences. Future work may involve training DeePSLiM on a dataset with class labels that contain information about physical properties of the proteins, such as their structures or functions. The model can be applied to a wide variety of datasets with two or more class labels. Another area for future work involves the model's tendency to train multiple motif detectors to recognize the same motif. A method of preventing motifs from being too similar to one another may help in the discovery of more motifs within and between protein families. For example, a similarity function between SLiMs can be used by defining the total loss of the network as the sum of the cross-entropy loss and similarity of motifs. This would force the network to find SLiMs that are different from one another. The similarity function must work using the position-weight matrix representation of SLiMs and must be differertiable for use with the backpropagation algorithm.

Further research must be done to determine the best filter lengths to use for a dataset. In our work we used a grid search method to determine that the model reaches a point of diminishing returns as the length of motifs is increased. The performance of models with motifs of length 11 and 12 is very similar to those with motifs of length 24. After this point the amount of memory needed to store the model becomes an issue. A fast method to approximate the optimal lengths of motifs would be useful as grid search is a very time consuming process when training on large

amounts of data.

To summarize, this work can be further extended as follows:

- Determination of the model's applicability to other classification criteria such as protein structures and functions.
- The model has a tendency to train multiple motif detectors to find the same motif. A method is needed to prevent this from occurring.
- Determination of the best filter lengths without the use of a time intensive process, like grid search.

References

[1] Alexandru Filip. *Github Repo: DeePSLiM*. `https://github.com/AlexFilip/DeePSLiM`. 2020.

VITA AUCTORIS

NAME:	Alexandru Filip
PLACE OF BIRTH:	Suceava, Suceava, Romania
EDUCATION:	BSc. (Honours) Biochemistry, University of Windsor, Windsor, Ontario, Canada, 2014
	BSc. Computer Science, University of Windsor, Windsor, Ontario, Canada, 2018
	M.Sc. Computer Science, University of Windsor, Windsor, Ontario, Canada, 2020