Master’s Thesis 2023 30 ECTS
Faculty of Science and Technology

Deciphering Transcriptional Regulation using Deep Neural Networks

Julie Wollebæk Førrisdal
Data Science

Julie Wollebæk Førrisdal
Author

Torgeir Rhoden Hvidsten
Supervisor

Co-supervisors
Lars Grønvold
Kristian Hovde Liland
Simen Rød Sandve

PREFACE

This thesis completes my data science studies at the Norwegian University of Life Sciences (NMBU). I want to thank my supervisor, Dean of the Faculty of Chemistry, Biotechnology and Food Science Togeir R Hvidsten, for providing a challenging and fascinating research topic. Further, I thank my co-supervisors, Researcher Lars Grønvold and Professor Simen Rød Sandve at the Faculty of Biosciences, for making their data and biological knowledge available. I thank Professor Kristian Hovde Liland at the Faculty of Science and Technology, for his data science expertise. I thank you all collectively for the great guidance and feedback, and consider myself lucky to have had such engaged, positive and witty supervisors this semester.

Thanks to all my friends and family for keeping up the support and interest in my work, even when I consequently failed to explain any of it in lay man's terms. A special thanks to my classmates at Studentenes Hus (TF6-206), it was perhaps a little too much fun studying together. Lastly, I would like to thank my dearest mother and beloved partner for their unconditional love and support that helped me through all of this.

Julie W. Førrisdal

May 15, 2023, Ås

ABSTRACT

The DNA holds the recipe of all life functions. To decipher the instructions, one has to learn and understand its complex syntax. The non-coding DNA contains regulatory elements, that are essential to control and activate gene expression in the right place at the right time. Previous studies have applied deep learning for gene expression prediction, directly from non-coding sequences, successfully. Almeida et al. [1] showed that a Convolutional Neural Network could learn regulatory syntax from long same-length fragments from the fruit fly. In this thesis, we tested how well deep neural networks could predict gene expression from short DNA fragments of varying lengths from the Atlantic salmon. Furthermore, we extracted what the models had learned, and tested if the sequence features corresponded to known regulatory sequence patterns (motifs).

Two deep neural network architectures were built, a Convolutional Neural Network (CNN) and a hybrid Convolutional and Long Short-Term Memory Neural Network (CNN-LSTM). We trained the models to predict the gene expression effect of DNA fragments from open chromatin of liver cells. The two model architectures performed equally well, and the performances depended on the amount of noise in the validation data, reaching a correlation of 0.68 on the sequences of top 10% base mean.

We extracted motifs both from the first convolutional filters and from sequence importance scores, and we compared the motifs to the JASPAR database of known vertebrate transcription factor binding site motifs. Among the significant matches to JASPAR, we found some general transcription factors like the TFCP2, HSF and AP-1, as well as some liver-specific transcription factors like the KLF15 and HNF6. Most motifs did not match any JASPAR motif. We explained the tendency of CNNs to distribute partial motifs across several filters, and that other sequence features might be important for prediction as well. Our results suggest that the models learned regulatory DNA syntax equally well, despite their different architectures, and we compared the motif findings in light of these differences.

This thesis demonstrates the potential of deep neural networks for analysis of ATAC-STARR-seq data, and suggests improvements worth exploring further to possibly increase performance. We also stress the need for more robust model interpretation techniques, which could unlock valuable knowledge in the future of genomics.

ABBREVIATIONS

ANN Artificial Neural Network.

ATAC-seq Assay for Transposase-Accessible Chromatin using sequencing.

BRNN Bidirectional Recurrent Neural Network.

CNN Convolutional Neural Network.

CRE Cis-Regulatory Element.

DNA Deoxyribonucleic Acid.

GRU Gated Recurrent Unit.

LSTM Long Short-Term Memory.

MLP Multilayer Perceptron.

MPRA Massively Parallel Reporter Assay.

RNA Ribonucleic Acid.

RNN Recurrent Neural Network.

STARR-seq Self-Transcribing Active Regulatory Region sequencing.

TSS Transcription Start Site.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1 INTRODUCTION

Ever since it was possible to read the DNA of organisms, predicting its functions has been a prominent goal in biology. Coding sequences (genes) consist of triplet nucleotides (codons) that code for the chain of amino acids that make up a specific protein. Codons are known and easy to decode since they exclusively represent one amino acid. These properties make coding sequences recognizable, and state-of-the-art gene prediction tools have accomplished highly accurate results [2–4]. However, the majority of a eukaryotic genome is non-coding, which was previously assumed to be "junk DNA". It is now known that the non-coding genome is essential for regulating genes. Promoters and enhancers are non-coding sequences that can regulate the activity of genes. They interact with both near and distal genomic regions, and are influenced by developmental stages and cell environment. Furthermore, their sequences are complex and vary between species, tissues, and cell types [5]. Thus, the non-coding genome is written in a complex syntax dependent on context, comparable to natural language [6]. Regulatory syntax and natural language are arguably similar in the way language is built on rules and grammar, but still highly variable in interpretation. Promoters and enhancers can be thought of as "sentences" that are made up of "words" (motifs) recognizable by individual regulatory proteins. Recent research applying Deep Neural Networks, a type of machine learning algorithm, to process natural language has been a success [7], and the same techniques are being explored for analysis of DNA sequences [8]. Robust networks that understand raw DNA sequences and predict their functions reliably under unseen conditions do not exist yet, but abundant data, wide interest, and the will to invest in such technology is paving the way for further development.

1.1 Motivation

There are several useful research applications for a DNA-language model well-trained to predict across genomic loci. Interpreting what features and general patterns the model learned to look for leads to discovery of novel regulatory rules and motifs [9]. As defined by Bailey and Elkan [10], motifs are approximate statistical representations of a sequence pattern believed to have a biological function, such as being a binding site for a regulatory protein. By training a generative model, it is possible to design synthetic motifs with desired regulatory effects [11]. Another application is to get predictions on variant effects by passing mutated sequences to a well-trained model [12].

This thesis aims to explore deep neural networks for analyzing non-coding sequences. In a broader context, we study the inner workings of gene regulation in domestic Atlantic salmon.

To improve cost efficiency and animal welfare in the salmon industry, it is important to know how the domestic salmon responds to different conditions, including diets and medical treatments. Overall, the knowledge of the functions and interactions of genetic elements and how they affect transcription of genes, will be fundamental for the future of animal breeding, food science, drug development and medicine. Interpretation of deep models is one way to unlock this knowledge.

1.1.1 Related Work

Almeida et al. [1] built a deep Convolutional Neural Network called DeepSTARR to predict enhancer activity in fruit flies (*Drosophila melanogaster*) based on DNA sequence. They validated the model experimentally and extracted motifs that are predictive of enhancer activity. These motif rules were adjusted to human enhancers, such that synthetic enhancers could be designed with desired activity levels. Their work demonstrated that a Convolutional Neural Network could predict enhancer activity from sequences of 249 nucleotides, obtained by UMI-STARR-seq, a Massively Parallel Reporter Assay (MPRA) experiment.

Quang and Xie [13] proposed a hybrid neural network called DanQ to predict the function of non-coding human DNA sequences. They used a combination of a convolutional layer and a bidirectional recurrent layer to capture regulatory motifs and long-term dependencies between motifs. The authors showed that a hybrid model outperformed other models in predicting the non-coding function from sequences of 1000 nucleotides.

1.2 Objectives

Deep neural networks have shown an ability to learn regulatory grammar in some species and for fixed DNA sequence lengths. Most previous studies have been conducted on a few widely studied genomes, but few studies have tried the same on the domestic Atlantic salmon, an important livestock for food production. Another interesting aspect to study is the robustness to variable input sequence lengths. To the best of our knowledge, no studies have applied deep neural networks to process data produced by the ATAC-STARR-seq method, a novel MPRA producing sequences that are largely non-overlapping and of varying lengths [14]. We propose the following objectives for this thesis:

- How does a deep neural network perform at predicting the gene regulation effect of DNA sequences from ATAC-STARR-seq data?
- Can a deep neural network learn to recognize regulatory sequence motifs from the Atlantic salmon?

We aim to answer these research questions by testing different deep learning approaches for predicting to what degree sequence fragments from the salmon genome can drive transcription. And if so, can the sequence features that drive transcription be extracted, and can they be associated with, e.g., known transcription factor binding sites?

1.3 Structure

The remaining work is structured in the following way: Relevant theoretical concepts will firstly be covered in Chapter 2. Chapter 3 then describes how a dataset of DNA sequences was created from liver cells of Atlantic salmon, followed by a detailed methodology of our Deep Learning approach to analyze this data. Next, results are presented in Chapter 4. The results are first discussed and later concluded in the final Chapter 5 and Chapter 6. We include supplementary information, such as detailed software specifications, as Appendices.

CHAPTER 2 THEORY

In this chapter, theory behind gene regulation and deep learning will be explained to provide a basis for our deep learning approach to DNA sequence and gene regulation analysis.

2.1 Gene regulation

Even though all cells in eukaryote species contain the same genome sequence in their nucleus, cells in e.g. heart, liver, and blood are drastically different from one another. Gene regulation is what enables cells to divide and differentiate into different cell types, carry out specialized functions, and adapt and respond to environmental changes. Gene regulation can occur at several levels, including enhancer and promoter activation or repression, transcription factor binding and chromatin remodeling [5]. The paragraphs under Section 2.1 will cover these biological concepts that are relevant for prediction of gene expression.

2.1.1 Gene expression

Gene expression is the process by which a gene product is synthesized from DNA. Deoxyribonucleic Acid (DNA) is a long chain of nucleotides, and is illustrated in Figure 2.1. There are four nucleotides, Adenine (A), Cytosine (C), Guanine (G) and Thymine (T). A fifth nucleotide Uracil (U) substitutes Thymine when the sequence is transcribed into Ribonucleic Acid (RNA). The central dogma of molecular biology states that DNA is transcribed into RNA, which is then translated into protein (see Figure 2.2). Transcription is performed by proteins called polymerases, and the polymerase activity is regulated by other molecules such as transcription factors (see Figure 2.3) [5].

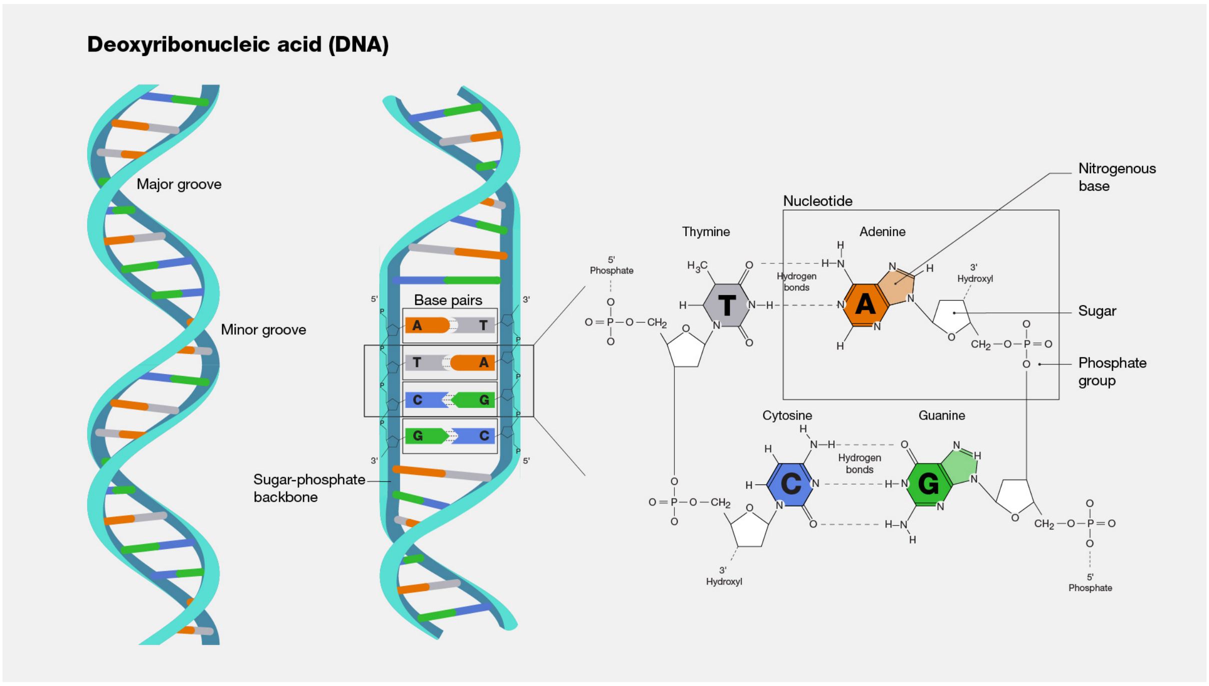


Figure 2.1: Deoxyribonucleic Acid (DNA) is a long chain of nucleotide pairs coiled together into the double helix shape. The nucleotides pair with each other through hydrogen bonds, and are supported by the sugar-phosphate backbone. Courtesy: National Human Genome Research Institute [15] (public domain).

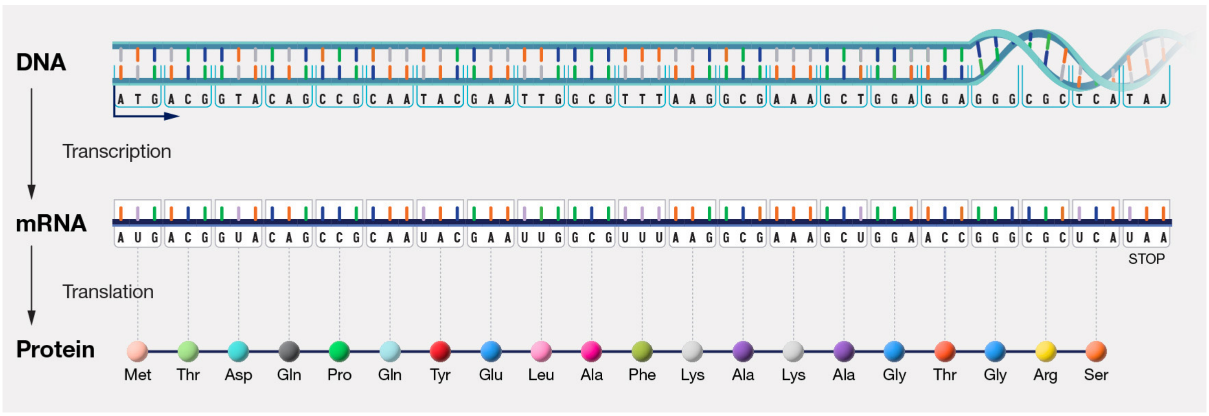

DNA
ATG ACG GTA CAG CCG CAA TAC GAA TTG GCG TTT AAG GCG AAA GCT GGA GGA GGG CGC TCA TAA
Transcription
mRNA
AUG ACG GUA CAG CCG CAA UAC GAA UUG GCG UUU AAG GCG AAA GCU GGA ACC GGG CGC UCA UAA
STOP
Translation
Protein
Met Thr Asp Gln Pro Gln Tyr Glu Leu Ala Phe Lys Ala Lys Ala Gly Thr Gly Arg Ser

Figure 2.2: The central dogma of molecular biology explains the flow of information from DNA to RNA to protein, mediated by transcription and translation of sequences. Courtesy: National Human Genome Research Institute [16] (public domain).

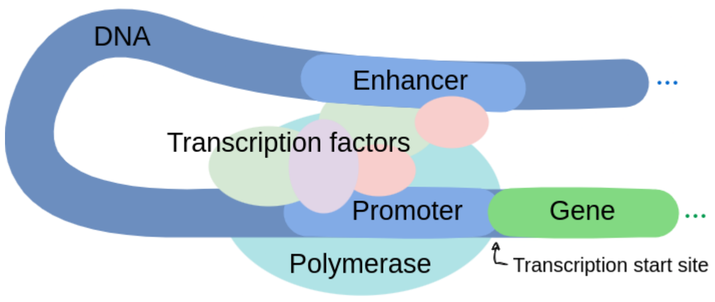


Figure 2.3: Transcriptional regulation. Transcription factors bind to cis-regulatory elements like enhancers and promoters and control the initiation of transcription.

2.1.2 Transcriptional regulation

An important mechanism of gene regulation in eukaryotes is transcriptional regulation, which controls the initiation of transcription in a temporal and spatial manner. Transcriptional regulation can occur through various biochemical mechanisms [5]. One of these mechanisms is binding of transcription factor proteins to DNA sequences near genes, referred to as Cis-Regulatory Elements (CREs). There are generally two types of CREs, called promoters and enhancers. Definitions of promoters and enhancers vary, but in terms of function, promoters are sequences where RNA-polymerases bind and initiate transcription. Promoters could more broadly also include sequences located nearby (approximately 1000 bp) upstream of a Transcription Start Site (TSS) that influence the binding. Promoters are therefore thought of as prerequisites for a gene to maintain basal gene expression levels. Enhancers are, again in terms of function, sequences that modify gene expression. They can be located nearby (100 to thousands of bp) or far away ($>$ 1 000 000 bp) from the start site in any direction, or even in the middle of introns within other genes. Enhancers affect gene expression in cell- and tissue-specific ways. An enhancer that decreases gene expression is often referred to as a repressor or silencer [5].

2.1.3 Chromatin accessibility

Illustrated in Figure 2.4 (left-most panel) is a chromosome during the metaphase of cell division, and its unraveled components (panels towards the right). Here we see that the chromatin organization, i.e., the DNA and associated proteins, is densely packed. The chromosome is made up of chromatin, which consists mostly of DNA wrapped around histone proteins called nucleosomes, that is neatly organized into chromatin fiber. The main purpose of chromatin is to efficiently and compactly organize all the DNA within the nucleus, but it also serves a regulatory function. The most tightly packed parts of the DNA are physically inaccessible for gene regulatory molecules and polymerases, simply due to the chromatin density. By remodeling the chromatin structure to be more open, rendering parts of the DNA more accessible, the polymerase and other regulatory proteins can bind to the DNA and gene transcription can be induced. These changes include chemical modifications to DNA and histones [5].

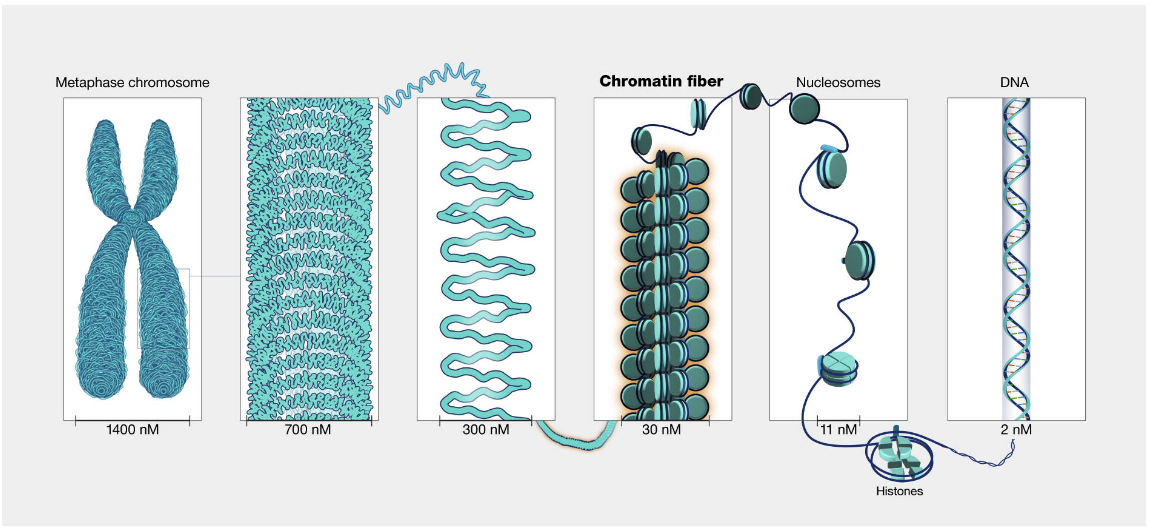


Figure 2.4: The genome is densely packed into chromatin. Courtesy: National Human Genome Research Institute [17] (public domain).

2.1.4 ATAC-STARR-seq

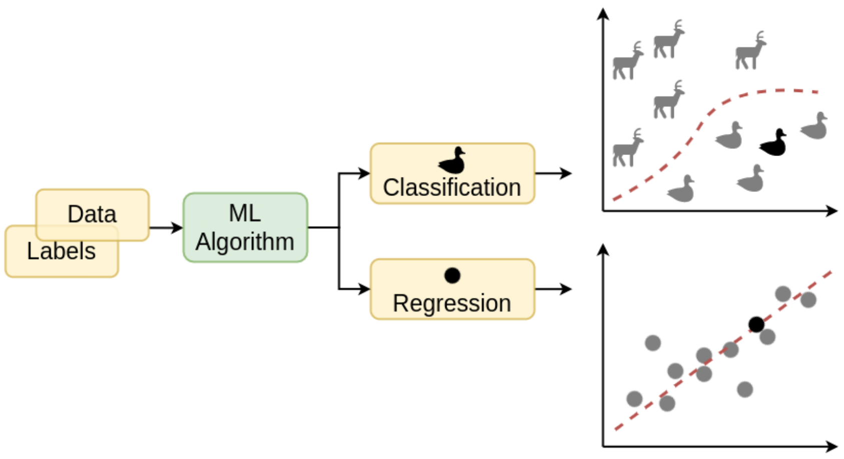
Massively Parallel Reporter Assays (MPRAs) are powerful molecular genetics tools used to screen thousands of sequences for regulatory activity in a single experiment [18]. In this thesis, we used data from an ATAC-STARR-seq experiment conducted in liver cells from Atlantic salmon. Here, we briefly explain the main steps in the ATAC-STARR-seq protocol.

ATAC-STARR-seq is a combination of Assay for Transposase-Accessible Chromatin using sequencing (ATAC-seq), and Self-Transcribing Active Regulatory Region sequencing (STARR-seq). The main idea behind the experiment is to use the hyperactive genetically modified Transposase (Tn5) enzyme, which cuts chromatin where it is loosely packed. Tn5 will cut out fragments of DNA that are accessible, and therefore possibly important for transcription, including the genomic regions that are actively being bound by transcription factors and polymerases. The resulting DNA fragments are enriched for CREs, both promoters and enhancers [19]. The next step in the protocol is to amplify these DNA fragments and clone them into self-transcribing reporter plasmids. Finally, these reporter plasmids can be transfected back into cells, and the resulting RNA produced is then sequenced using high-throughput sequence techniques to quantify the regulatory activity of each fragment. This experiment provides a genome wide quantification of regulatory activity of the accessible DNA with high resolution [14].

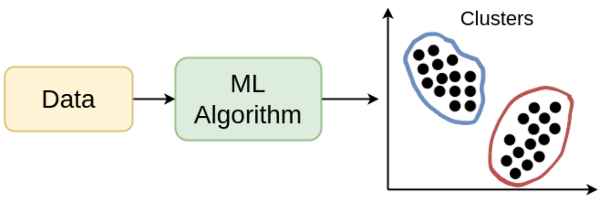
2.2 Machine Learning and Artificial Neural Networks

The goal of machine learning is to create algorithms that are self-taught to recognize patterns in data and use those patterns to make predictions or decisions. Learning algorithms are based on statistical models and trained on large datasets. During the training process, the algorithm iteratively adjusts its internal parameters until it can accurately predict the output for a given input. Once the algorithm has been trained, it can be used to make predictions on new, unseen data.

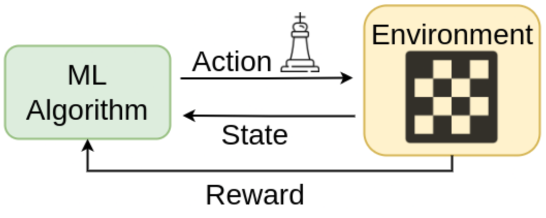
There are different ways a machine learning algorithm can learn, grouped into three major branches: supervised learning, unsupervised learning, and reinforcement learning. Supervised learning algorithms learn to make predictions based on labeled data. The data labels can be either categorical or continuous. In the case of categorical labels, the task becomes a classification problem, whereas with continuous values, it is a regression task. In both cases, the algorithm is trained on a set of input-label pairs, and the goal is to learn the underlying relationship between the input and label values (see Figure 2.5a). Unsupervised learning algorithms are used to find patterns or structures hidden in data that are not labeled. These algorithms try to group data points based on similarity into clusters or reduce the dimensionality of the data by engineering more descriptive features (see Figure 2.5b). Reinforcement learning algorithms learn interactively by trial and error, based on feedback in the form of a reward or penalty signal and the state of the environment. The algorithm adjusts its actions in an environment based on the feedback it receives, with the goal of maximizing the total reward over time (see Figure 2.5c) [20].



(a) Supervised classification and regression illustrated. The Machine Learning algorithm learns the relationship between the input and the target labels. When it is given a new sample, it predicts a label.



(b) Unsupervised clustering illustrated. The machine learning algorithm tries to cluster similar data points together.



(c) Reinforcement learning illustrated. The machine learning algorithm interacts with an environment and learns from feedback.

Figure 2.5: The three basic learning approaches of machine learning (ML) algorithms.

In this thesis, the focus is on supervised learning with labeled data. The task is a regression problem, and we narrow down our theoretical background to cover relevant topics for this specific task. The remaining part of this chapter will revolve around Artificial Neural Networks (ANNs). ANNs are a type of machine learning algorithm that was inspired by biological neurons in the brain [21, 22]. Starting with the perceptron, we will build an essential background of artificial neural networks, and then elaborate on deep neural networks.

2.2.1 The Perceptron

The perceptron is the smallest component of a neural network. It is also called a unit, node or artificial neuron. A sketch of the perceptron and its inner workings is drawn in Figure 2.6. The perceptron processes inputs ($\boldsymbol{x}$) by taking the weighted ($\boldsymbol{w}$) sum of them, adding a bias (b), and employing an activation function (f) to calculate the final output value (y), see Equation 2.1. The activation function is important because it defines the range of possible output values, and transforms the weighted sum into a signal (more on activation functions in Section 2.2.2). This signal can be thought of as a biological neuron that fires. The weights and bias are learnable parameters that must be adjusted to produce the desired output (see Section 2.2.3 on backpropagation).

$$y = f(\boldsymbol{w} \cdot \boldsymbol{x} + b) = f(w_1x_1 + w_2x_2 + \cdots + w_nx_n + b) \tag{2.1}$$

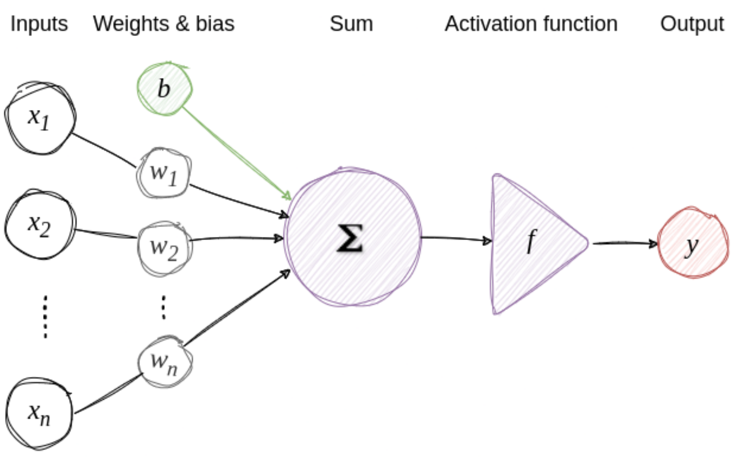


Figure 2.6: Schematic representation of the perceptron. It takes the inputs $(x_1, \ldots, x_n)$ and computes a weighted sum $(w_1, \ldots, w_n)$ and adds a bias (b). Then the activation function (f) computes the output (y).

2.2.2 Multilayer Perceptron

The Multilayer Perceptron (MLP) is a type of artificial neural network that consists of multiple layers, each consisting of multiple perceptrons. In an MLP, the output of each perceptron in a layer serves as the input to the next layer, sending a signal forward, until the final layer produces the network's output. This process is called forward propagation. The layers are fully connected, meaning every node receives the output signal from all nodes in the previous layer. All layers between the input and the output layer are referred to as hidden layers, and more hidden layers in the architecture is a "deeper" model. Figure 2.7 illustrates a simple multilayer perceptron. MLPs are powerful models that can learn complex patterns in data due to their ability to learn non-linear relationships between inputs and outputs, which allows them to model complex functions [23].

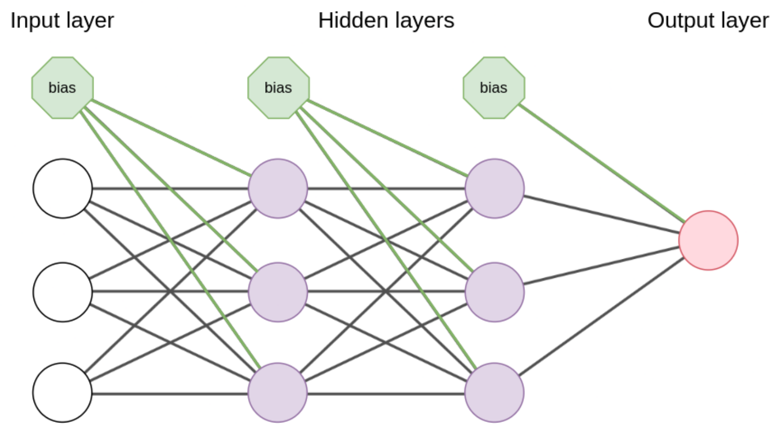
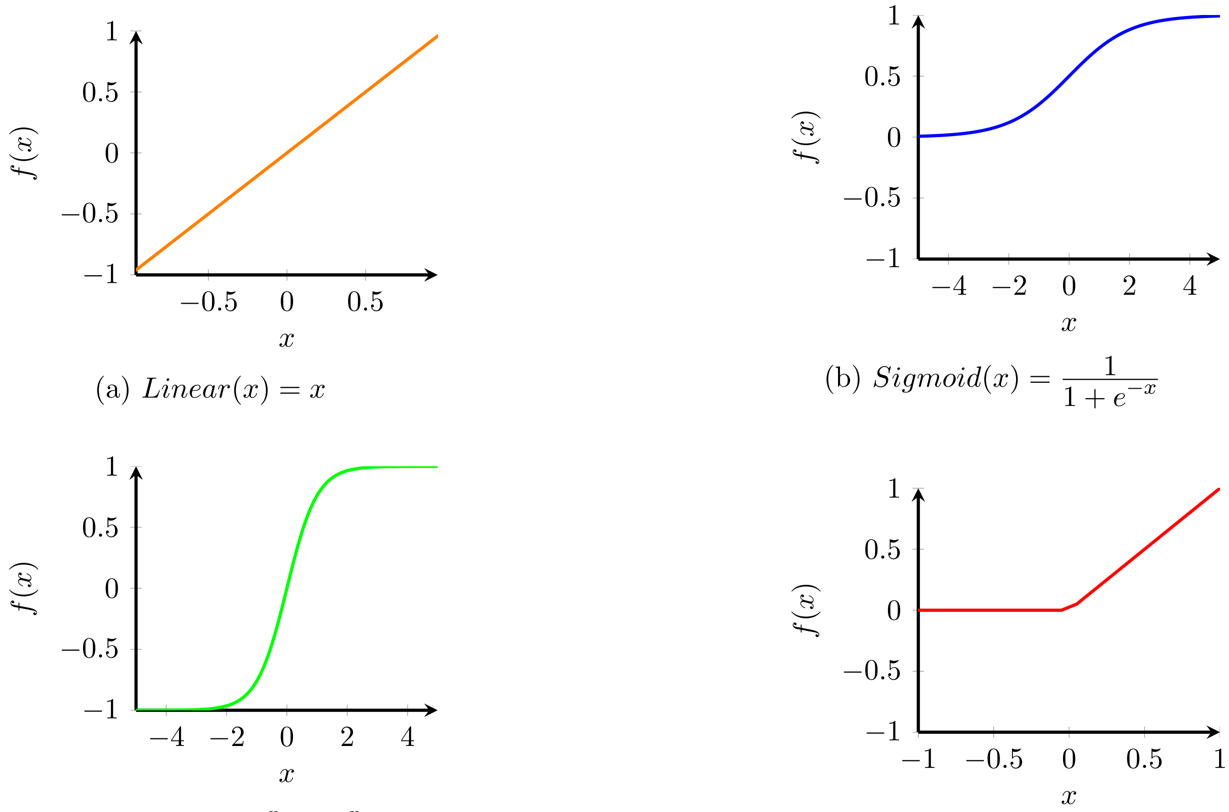


Figure 2.7: Schematic representation of a multilayer perceptron (MLP) network. It consists of an input layer, two hidden layers, and an output layer. The biases are shown in green.

Activation functions

The activation function is applied to the output of each layer during forward propagation, and introduces nonlinearity into the model. The choice of activation function can have a significant impact on the performance of the network. The most used activation functions are the linear, sigmoid, hyperbolic tangent (tanh) and the Rectified Linear Unit (ReLU), we plotted them in Figure 2.8.



(a) $Linear(x) = x$

(b) $Sigmoid(x) = \frac{1}{1+e^{-x}}$

(c) $Tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$

(d) $ReLU(x) = max(0, x)$

Figure 2.8: Activation functions.

2.2.3 Learning

We have so far explained how a simple neural network processes inputs and produces an output through forward propagation. The following paragraphs will go into detail on how a neural network then learns to produce a desired output from labeled data. A flowchart of the learning process of an MLP is shown in Figure 2.9.

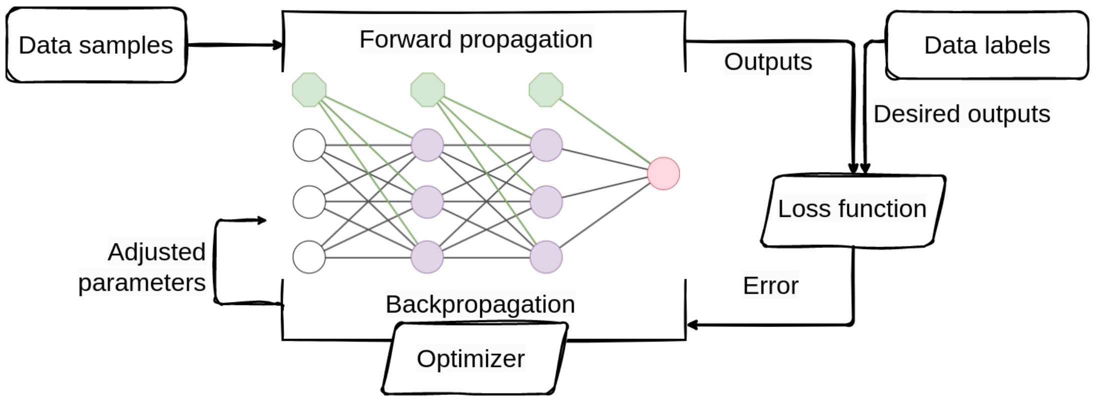


Figure 2.9: Flowchart of the training process of a neural network. The network processes inputs through forward propagation and updates its trainable parameters through backpropagation.

After forward propagation, the learning can begin. A loss function measures the error between the predicted output and the desired output. Then, the trainable parameters are adjusted through backpropagation. This involves computing the gradients of the loss function with respect to the model parameters and employing an optimizer to calculate the updates based on these gradients. Popular optimizers are Stochastic Gradient Descent (SGD), Root Mean Square Propagation (RMSProp), and Adaptive Moment Estimation (Adam). These optimizers are all based on gradient descent, a mathematical algorithm that finds a local minimum by taking steps in the opposite direction of the gradient. The learning rate is a hyperparameter of the optimizer that decides the step size, and can heavily influence the model's ability to minimize its prediction errors [20].

Backpropagation

Backpropagation is an efficient way to calculate how much each of the model's parameters needs to change to reduce the error of the model's predictions. Using the chain rule of calculus, it is possible to compute the derivative of the loss function with respect to the output from each layer, and then propagate these derivatives backwards through the network [20]. The network is a nested function where each layer takes the previous layer's output as input. Equation 2.2 illustrates how the derivative of a nested function is computed using the chain rule for a network consisting of five layers (f, g, h, u, v). Modern deep learning software implements the backpropagation algorithm extremely efficiently, such that it is possible to train complex deep neural networks with a rich selection of different configurations, hidden layers and activation functions.

$$\frac{df}{dx} = \frac{d}{dx}(f(g(h(u(v(x)))))) = \frac{df}{dg} \cdot \frac{dg}{dh} \cdot \frac{dh}{du} \cdot \frac{du}{dv} \cdot \frac{dv}{dx} \tag{2.2}$$

Training, testing and validating a model

A key concept in machine learning is to use different data samples to train, test and validate a model, as demonstrated in Figure 2.10. This is an important practice to prevent information leakage into the model, such that it seems like it is performing well, when in reality the model is simply remembering the training data. To ensure that the model is generalizing and learning from the training data, it is therefore vital to hold out unseen samples to validate the model post training. These validation samples would optimally be a representative selection of the training data, which again would be representative of the real world.

A common practice is to randomly assign fractions of the samples for training, testing, and validation. For DNA sequences, this often results in an unrepresentative selection due to overlapping fragments. To measure how well a model generalizes to unseen genomic loci, it is therefore preferred to hold out entire chromosomes for validation. If the purpose of the model is to be used as a predictor of unseen cell types, it is wise to go even further and train and validate the model across both chromosome and cell type [24]. In both cases, we assume that the patterns in the training samples are representative of the held out chromosomes.

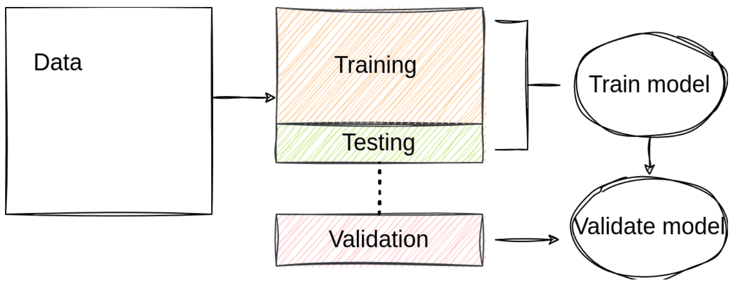


Figure 2.10: Splitting data into sets for training, testing, and validation is important in machine learning. The testing and validation samples can be randomly selected or, for genomic data, from specific chromosomes.

Batches and epochs

During training, a neural network processes multiple inputs, a batch, before updating its parameters. The batch size must therefore be a subset of the total number of samples in the training data. An epoch is when the network has processed enough batches to complete all training samples once. The batches are randomized such that they are different every epoch. It is useful to run the model for multiple epochs, reiterating over the data to keep updating the model parameters and improving its performance.

Overfitting

When fitting a model to training data, we want the general patterns to be found. In Figure 2.11, we illustrate how an underfitted model does not follow the data closely enough, and an overfitted model follows the training data too closely. Overfitting occurs when the model memorizes the training data, instead of generalizing and learning how to predict it. The symptom of overfitting is that the model performs much better on the training data, than on the testing and validation data. This can happen if the model has been training for too many epochs and has too large capacity, i.e. too many parameters. It is a fundamental problem in machine learning, since the model essentially becomes a perfect predictor of the training data at the expense of weakening its ability to predict on unseen samples [25]. Therefore, there are plenty of strategies to mitigate overfitting, like early stopping, dropout and batch normalization.

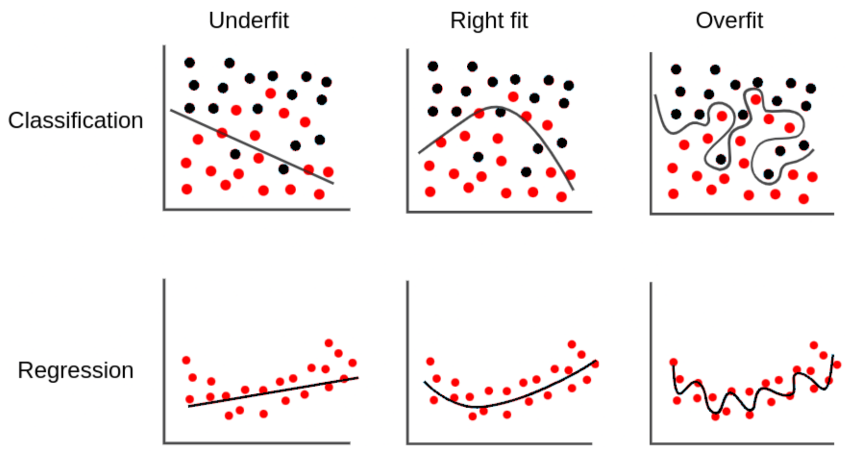


Figure 2.11: Illustration of the three model fitting situations for classification and regression.

Early stopping

A simple way to prevent overfitting is to monitor the testing performance after each epoch and finish training if the test error stops decreasing or starts increasing. This is called early stopping, and its "patience" is the pre-defined number of epochs we wish to wait for further improvement before termination of the training. Early stopping provides a way to restore the model parameters at their best point during training, while reducing the runtime [25].

Dropout

Dropout is another simple and effective mechanism that forces a model to generalize. Dropout is a predefined fraction of the nodes in a layer whose outputs are multiplied by zero. Essentially, a fraction of the nodes are randomly left out, or turned off. This makes it easier for the model not to learn redundant noise in the data, and forces it to generalize. When different subsets of the hidden nodes are turned off, one is effectively training lots of smaller and thinned out networks, that share parameters [26]. This has a great effect of preventing overfitting. During testing and validation, dropout is not applied on any nodes, and the model as a whole predicts the output.

Batch normalization

Batch normalization is applied both to reduce overfitting, and also accelerate the training process. To normalize the inputs of a layer, the mean and standard deviation of inputs over a batch of training samples needs first to be calculated, and then these values are applied to normalize the inputs. Since the distribution of the inputs to the layer is stable at zero mean and unit standard deviation, the network can handle greater learning rates and learn faster, due to more stable gradients [27]. A batch normalization layer also keeps track of a moving average and a moving standard deviation, which is needed after the model is done training, such that it can make predictions on a single sample [25].

2.3 Deep Neural Networks

In Section 2.2.2, a simple fully connected neural network, the MLP, was described. In deep learning, multiple hidden layers are applied to increase the capacity and complexity of the model. Deeper models have the capacity to transform inputs into higher order feature representations, and therefore handle raw data of high dimensionality better than classical machine learning models, so-called “shallow” networks. The process of extracting and engineering features from raw data is left for the deep model to learn. There are different types of deep neural networks, the relevant ones for this thesis will be presented in the next sections 2.4-2.5. Finally, we also delve into important concepts like evaluation, tuning, and interpretation of deep models.

2.4 Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a different kind of network that uses shared parameters and spatial configurations to process inputs. In a CNN, filters (also called kernels) scan the input data for local patterns (see Figure 2.12). The filter size is much smaller than the input data, such that it can learn small components and simple patterns. The convolution is essentially a dot-product operation between the input and the filter at each position. The consideration of spatial context is what makes the CNN well-designed for data where the arrangement of the data points matter, such that randomly shuffling them would cause a great loss of information, like sentences, images, and DNA sequences. The filters can learn to search for phrases in a sentence, edges in an image, or motifs in a DNA sequence. One filter of a single convolutional layer scans the entire input like a sliding window, with the same shared parameters for every location. The filter can therefore detect the presence of a pattern, regardless of where it occurs. This sharing of weights reduces the need for numerous parameters, since it is not required to learn them for every possible location of the pattern. The output of a filter in a convolutional layer is referred to as a feature map, since it maps out where a pattern was discovered. CNNs are popular and especially successful in image recognition and object detection tasks [28].

Stride

The stride is a hyperparameter of the convolutional layer that determines the amount by which the filter shifts over the input at each step. A stride of 1 means that the filter shifts one position at a time, while a stride of 2 means that the filter shifts two positions at a time, and so on. Using larger strides reduces the size of the feature map and can therefore speed up computations. However, it can also reduce the amount of information that is preserved in the feature map, in the worst-case losing valuable information such that the overall performance of the model is affected [28].

Padding

Another important hyperparameter of the convolutional layer is the padding. The padding is the number of data points with a value of 0 to add as an extra border around the input, as illustrated in Figure 2.12. The reason for doing this is to increase the output size of the feature map. Without padding, the output size would always shrink dependent on the filter size, where a larger filter shrinks the output by a larger factor. This would limit the number of possible convolutional layers one could apply after one another, since the feature map would eventually reach a size of only one value. It also impacts the choice of filter sizes, since big filters would shrink the feature map further. Padding the input allows for more freedom of the architecture and filter size. Including numerical padding settings, two other padding options are “valid” and “same”. Where valid padding simply means no padding, same padding means adding the amount of padding needed to produce an output feature map of the same size as the input [28].

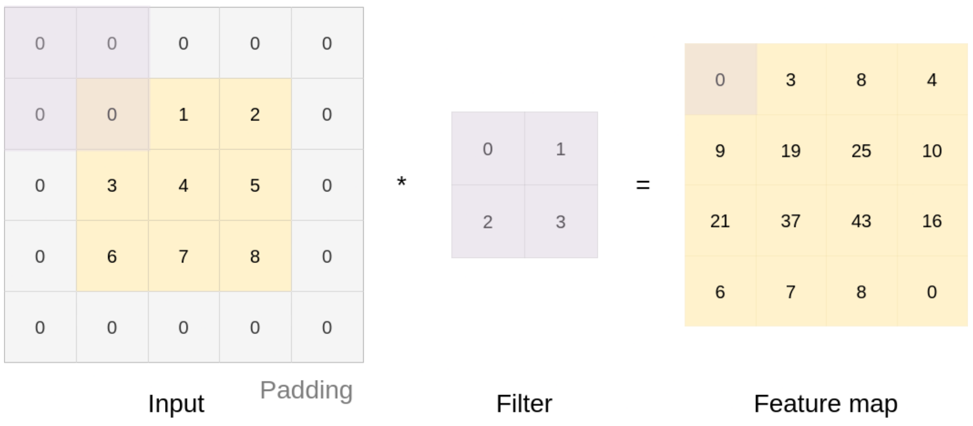


Figure 2.12: A convolution involves applying the same filter weights to multiple positions over the input data to produce an output feature map. In this example, a border of padding is added to the input data (padding = 1). The output values are the dot products of the input and the filter at each position, here with a stride of one. Figure inspired by Zhang et al. [29].

2.4.1 Pooling

Applying a pooling operation after a convolution is a common way to process feature maps and make them more robust. Pooling is compressing the convolution feature map into a smaller output by downsampling from neighborhoods of data points (see Figure 2.13). The pool size is the size of this local area. Average pooling is computing the average of the activations over small regions of the feature map, while max pooling is finding the maximum activations. The output therefore describes approximately where these activations occurred in the input using a smaller feature map, and each value corresponds to a larger area of the input. Pooling operations are useful for plenty of reasons: They retain the most important information from the feature maps, increase the receptive field further into the model, remove noise, reduce the chances of overfitting, and reduce the number of parameters in the model [28].

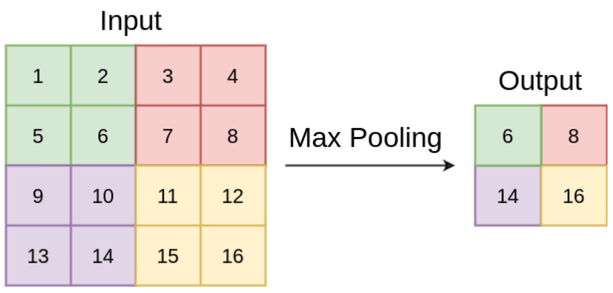


Figure 2.13: Max pooling illustrated. In this example, the maximum value is extracted from a local area with pool size 2 by 2, and the region moves with a stride of 2 positions, which effectively reduces the input feature map by a factor of 2.

2.5 Recurrent Neural Networks

A Recurrent Neural Network (RNN) is a type of artificial neural network that is designed to work with sequential data, where the order of inputs is of significance, such as time series, videos, texts, and sequences. Unlike traditional feedforward neural networks, which process data one input at a time in one direction, RNNs retain information from previously processed inputs by maintaining a memory. Practically, RNNs are stacked perceptrons that use loops in their architecture, as shown in Figure 2.14. In a simple RNN architecture, each neuron has a feedback loop that allows it to receive information from the previous sample, process it along with the current input sample, and then send the output both forward to the next layer and to the next sample step. This allows RNNs to process sequential data and capture temporal dependencies between inputs. A Bidirectional Recurrent Neural Network (BRNN) is essentially two RNNs employed together in the same layer, where one processes inputs in the forward direction, and the other processes them in the backward direction. BRNNs are useful when applied to genomic data, since information can be found both upstream and downstream along a sequence.

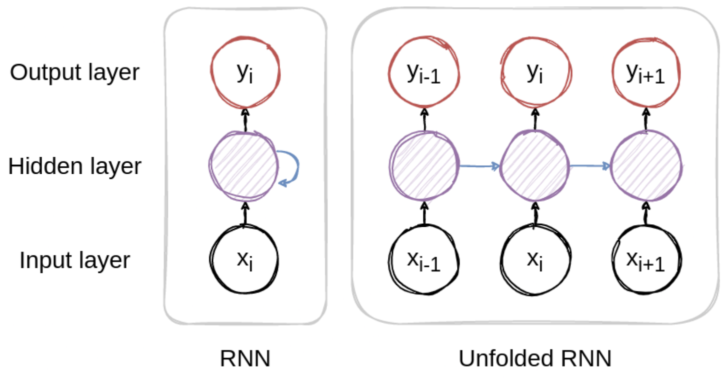


Figure 2.14: Sketch of a simple Recurrent Neural Network with a single node. The node processes an input x_i, and produces an output value y_i that is also sent to the node itself for subsequent inputs (blue arrow). When unfolded, the network resembles stacked perceptrons.

However, one issue with simple RNNs is that they can suffer from the vanishing gradient problem, where the gradients used for learning become increasingly smaller when propagated backwards. The influence of earlier data points will decrease as more inputs are processed, making it difficult for the network to retain information for many steps, a long-term memory. To address this issue, more advanced processing units have been developed, such as Long Short-Term Memory (LSTM) cells and Gated Recurrent Units (GRUs), which are designed to better capture these long-term dependencies.

2.5.1 Long Short-Term Memory

Long Short-Term Memory cells are designed to handle the vanishing gradient problem that occurs in standard RNNs. The key feature of an LSTM is its ability to choose what information to retain from a sample and keep this information in its cell state. The LSTM cell has three gates: the forget gate, the input gate, and the output gate, which control the flow of information into and out of the cell (see Figure 2.15). The input gate regulates what new information is relevant and kept in the cell, while the forget gate controls what information should be discarded from the cell. The output gate then processes both the input and the cell state to produce the final output value.

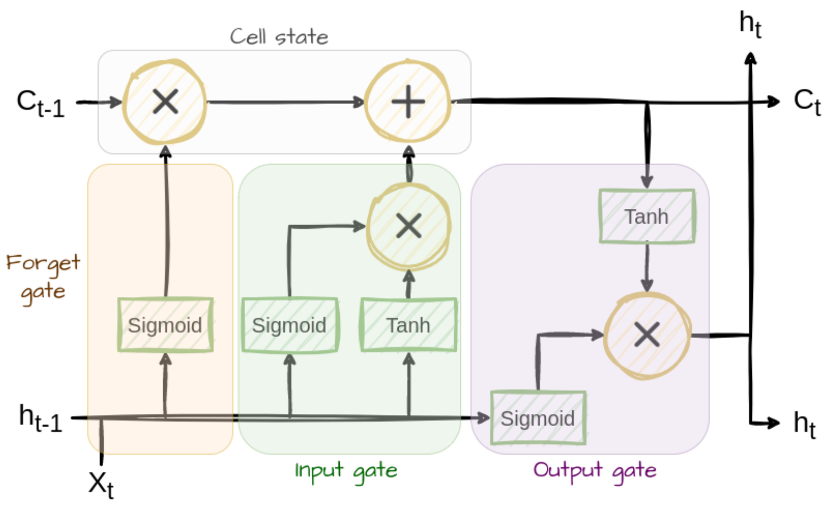


Figure 2.15: An LSTM cell maintains a selective memory (cell state) by processing inputs through a forget gate, input gate, and an output gate. Yellow circles are element wise operations. Green boxes are activation functions. Values are simply concatenated or copied where arrows join and split, respectively. Outputs h_t are passed to both the next layer and the next step at input x_{t+1}.

2.6 Evaluation metrics

To assess the performance of a deep neural network, evaluation metrics are needed. The choice of evaluation metric depends on the problem. For example, if the problem is classification, one can measure accuracy, precision, and recall of the model. If it is a regression problem, one can measure Mean Squared Error, Mean Absolute Error, and correlation. Here, the definitions of some relevant metrics for this thesis are presented.

2.6.1 Mean Absolute Error

The Mean Absolute Error is a common metric to measure errors of a regression model. Specifically, it measures the average absolute difference between the predicted and actual values. The calculation is shown in Equation 2.3, where n is the number of samples in the dataset, x_i is the predicted value of sample i, and y_i is the observed value of sample i.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|x_i - y_i| \tag{2.3}$$

The MAE metric is always positive, and the value is on the same number scale as the target values. This means that the MAE number must be interpreted in light of the values that are predicted. An example is if a model tries to predict tomorrow's weather in degrees Celsius and has an MAE of 20, it means the predictions are on average 20 degrees Celsius off, which to many would be considered a bad weather forecaster. On the other hand, if the model is predicting the monthly income of a local store in dollars, an MAE of 20 can be highly accurate. The MAE is thus a metric that is not intuitive on its own, due to this problem specificity [30].

2.6.2 Mean Squared Error

Mean Squared Error (MSE) measures the average squared difference between the predicted and observed values. The formula is presented in Equation 2.4 where x_i is the predicted value of sample i, y_i is the observed value of sample i, and n is the total number of samples in the validation set.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2 \tag{2.4}$$

The MSE is a useful metric for evaluating regression models because the squaring operation causes larger errors to be penalized more than smaller errors. However, it can therefore also be sensitive to outliers. In general, a lower MSE value indicates better performance of the model. The interpretation depends on the specific problem and the scale of the target variable, and can be unintuitive since it is the square of a measurement [20].

2.6.3 Pearson Correlation Coefficient

Equation 2.5 defines the Pearson Correlation Coefficient, where n is the number of observations in the dataset, x_i and y_i are the predicted and observed values of sample i, and $\bar{x}$ and $\bar{y}$ are the means of the two variables.

$$PCC = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{2.5}$$

The Pearson correlation coefficient ranges from -1 to +1, where a value of +1 indicates a perfect positive correlation, a value of -1 indicates a perfect negative correlation, and a value of 0 indicates no correlation between the variables. The correlation is always on the same scale, and thus more intuitive [20].

2.7 Hyperparameter Tuning

The performance of models is limited by the architecture and configurations of the model itself. These model configurations are called hyperparameters, to avoid confusion with the trainable parameters within the model. Essentially, they are settings decided prior to training, for example the number of nodes in a layer, or even the number of layers in total. Finding an optimal combination of hyperparameters can improve the performance of the model significantly, and this process is termed hyperparameter tuning. The range and set of values make up the search space, and is defined prior to tuning. It is exponentially time-consuming to test additional hyperparameter values due to the increased amount of hyperparameter combinations. Educated guesses and some trial and error are needed to narrow down the search space, but tools are in development to assist and automate this process [31].

2.8 Deep neural network interpretation

Interpreting deep neural networks is an active area of research. It can be a challenging task, as these models are highly complex and often have thousands of parameters. Deep models are good at understanding underlying patterns in real-world data, but they hide their insight behind complex computations. They are, in this regard, "black boxes". In research applications, having a good predictor is not the only goal of training a model. It can be more valuable to get a hold of the knowledge that the model learned.

There are several techniques that can be used to gain insight into how a model is making its predictions. A common approach is to analyze the activations of different neurons in the network. This involves identifying which neurons are firing most strongly in response to particular

inputs, and then examining the patterns of activation [32]. This can provide insight into which features or combinations of features are most important for the model's predictions [33].
With CNNs, it can be valuable to visualize the individual filter activations from convolution layers. This can reveal what patterns from the input data are good predictors. In the field of image processing, many tools have been made to visualize what patterns, textures, edges, or shapes the model is focusing on [34]. The same can be done for CNNs trained on DNA sequences, where the filters of convolutional layers reveal DNA motifs [35].

CHAPTER 3

MATERIALS AND METHODS

To answer the research questions from Chapter 1, we obtained a dataset of DNA sequences including their observed gene expression values. Two deep neural networks appropriate for sequential data were built, and two model interpretation methods were applied to the trained models. In this chapter, we present how DNA sequences from open chromatin were collected, and the regulatory activity was quantified for each sequence. Furthermore, we detail the approach to analyze the data using the deep neural networks.

3.1 ATAC-STARR-seq

The sequence data used in this thesis is a result of an ATAC-STARR-seq experiment conducted by Sandve et al. (unpublished results) at the Norwegian University of Life Sciences (NMBU). The experiment largely followed the method described by Wang et al. [14], with the exception that the mean fragment length was shorter. In brief, a modified Transposase Tn5 enzyme was used to induce double-stranded DNA breaks in accessible chromatin in primary liver cells from Atlantic salmon (*Salmo salar* L.). Next, very long fragments were discarded from the pool of resulting DNA fragments, to ensure only fragments originating from the open chromatin remained. Note that the Tn5 enzyme also appended adapters onto each end of the fragments when cutting, which provided a target for sequencing later on in the process. The next step was to create a library of circular DNA vectors, each including one of the DNA fragments from open chromatin in liver cells. These random fragments were cloned into reporter vectors at the 3' end of a truncated GFP reporter gene with a basal promoter driving transcription. This vector construct would then self-transcribe the cloned fragment upon transcription in the cell, enabling identification and quantification of the number of RNA transcripts with different fragment types.

After an amplification step in E. coli, these vectors were then inserted back into liver cells, ensuring a relevant cell environment, using transfection by electroporation. After approximately 48 hours, RNA was isolated and sent for mRNA sequencing using Illumina paired-end sequencing, targeting the adapters added by the Tn5 enzyme in the initial ATAC reaction.

Finally, the regulatory activity of each fragment was quantified by comparing the count of RNA with specific ATAC-fragments to the original fragment count following E. coli amplification. We calculated the ratio of RNA to DNA to account for variations in copy number of the unique fragments in the original vector library. The final dataset contained the raw DNA sequence fragments, as well as statistics from a differential expression analysis using the R package DESeq2 [36]. These include the mean of normalized counts for all samples (base mean) and the log2 fold change. The log2 fold change measured the difference in RNA count, where a value of 1 means a doubling, 2 a quadrupling, and a value of -1 indicates a halving.

3.2 Data Preprocessing

Deep networks have the ability to learn higher order data representation. This advantage omits the task of feature extraction and feature engineering. Still, preprocessing of the raw sequences is needed for the network to read the raw data. To avoid information leakage, it is also essential to split the dataset into training, testing, and validation sets. In this section we present how the raw data was filtered, one-hot encoded, padded and split before modeling.

3.2.1 Raw data

The raw data contained 5 832 889 sequences varying in lengths, as shown in Figure 3.1, and base mean, as shown in Figure 3.2. The distribution of observed log2 fold change for all sequences is shown in Figure 3.3. Five sequences contained the ambiguous symbol "N" and these were discarded.

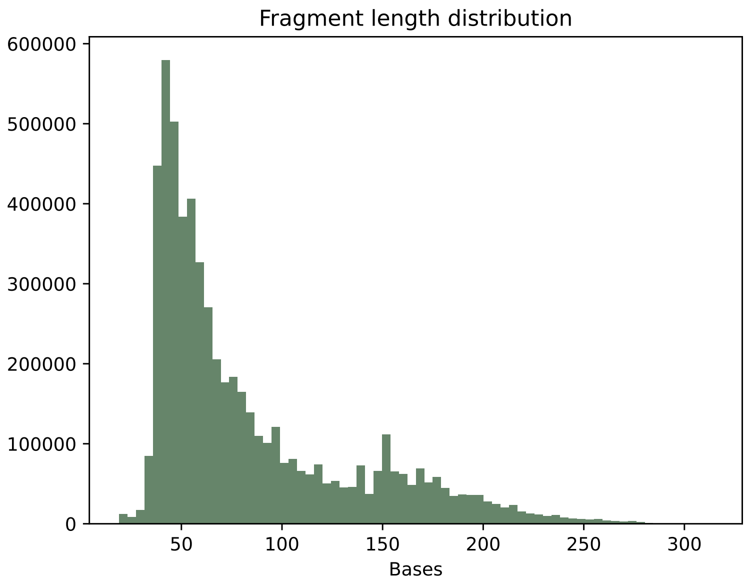


Figure 3.1: Length distribution of all the approximately 6 million DNA fragments.

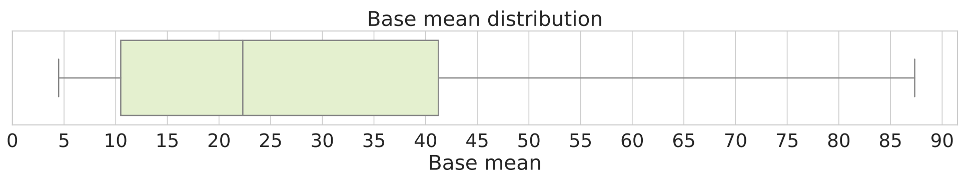


Figure 3.2: Base mean distribution of all samples in the data. Outliers are excluded.

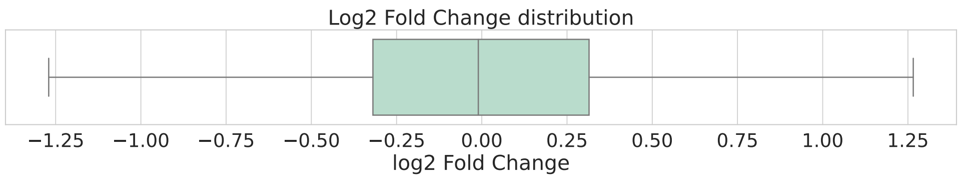


Figure 3.3: Log2 Fold Change distribution of all samples in the data. Outliers are excluded.

3.2.2 Data Sets

From the raw data, four different data sets were created by filtering out sequences based on different base mean thresholds. Table 3.1 gives a detailed description of the data sets, and they are visualized in Figure 3.4. Each set was later split for training purposes (see Section 3.2.4).

Table 3.1: Detailed description of the four data sets created from the raw data.

Name	Base mean threshold	Fraction of samples	Number of samples
10bm	> 62	10%	583 287
25bm	> 41	25%	1 458 219
50bm	> 22	50%	2 916 439
100bm	None	100%	5 832 879

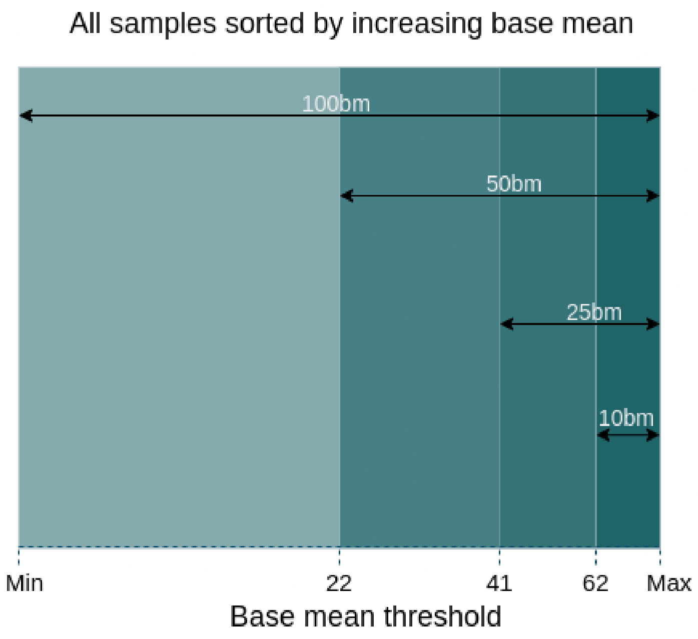


Figure 3.4: A visualization of how the data was filtered by base mean into four data sets. A darker color and shorter arrow represents the fraction of the total number of samples in the data, and the names of each set are written in white. The specific number of samples in each data set can be found in Table 3.1.

3.2.3 One-Hot encoding and zero padding

The sequences were one-hot encoded such that each nucleotide A, C, G and T were represented by a vector, see Table 3.2. Each sequence was therefore converted to a matrix with four columns, as illustrated in Figure 3.5. A maximum length of 200 was chosen, and longer sequences were trimmed by discarding nucleotides at the beginning of the sequence. Shorter sequences were padded with vectors of only zeros at the end, to fill the sequence representation matrix such that all input samples had size (200, 4).

Table 3.2: Nucleotides and corresponding vector representations.

Nucleotide	Vector
A	[1, 0, 0, 0]
C	[0, 1, 0, 0]
G	[0, 0, 1, 0]
T	[0, 0, 0, 1]

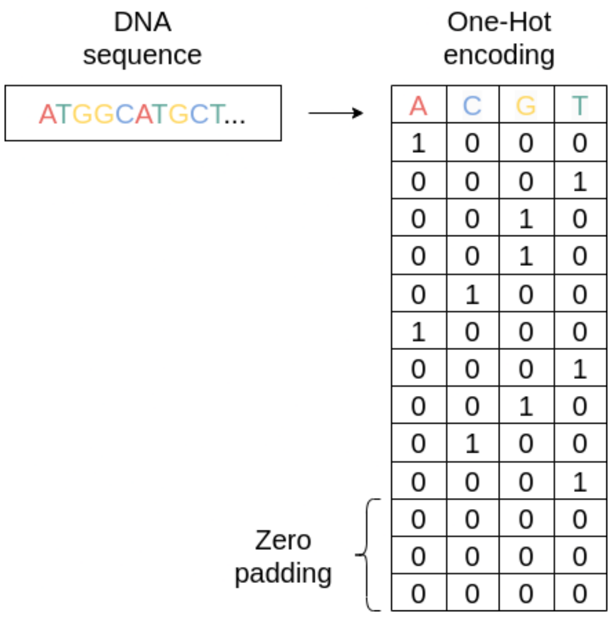


Figure 3.5: One-Hot encoding and zero padding used to represent a DNA sequence as a matrix with dimensions (max-length, 4).

3.2.4 Training, testing, and validation split

The data was split into three data sets for training, testing, and validation purposes. Chromosome 21 was held out for testing, and chromosome 25 was held out for validation. Table 3.3 shows the resulting number of samples in each split, for each of the four data sets.

Table 3.3: Detailed description of the four data sets created from the raw data. The testing and validation samples are from chromosome 21 and 25, respectively.

Name	Number of training samples	Number of testing samples	Number of validation samples
10bm	559 844	12 256	11 187
25bm	1 397 939	31 397	28 883
50bm	2 798 974	61 127	56 338
100bm	5 625 910	106 057	100 912

3.3 Convolutional Neural Network

General theory of Convolutional Neural Networks is explained in Chapter 2, Section 2.4. This section describes how the CNN architecture was built, implemented and tuned.

3.3.1 Architecture

The CNN architecture was heavily inspired by the DeepSTARR model proposed by Almeida et al. [1]. We modified the DeepSTARR model by removing the multitask output mechanism such that the model could predict only one output value, see Figure 3.6 for a visualization. To enable motif extraction, we also modified the first convolutional layer to have valid padding. The network takes in a one-hot encoded sequence, and predicts one log2 fold change value. The hidden layers consist of four convolution "blocks" followed by two fully connected layers. Each convolution block is constructed as follows: A one-dimensional convolution layer, batch normalization, ReLU activation, and a one-dimensional maxpooling layer. After the first layer, all the following convolutional layers have "same" padding. All maxpooling layers have pool size 2. A flatten layer links the convolutional blocks with the fully connected part of the network. The two fully connected layers are both followed by batch normalization, ReLU activation, and have a dropout probability of 40%. The output layer has one output node with linear activation, and thus the prediction range of log2 Fold Change values is unrestricted.

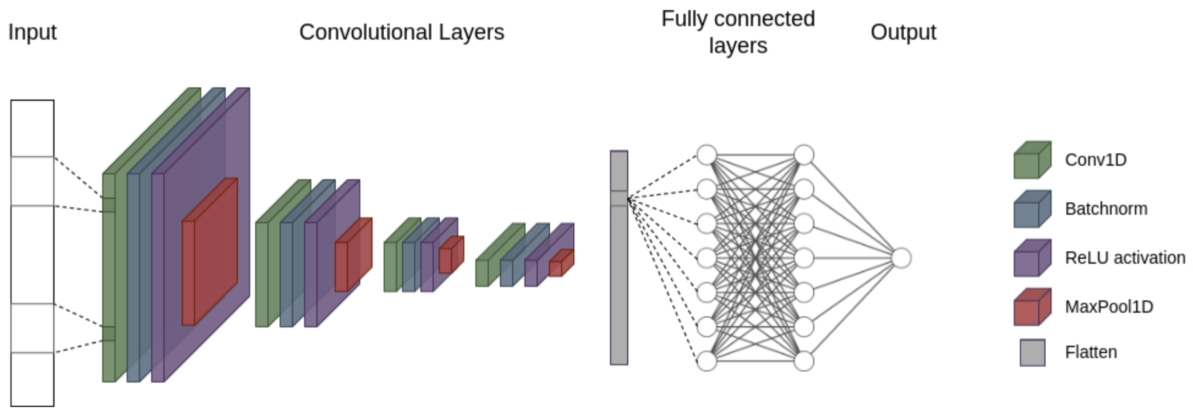


Figure 3.6: Visualization of the Convolutional Neural Network architecture.

The CNN was compiled with the Adam optimizer, Mean Squared Error loss function, and early stopping monitoring the validation loss with patience of 10 epochs.

3.3.2 Tuning

In an effort to improve the performance of the CNN, we used GridSearch Tuner [37] to search for the best model configuration among several relevant hyperparameters. The models were trained on the 10bm data set, and the objective was to find the model with minimal Mean Absolute Error on the validation samples. An initial search tested different numbers of convolution blocks, ranging from 4 to 7. The search revealed that 4 convolution blocks were sufficient. Follow-up searches each tested different model configurations and compared CNNs with varying number of nodes, filters, and kernel sizes, as detailed in Table 3.4. All possible combinations of the hyperparameters were not tested due to time constraints. We applied some strategies to reduce the search space and runtime, like keeping some hyperparameters identical. This way, we were able to test a wider variety of model configurations. The best model had 468 105 trainable parameters, and its hyperparameters are highlighted in Table 3.4.

Table 3.4: Convolutional Neural Network hyperparameter search space. Layer$_n$ clarifies the type and order of the hidden layers, and the set of tested values are shown in brackets. The best hyperparameters are highlighted.

Layer$_n$	Filters or nodes	Kernel sizes
Conv1D$_1$	[**224**, 256]	[7, **9**]
Conv1D$_2$	[30, **60**]	[5, **7**]
Conv1D$_3$	[30, **60**]	[3, **5**]
Conv1D$_4$	[60, **80**, 120]	[5, **7**]
Dense$_5$	[128, **256**]	-
Dense$_6$	[128, **256**]	-

3.4 Hybrid Convolutional and Long Short-Term Memory Neural Network

The Hybrid CNN-LSTM architecture was inspired by the DanQ model presented by Quang and Xie [13]. General theory of Recurrent Neural Networks is described in Chapter 2, Section 2.5. This section describes how the hybrid model architecture was built, implemented and tuned.

3.4.1 Architecture

The hybrid model consists of three parts: a convolutional part, a recurrent part and a fully connected part at the end, as visualized in Figure 3.7. The first part has only one convolutional layer with ReLU activation, and we modified it to match exactly the first layer of the CNN, such that motif extraction would be identical. Maxpooling with pool size 2, and dropout with 30% probability is added after the convolution. Next follows a bidirectional recurrent layer with equal number of LSTM units in both directions, and another dropout of 30% probability. The final hidden layer is a dense layer, fully connected to the output layer, that has one output node with linear activation. The hybrid model was compiled in the same way as the CNN, using Adam optimizer, Mean Squared Error loss, and early stopping.

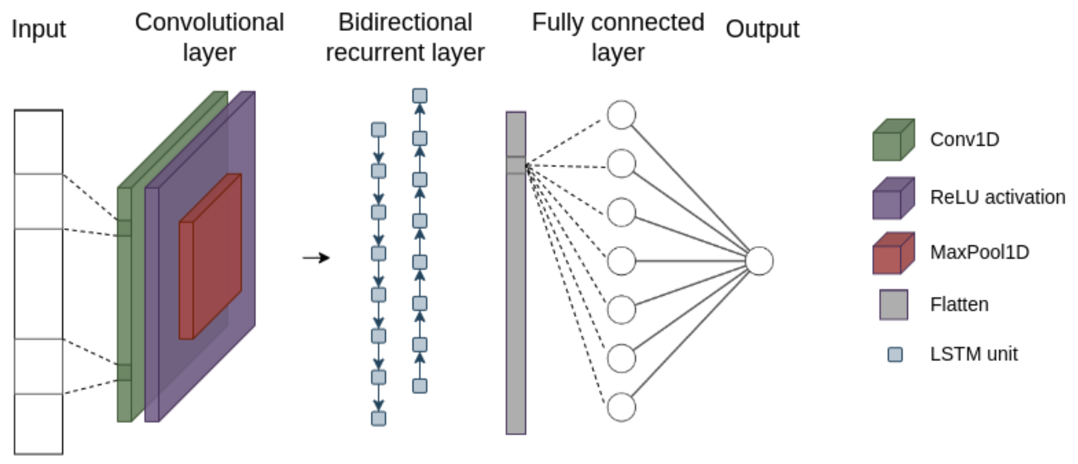


Figure 3.7: Visualization of the Hybrid Convolutional and Long Short-Term Memory Neural Network architecture.

3.4.2 Tuning

We tuned the hybrid model with GridSearch Tuner [37] exploring the hyperparameters in Table 3.5. All combinations of hyperparameters were not tested, but all values were covered through multiple tuning tests with overlapping smaller search spaces. The best hyperparameters discovered are highlighted in Table 3.5. Initial tuner searches quickly revealed that the default DanQ [13] model was too complex. The final best architecture resulted in a model with only 130 319 trainable parameters, compared to the approximately 10 million parameters of DanQ.

Table 3.5: Hybrid Convolutional and Long Short-Term Memory Neural Network hyperparameter search space. Layer$_n$ clarifies the type and order of the hidden layers, and the set of tested values are all shown in brackets. The best hyperparameters are highlighted.

Hyperparameter	Values
Nodes in Bidirectional LSTM$_2$	[10, 15, **25**, 50, 100]
Nodes in Dense$_3$	[10, **15**, 25, 50, 100]
Learning rate	[0.0001, **0.001**, 0.002, 0.1]

3.5 Motif extraction from first convolution layer

We extracted motifs from the first convolutional layer of all the trained models. The method and code was obtained from Quang and Xie [38]. A position frequency matrix was created for each of the 224 filters in the convolutional layer, by passing batches of 100 sequences from the 10bm validation samples through this layer. The layer had a kernel size 9, meaning it scanned the sequence with a window of 9 nucleotides. The subsequences at which the maximum positive activation occurred were recorded into a position frequency matrix. This was later converted to a position weight matrix by dividing each nucleotide frequency by the sum of all frequencies at that position. The result was 224 DNA motifs, all 9 in width, for each of the 8 models that were trained.

3.6 Motif extraction with deepExplainer and TF-MoDISco

To further interpret the deep models, we also used deepExplainer (Deep SHAP implementation of DeepLIFT [39]) on one CNN and one hybrid model, the ones that were trained on the 10bm data. DeepExplainer calculated importance scores along the sequences from the 10bm validation samples, using Shapley additive explanations (SHAP) [33] for every nucleotide in the sequence. The scores indicate how much each position contributes to the output prediction. We then applied TF-MoDISco (Transcription-Factor Motif Discovery from Importance Scores [40]) to find repeated patterns and create motifs. We filtered out sequences that were shorter than 100 in length. Following the method of Almeida et al. [1], we also used 100 dinucleotide-shuffled versions of each input sequence as reference sequences, and instructed TF-MoDISco to create motifs of length 16.

3.7 Motif matching to JASPAR with Tomtom

The Tomtom algorithm by Gupta et al. [41] was used to compare the extracted motifs with known motifs in the JASPAR CORE Vertebrates database. We used the MEME Suite version 5.5.1 [42] to run Tomtom with default settings. All Tomtom matches with a q-value < 0.05 were regarded significant.

3.8 Reproducibility

In an effort to make this master's thesis project reproducible, we made the source code publicly available on GitHub, see Table B.1 in Appendix B for permanent links and commit hashes. TensorFlow seeds were used to set the sequence of pseudo-random number generators at the beginning of the deep learning scripts. However, since the code was run on a GPU, reproducibility cannot be guaranteed due to parallelization, optimization and nondeterminism in software compiled for GPU usage (e.g., TensorFlow, cuDNN) [43, 44]. Therefore, we also publish all trained models, see Table B.2 in Appendix B. Everything is available at the GitHub repository *github.com/juforris/datasci-thesis23*. See Appendix A for software (Table A.1) and hardware (Table A.2) specifications.

# CHAPTER 4 RESULTS

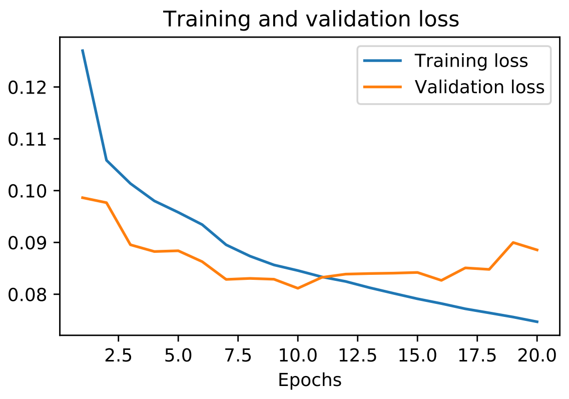
In Chapter 3, we defined four data subsets referred to as *10bm*, *25bm*, *50bm*, and *100bm* (see Table 3.1). We also presented two model architectures—the CNN and the hybrid CNN-LSTM. The datasets were used to test the effect of noisy data on the models' ability to predict gene expression levels from sequence. This chapter presents and discusses performance results and motif findings from the two deep neural networks. These results aid in answering the research questions formulated in Chapter 1, Section 1.2. We discuss our findings as a whole in Chapter 5.

4.1 Convolutional Neural Network

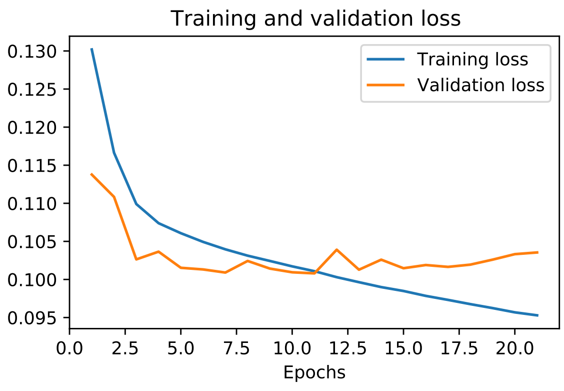
The CNN was based on DeepSTARR [1], and tuned on our 10bm dataset. The tuning only slightly improved its performance, possibly because DeepSTARR already is a well-tuned model on similar data (UMI-STARR-seq), which is why our hyperparameters closely resemble the DeepSTARR's. We wanted to investigate the effect of noisy data on the model predictions, and trained the CNN on samples with different ranges of base means. We then cross tested the CNN by training and validating it on all combinations of the four datasets. The performance results are presented in Section 4.1.1. After training, we extracted motifs to investigate what the model had learned, and we report the findings in Section 4.1.2.

4.1.1 Performance

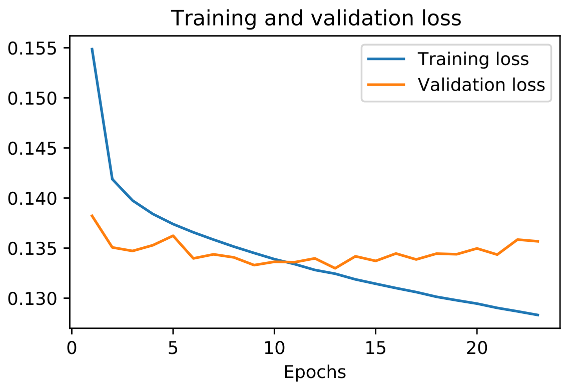
In Figure 4.1 we have plotted the learning curves of the CNN when trained on different subsets of the data. The plots show how training loss and validation loss changed over time when training the CNN for multiple epochs. The validation loss is the MSE of the predictions of test samples (chromosome 21). During the first 10 epochs in all the plots, the validation loss was lower than the training loss, which means that the model had a lower error when predicting on the test samples than the training samples. Then, during the following epochs, the model learned to predict the training samples better (training loss decreased), but in Figure 4.1a, 4.1b and 4.1c the validation loss started to increase and become greater than the training loss. In Figure 4.1d, the validation loss did not exceed the training loss, but it stopped improving after only 4 epochs. The improvements were minimal—validation loss stayed between 0.19 and 0.185. We can clearly see that the more noisy data results in a greater loss overall from the different ranges on the y-axis' of all the four plots. The CNN trained for approximately 20 epochs on all data sets except the 100bm. The learning curve when training on the 100bm data in Figure 4.1d shows the least improvement in validation loss, the greatest loss values, and the fewest epochs. Overall, we can clearly see that more noise in the data was making it harder for the CNN to improve its predictions.



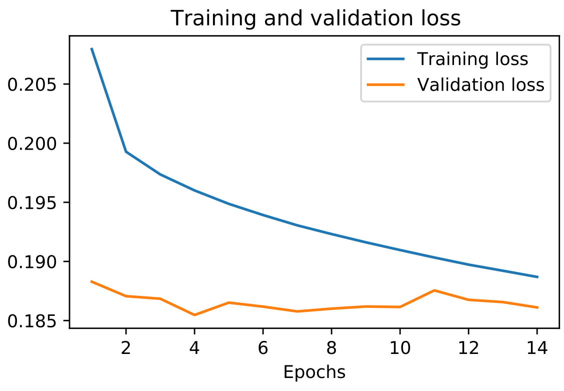
(a) CNN trained on the **10bm** dataset.



(b) CNN trained on the **25bm** dataset.



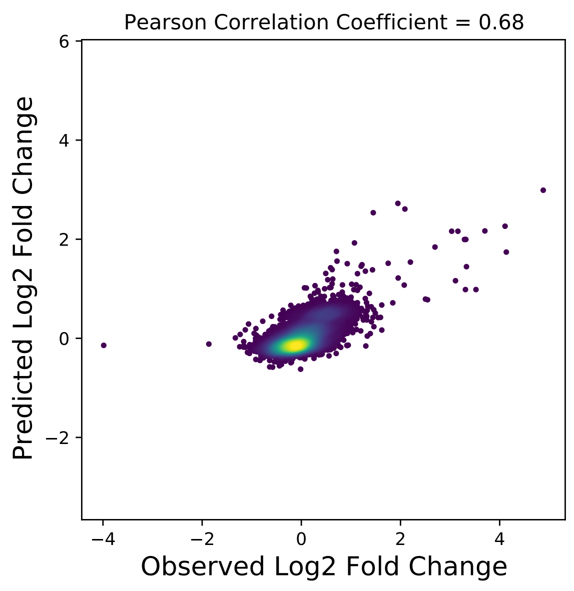
(c) CNN trained on the **50bm** dataset.



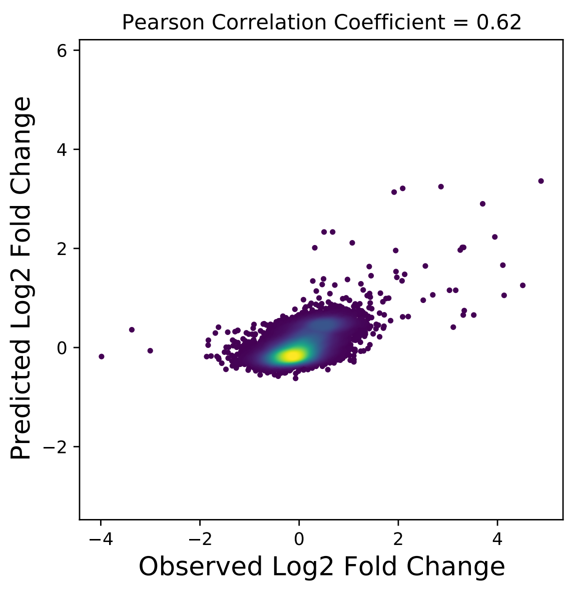
(d) CNN trained on the **100bm** dataset.

Figure 4.1: The CNNs' learning curves plotted as loss after each epoch when trained on different subsets of the data.

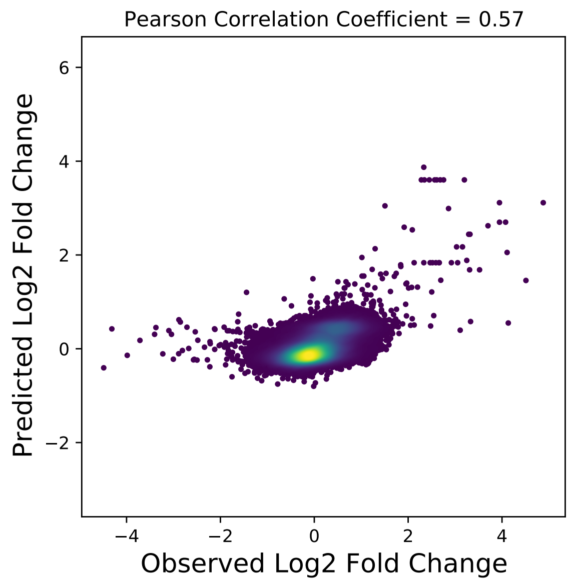
After training the CNN, the model parameters from the best epoch were restored. Then the model was evaluated on the held out chromosome 25 with samples filtered on the same base mean range as the training data. In Figure 4.2, the predicted values versus the observed values are shown for each training and validation set, along with their correlation. The correlation was greatest on the 10bm dataset with a score of 0.68 (Figure 4.2a), and it decreased when we included samples of lower base mean, ending up with a correlation of 0.48 when training and validating on all data (Figure 4.2d). The model struggled to predict the negative observed values in all datasets. In Figure 4.2c and Figure 4.2d, some points form horizontal lines. This indicates that the CNN predicted identical values for sequences where the observed values varied slightly. These could be sequences that are identical or nearly identical, but have variation in observed log2 fold change. This is likely due to randomness, considering that these samples appear only in the data sets that include the lowest base mean ranges, meaning they have the most noise, which explains why the CNN's performance on these data sets is significantly worse.



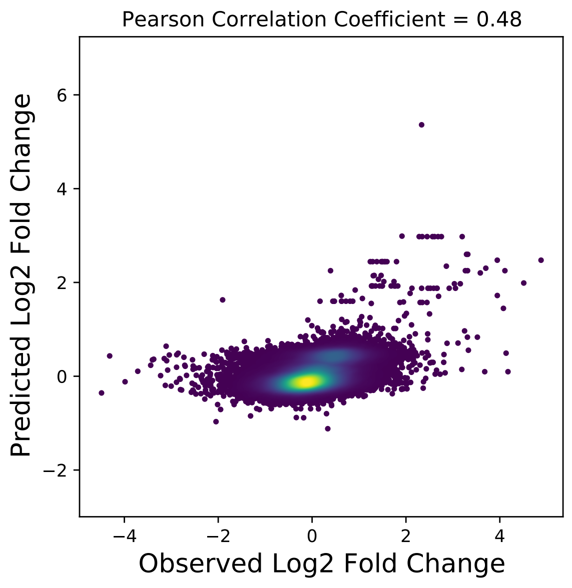
(a) CNN trained on the **10bm** dataset.



(b) CNN trained on the **25bm** dataset.



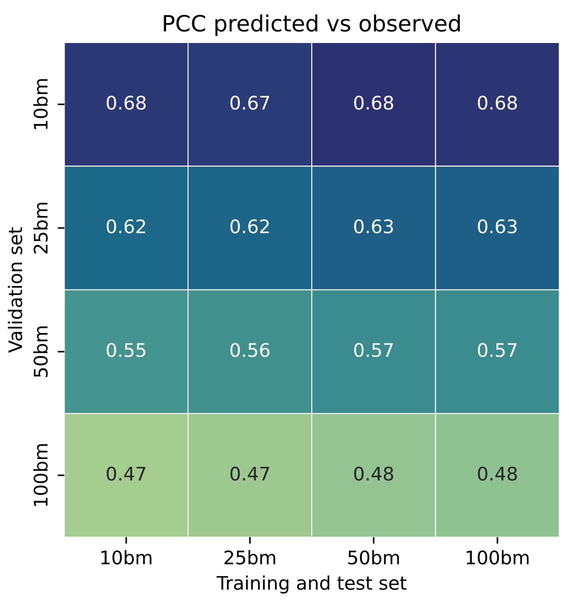
(c) CNN trained on the **50bm** dataset.



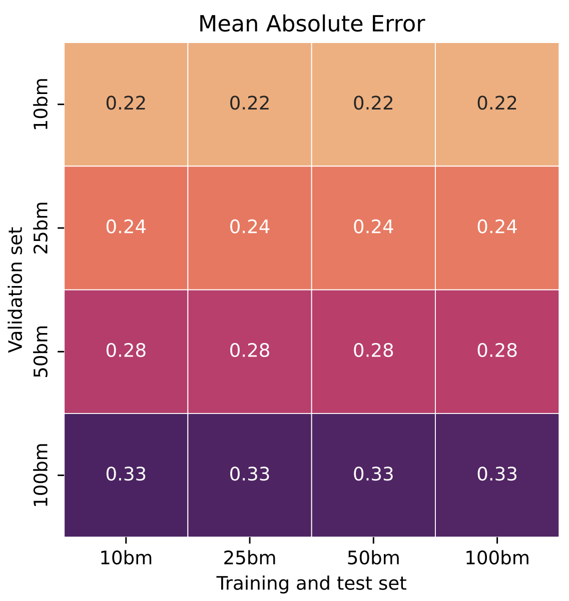
(d) CNN trained on the **100bm** dataset.

Figure 4.2: The CNN trained and validated on different subsets of the data. The plots show predicted versus observed values, and color indicates density of points.

To further test performance, we cross tested the trained models on all validation sets. Results from validating the trained CNNs on different validation data are shown in Figure 4.3. Figure 4.3b shows the Mean Absolute Errors of the predicted log2 fold change by the CNNs on each validation data set. A low MAE indicates a smaller error and hence, a better performing model. In Figure 4.3a the same results are shown using a different metric, the Pearson Correlation Coefficient, where a higher value means that the CNN's predicted values are more correlated with the observed values. Interestingly, the results clearly show how the CNN's performance is only dependent on the validation data, and not affected by more noise in the training data. Furthermore, training on all samples does not increase the performance of the model, despite that the 100bm dataset contains 10 times the number of DNA samples than the 10bm data set. This means that the same information could be learned by the CNN only by training on a tenth of the entire raw data, by selecting the sequences with high base mean.



(a) Cross test results showing Pearson correlation of predicted versus observed values.



(b) Cross test results showing mean absolute error of observed log2 fold change values.

Figure 4.3: CNN performance test results from cross testing the model on different training and validation data.

4.1.2 Motif discovery

Convolution filter motifs

All extracted motifs from the CNNs' first convolution filters were matched with the JASPAR database. In total, 33 motifs were significant matches. We used JASPAR matrix clusters to group the matches by similarity, and present a summary of the best representative motif from each cluster in Figure 4.4. All motifs are included in Appendix C. The TFCP2 was the most occurring match. Most of the filter motifs did not have any significant JASPAR motif match. These motifs in general had less information content and were patchy, which is why they did not resemble any full JASPAR motif. Nonetheless, these filters might account for partial motifs or other features of the sequences.

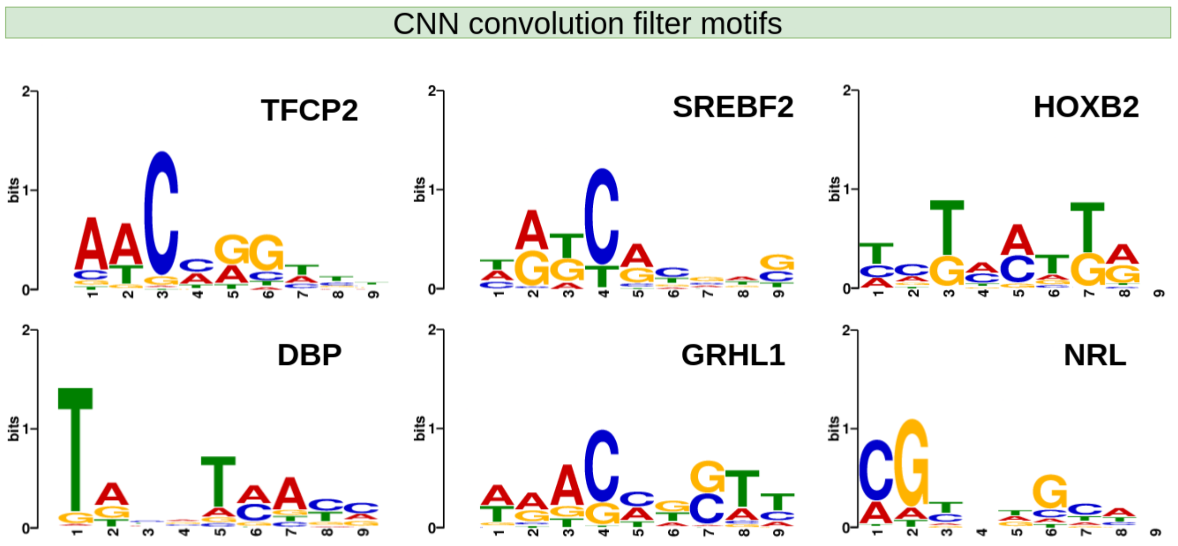


Figure 4.4: JASPAR motifs extracted from the CNNs' first convolutional layers.

Importance score motifs

TF-MoDISco found 29 motifs of the CNN model. Figure 4.5 presents the 7 motifs that had a significant match with JASPAR. These results were different from those of the convolution filters. The TF-MoDISco method aligned subsequences from the calculated importance scores, which came from the collective effects of all convolutional layers in the network. This brought out the most distinct motif pattern: the combination of FOS and JUN, commonly referred to as AP-1. The AP-1 motif was present in the convolutional filters as well, but TF-MoDISco helped reveal the full motif. The TFCP2 motif was found as well.

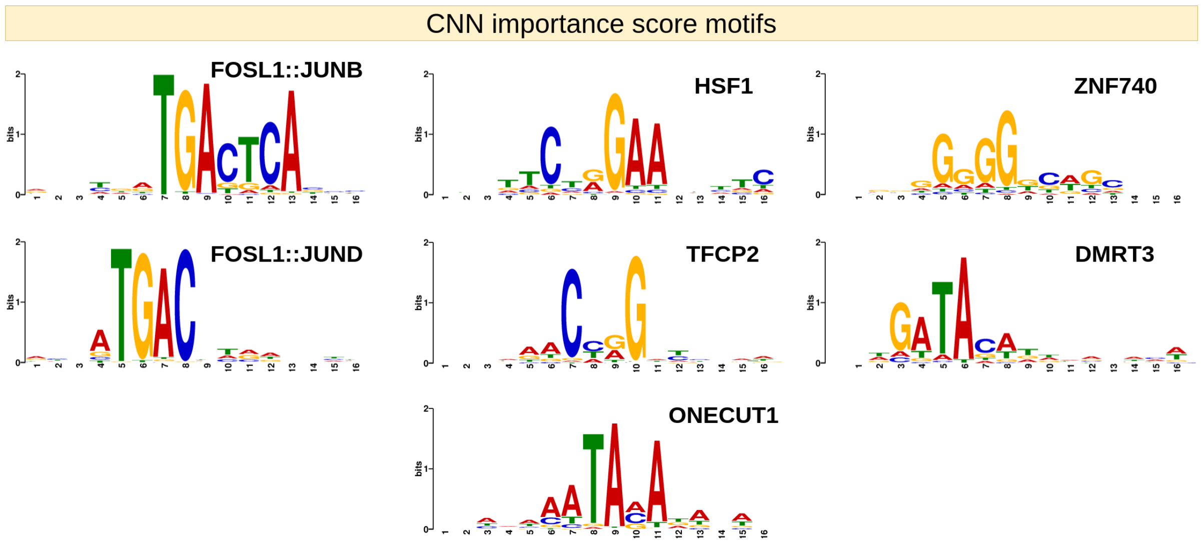


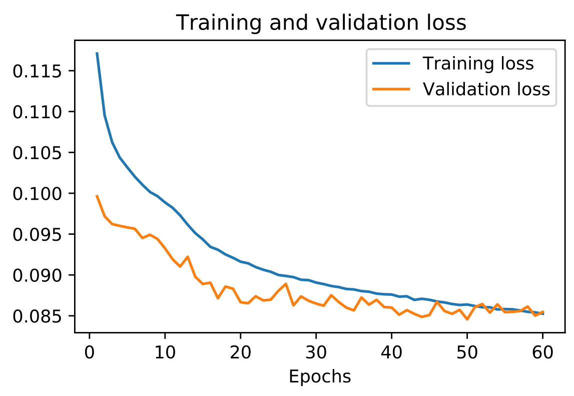
Figure 4.5: JASPAR motifs discovered using TF-MoDISco on SHAP importance scores from the CNN.

4.2 Hybrid Convolutional and Long Short-Term Memory Neural Network

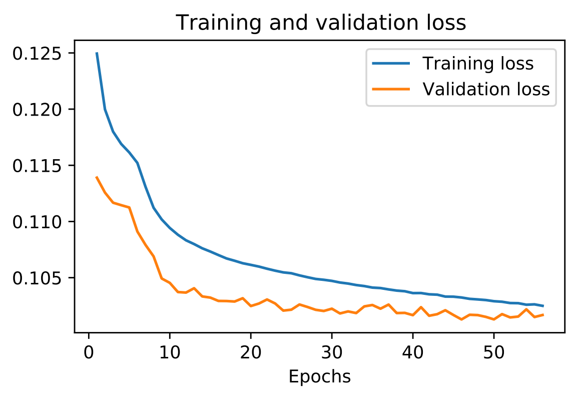
The CNN-LSTM was based on DanQ [13], and tuned on our 10bm dataset. The tuning drastically improved its performance, possibly because the data and our prediction task was very different from that of Quang and Xie [13], and required a less complex model with fewer parameters. In the same way as we did with the CNN, we investigated the effect of noisy data on the model's predictions by training and cross testing it on all combinations of the four datasets. The performance results are presented in Section 4.2.1. After training, we also extracted motifs to compare to the CNN and interpret what the model had learned, and we report the findings in Section 4.2.2.

4.2.1 Performance

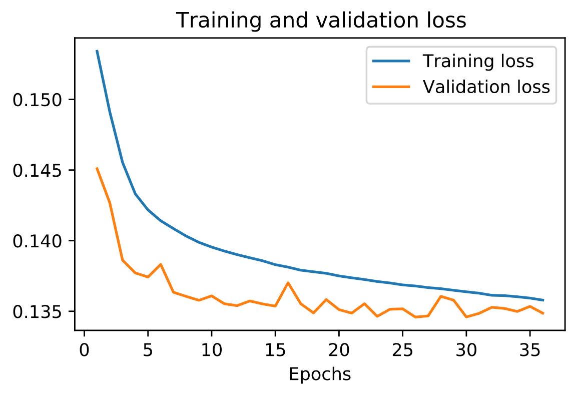
The plots in Figure 4.6 show how training and validation loss changed over time when training the model for multiple epochs on different data sets. We used early stopping to terminate the training when no further improvement of validation loss was seen over the last 10 epochs. We can clearly see that the more noisy data results in a greater loss overall from the different ranges on the y-axis' of the four plots. The CNN-LSTM spent many more epochs training than the CNN. Interestingly, the number of epochs decreases when training on more noisy data, except when training on the full 100bm data set, on which the model spent over 80 epochs training. All the learning curves show a healthy downward trend that stabilizes and converges with time. As with the CNN, the least improvement in validation loss and greatest loss values are seen when training on the full 100bm data (Figure 4.6d). Overall, we can clearly see that more noise in the data is making it harder for the model to improve its predictions.



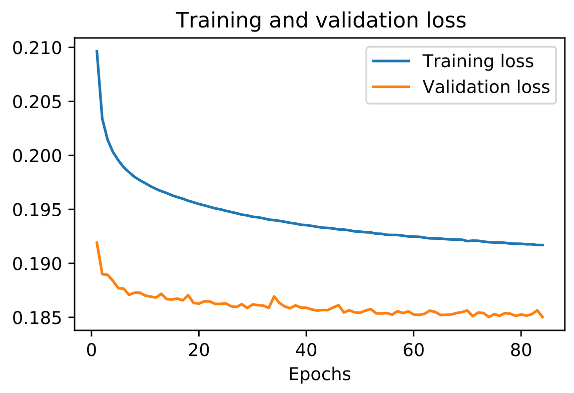
(a) Hybrid model trained on the **10bm** dataset.



(b) Hybrid model trained on the **25bm** dataset.



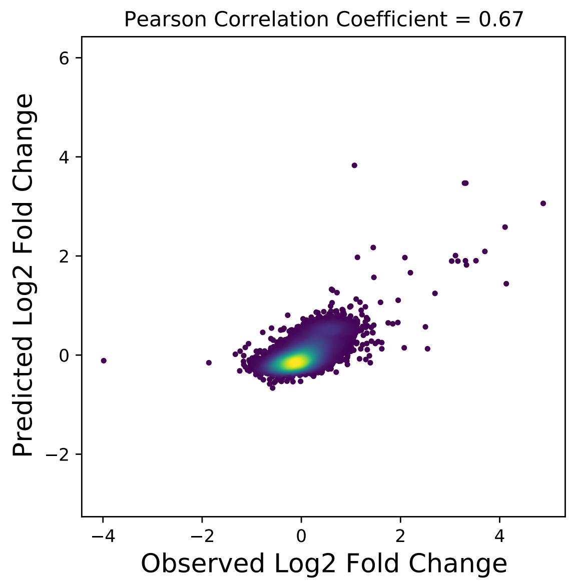
(c) Hybrid model trained on the **50bm** dataset.



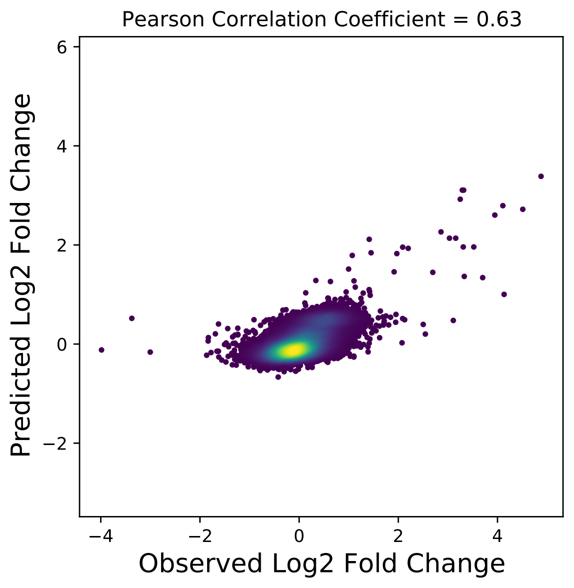
(d) Hybrid model trained on the **100bm** dataset.

Figure 4.6: The hybrid CNN-LSTMs' learning curves plotted as loss after each epoch when trained on different subsets of the data.

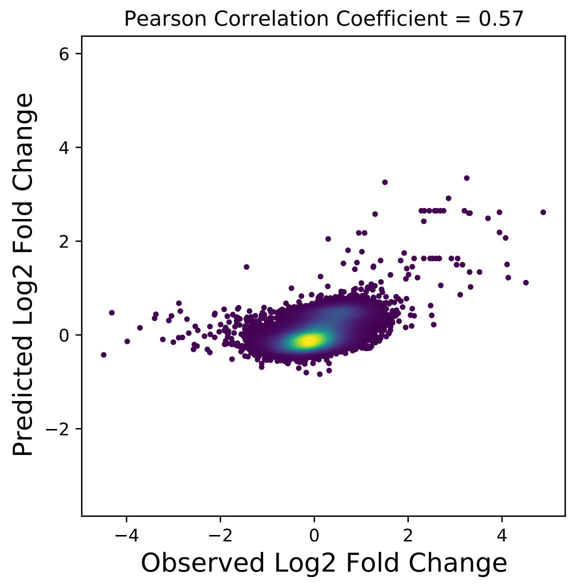
After training the hybrid model on each data set, we evaluated it on the held out chromosome 25 with samples filtered correspondingly as the training data. In Figure 4.7, the predicted values versus the observed values are shown for each training and validation set, along with their correlation. The correlation is greatest on the 10bm dataset with a score of 0.67 (Figure 4.7a), and it decreased when we included more noisy samples, ending up with a correlation of 0.48 when training and validating on all data (Figure 4.2d). The predictions are remarkably similar to those of the CNN. The hybrid model also predicts identical values for sequences where the observed values vary slightly in the data sets that include the lowest base mean ranges. The model struggles to predict the negative observed values in all datasets.



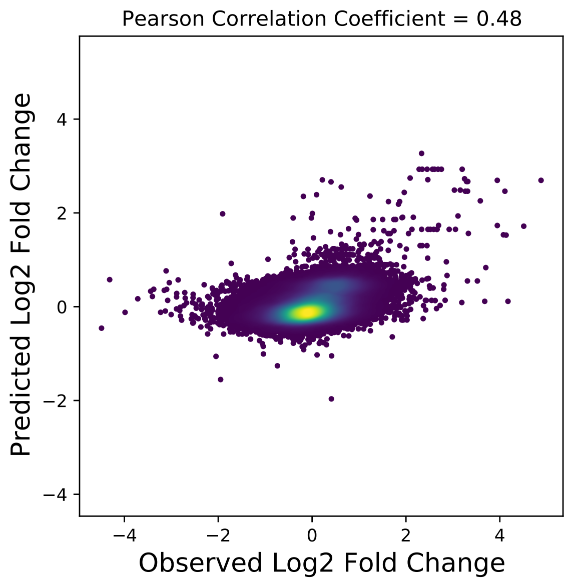
(a) Hybrid CNN-LSTM trained on the **10bm** dataset.



(b) Hybrid CNN-LSTM trained on the **25bm** dataset.



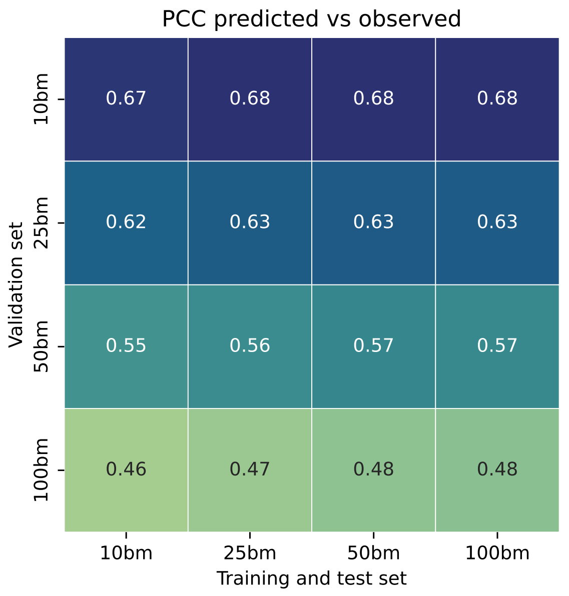
(c) Hybrid CNN-LSTM trained on the **50bm** dataset.



(d) Hybrid CNN-LSTM trained on the **100bm** dataset.

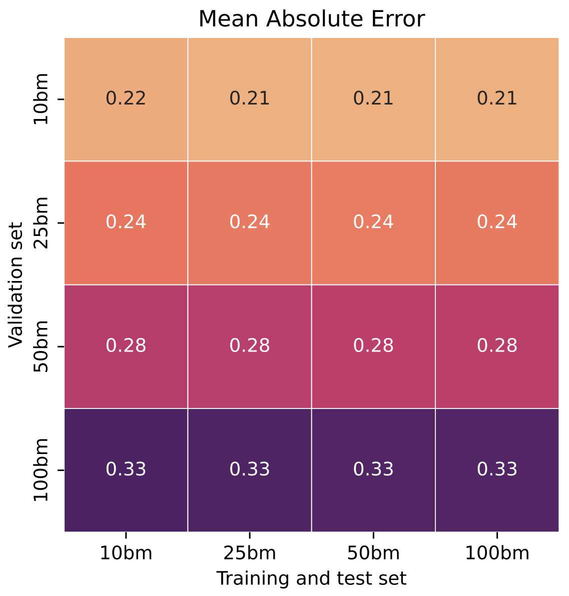
Figure 4.7: The hybrid CNN-LSTM trained and validated on different subsets of the data. The plots show predicted versus observed values, and color indicates density of points.

To further test performance, we cross tested the trained hybrid models on all validation sets. Results from validating the trained CNN-LSTMs on different validation data are shown in Figure 4.8. Figure 4.8b shows the mean absolute errors of the predicted log2 fold change by the CNN-LSTMs on each validation data set. In Figure 4.8a the same results are shown using the Pearson correlation coefficient. As with the CNN, our results again clearly show how the hybrid model's performance is only dependent on the validation data, and not affected by more noise in the training data, and that the sequences of top most base mean include the most information to be learned by the models.



	10bm	25bm	50bm	100bm
10bm	0.67	0.68	0.68	0.68
25bm	0.62	0.63	0.63	0.63
50bm	0.55	0.56	0.57	0.57
100bm	0.46	0.47	0.48	0.48

(a) Cross test results showing Pearson correlation of predicted versus observed values.



	10bm	25bm	50bm	100bm
10bm	0.22	0.21	0.21	0.21
25bm	0.24	0.24	0.24	0.24
50bm	0.28	0.28	0.28	0.28
100bm	0.33	0.33	0.33	0.33

(b) Cross test results showing mean absolute error of observed log2 fold change values.

Figure 4.8: Hybrid CNN-LSTM performance test results. PCC: Pearson Correlation Coefficient. MAE: Mean Absolute Error.

4.2.2 Motif discovery

Convolution filter motifs

All extracted motifs from the CNN-LSTMs' first convolution filters were matched with the JASPAR database. In total, 45 motifs had at least one significant match, all included in Appendix D. We used JASPAR matrix clusters to group the matches by similarity, and present a summary of the best representative motif from each cluster in Figure 4.9. A striking result was that plenty of the CNN-LSTM filters showed variations of the AP-1 motif. We show a selection of them in Figure 4.10.

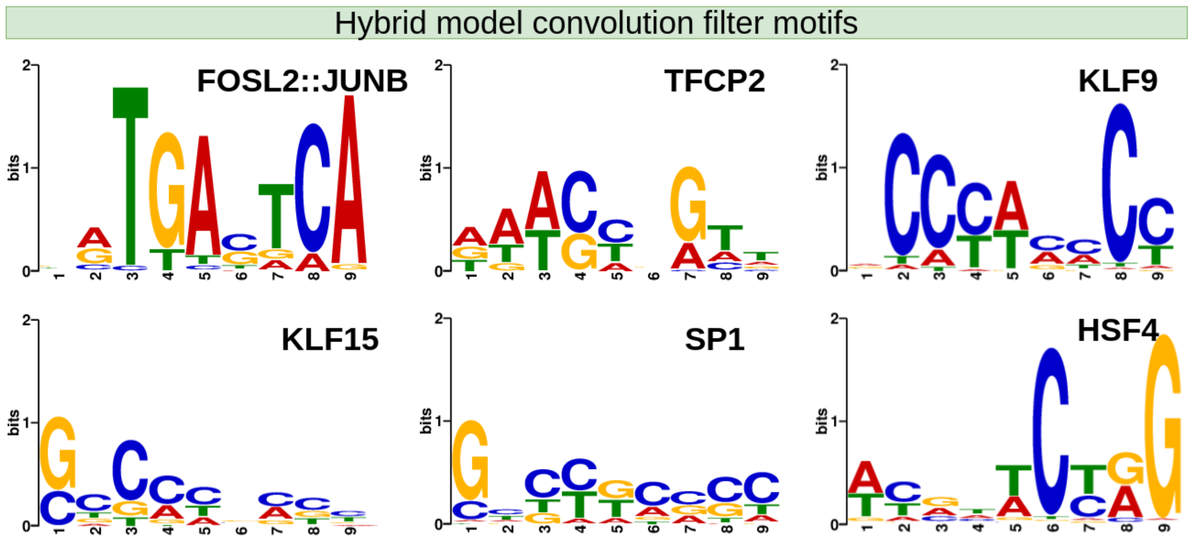


Figure 4.9: JASPAR motifs extracted from the hybrid CNN-LSTMs' first convolutional layers.

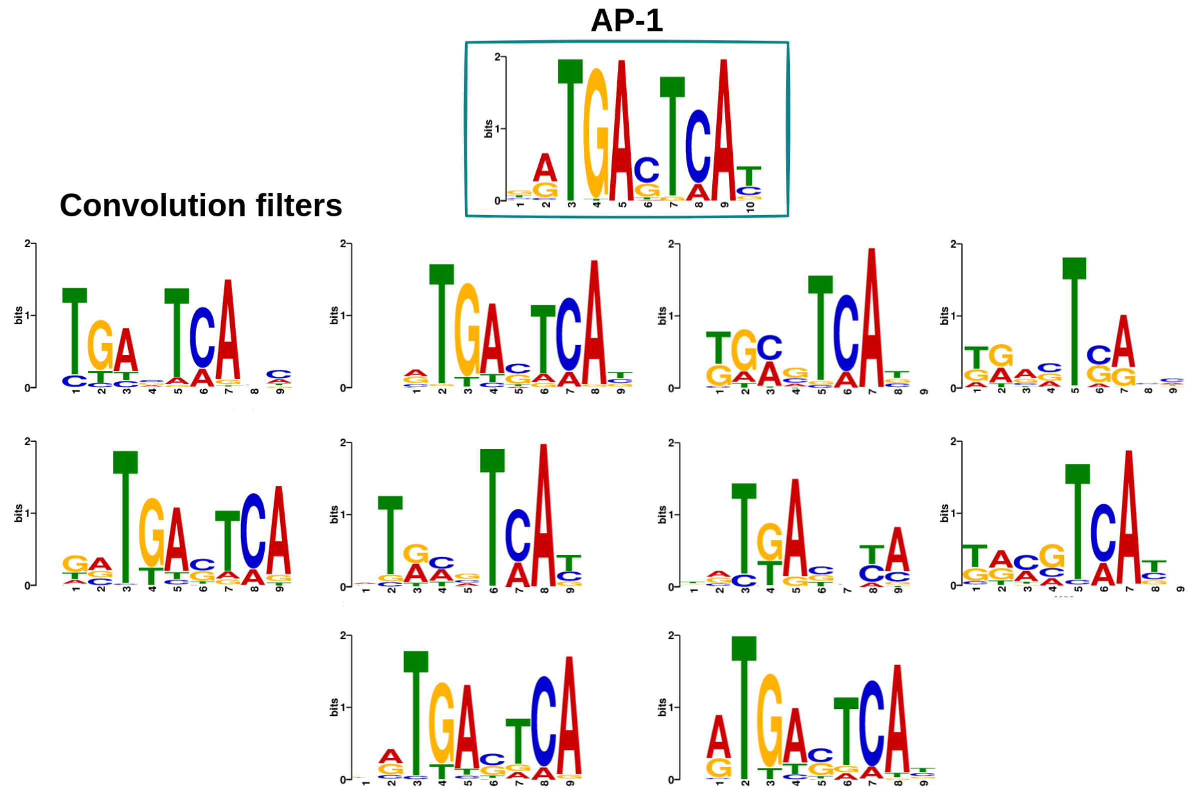


Figure 4.10: Various motifs of the CNN-LSTM convolution filters coincided with the AP-1 motif.

Importance score motifs

TF-MoDISco found 34 motifs of the Hybrid model. Figure 4.11 presents the 9 motifs that had a significant match with JASPAR. The TF-MoDISco results again differed from those of the convolution filters. The THAP1 motif is clearly of importance to the hybrid model, and we can see two versions of it that emphasize different parts of the motif, but it was not encountered when only looking at the first convolution filters. The AP-1 motif, here represented by the MAFK motif, has less information content and is very different from the AP-1 motifs we found among the convolution filters. This might be due to the numerous filters that had variations of the AP-1 motif, which are all aligned into the more watered-down MAFK result with TF-MoDISco (lower-right corner, Figure 4.11). The KLF15 motif was found with both methods.

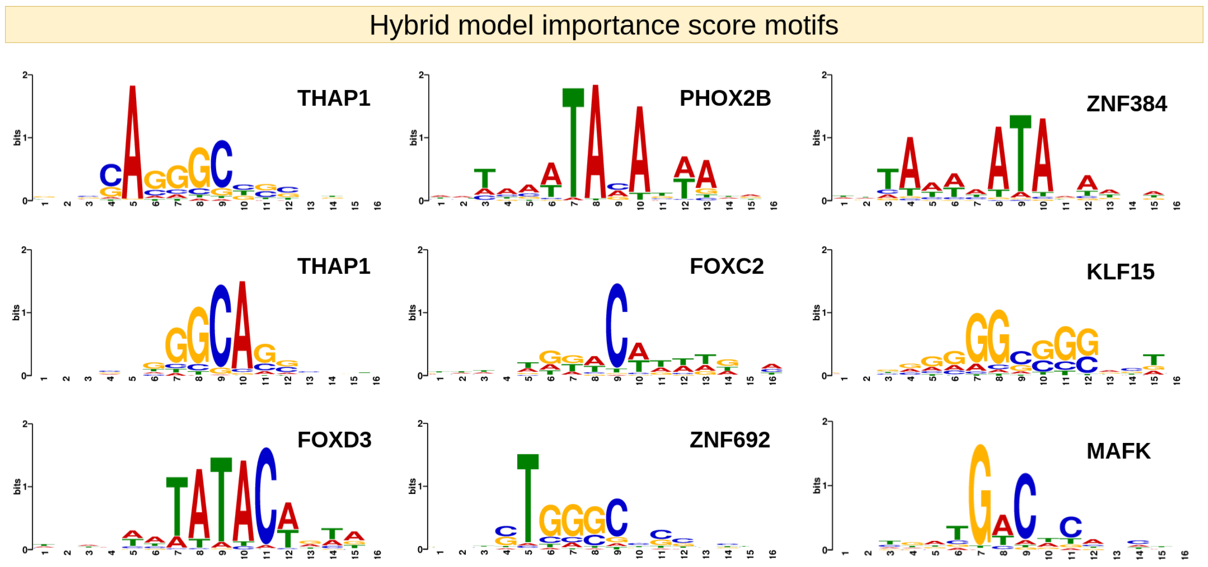


Figure 4.11: Motifs discovered using TF-MoDISco on SHAP importance scores from the hybrid CNN-LSTM.

CHAPTER 5 DISCUSSION

5.1 Performance and data

The convolutional model (CNN) and the hybrid convolutional recurrent model (CNN-LSTM) learned to predict gene expression from sequence with varying success. The performance depended on the base mean of the validation samples, where fragments with a clear ability to drive transcription (high base mean) were easier to predict. We also saw that different levels of noise (base mean) in the training data had no effect on predictions, which suggests that the models were robust to noise. No model could predict negative values well, indicating that they were unable to find patterns or features related to repressing gene expression effects. Both of the tested models' predictions reached a Pearson correlation coefficient (PCC) of 0.68 on the samples with base mean greater than 62, which was the highest range tested. The DeepSTARR model created by Almeida et al. [1] predicted log2 fold change values from UMI-STARR-seq fragments. UMI-STARR-seq applies unique molecular identifiers (UMIs) to accurately count RNA, which is primarily used for STARR-seq libraries of low complexity [45]. Additionally, only fragments of length 249bp were used. DeepSTARR's predictions on housekeeping enhancers agreed with the observed values with a PCC of 0.74, and a PCC of 0.68 for developmental enhancers. In this thesis, we used ATAC-STARR-seq fragments that were much shorter (see Figure 3.1). We did not select lengths, but trimmed longer fragments to a maximum of 200bp and added zero padding to the shorter fragments. The ATAC-STARR-seq fragments were also largely non-overlapping, resulting in a highly complex library. The ability of any machine learning model to learn relationships in data is limited by the quality of the data. We showed that the models learned the same patterns regardless of noise in the data, but the short fragments and complex library might stall further performance. Both models reached the same performance levels, which also suggests a data quality limit. It would be interesting to see if the predictions improved further by using longer, more overlapping fragments, and a stronger promoter in the ATAC-STARR-seq experiment to drive more transcription and increase the overall base mean of all fragments. This might provide a richer foundation for deep models to learn regulatory syntax. Further research could also look into the more specific effects of fragment lengths and overlap on prediction.

5.2 Deep neural network architectures

The convolutional neural network architecture consisted of 4 convolution and pooling parts (illustrated in Figure 3.6), each pooling halving the feature map of the convolution. The network could thus learn features and patterns of the DNA sequences and their constellations and positions. The hybrid convolutional long short-term memory neural network only had one convolution and pooling layer. The two architectures had identical first layers, but the motifs found in the first layers were different. Motifs from the CNN's first layer often had only one or a few positions with strong information in bits, while the motifs from the CNN-LSTM's first layers were fuller and more composite. This might be the reason the CNN-LSTM got more significant matches to known JASPAR motifs. In light of the differences in architecture, it is plausible that the CNN utilizes its four convolution parts to model motif patterns and locations, such that each filter of the first convolution need only portray sub-patterns, like pieces of a puzzle. This partial motif modelling is supported by the findings in a study by Koo and Eddy [46]. The CNN-LSTM on the other hand, only has one convolution layer, and then a few LSTM cells that could account for the order and positions of the patterns found in its convolution filters; therefore the convolutional filters would represent bigger puzzle pieces of the DNA sequence patterns. The two architectures performed equally at predicting gene expression, despite their different modeling approach. It is worth noting that the CNN-LSTM had fewer trainable parameters, but spent more epochs training. Overall, both architectures were appropriate for the ATAC-STARR-seq DNA sequences because they found local patterns and accounted for spatial relationships between them.

5.3 Motif extraction methods

Motifs were extracted from the filters of the very first convolution layer and also from importance scores of sequences based on the models' predictions with TF-MoDISco. The first method only considers one layer of the model, while the importance scores result from additively propagating scores through the whole network. In this regard, the TF-MoDISco method provides insight into a model's attention on the DNA sequence during the entire prediction process. The CNN-LSTM had more consistent results between the two motif extraction methods than the CNN. The first method, using only one layer to explain a model's predictions, was perhaps too simple for a complex model of repeated convolution layers like the CNN, but appropriate for the CNN-LSTM that only modelled sequence patterns in the one convolution layer. Moreover, due to a limited number of validation sequences, there might be patterns and features we did not discover or completely reveal from the models. The variable motif results suggest that neither of the two methods exposed exactly how the models utilized sequence features, and that the motifs are part of a more complex decision process leading to the final prediction.

5.4 Motif relevance for hepatocyte gene regulation

The most recurring and significant JASPAR motif we found were FOS/JUN transcription factors (Figure 4.10), collectively called Activator Protein-1 (AP-1). AP-1 mediates gene regulation in response to growth factors, stresses, and other cell signals during cell growth, differentiation and programmed cell death or transformation [47]. AP-1 members also include Atf1 and Maf variants like Mafg, Mafb, and Mafk, which we also found among the convolution filter motifs. AP-1 is known to regulate genes in liver cells [48]. In addition, there were some interesting findings among the significant motif matches regarding hepatocyte (liver cell) regulation specifically. For example, KLF15 and DBP are known liver-specific transcription factors [49] and ONECUT1 is a hepatocyte nuclear factor also known as HNF6 [50].

A study by Reed et al. [51] found that SP1 and SREBP1 coordinate complex transcriptional responses in the liver together with another transcription factor NFY. We did not find NFY, but paralogs of SP1 and SREBP1 (SP4 and SREBP2, respectively) occurred in our results. This might be due to their role in hepatocyte regulation, but validation of these motifs is needed to conclude.

Another common hit was the TFCP2 and paralog motifs like GRHL1, GRHL2 and Tfcp2l1 motifs. They are members of the Grainyhead family of proteins that are involved in many biological events, including regulation of cellular and viral promoters, cell cycle, DNA synthesis and cell survival. The TFCP2 is a general transcription factor with low tissue specificity. Studies have shown that overexpression of TFCP2 plays a key role in human liver cancer cells (hepatocyte carcinoma) [52]. This indicates that the basal TFCP2 activity is vital for normal liver cell health. We also found other general factors, such as the heat shock factors (HSF), which are common regulators of stress response in many eukaryotic cell types [53].

Overall, many JASPAR motifs were found, but the results varied between models and motif extraction methods. Some motifs were relevant for hepatocyte regulation specifically, and many motifs were not. There are also many commonly known liver-specific motifs that we did not find in our JASPAR comparison, most notably many of the HNF motifs.

5.5 Remaining Challenges

The genome naturally has repeating sequences, because of transposable elements for instance, which is hard to account for when training a deep learning model. In the worst-case scenario, it may give rise to a deceivingly high performance. The salmon chromosomes 21 and 25 are very similar; therefore we did not include them in the training data, but used them for testing and validation. Even so, the effect of sequence repeats, as well as fragment overlap, is an important question for further research.

Determining reliability of results gets complicated in analysis pipelines of many stages, as each stage has some level of uncertainty associated with it. In our case, there is some uncertainty introduced in all steps, including DNA sequencing, differential expression analysis, deep learning, and model interpretation. Discussion of results is based on an assumption that our data and results are representative of real events. A remaining challenge of this work is to thoroughly address confidence behind the experiment and analysis.

We would like to mention that the JASPAR database contains curated motifs validated by previous studies, and is continuously being updated with new motif discoveries. Comparing extracted features to JASPAR therefore introduces a historical bias to our interpretation of the results. The deep neural networks are not restricted and could learn novel motifs that would not be detected when compared to JASPAR.

5.6 Future Work

A deep learning framework always has a multitude of hyperparameters, which opens up a range of possibilities for optimization. We only explored a few hyperparameters of each model, and there are more architectures to test. For example, we used LSTM cells in our recurrent model, where Gated Recurrent Units are another option. Additionally, further tests of motif extraction methods include applying deepExplainer and TF-MoDISco on all trained models, and testing the impact of different sizes of mini batches when searching for convolution filter motifs. In further work, validation of these motif results would be needed to confidently answer whether the models learned to recognize regulatory sequences.

An interesting idea would be to flip the task of training a model on DNA to extract motifs, by hard-coding known motifs into the model's filter parameters before training. Quang and

Xie [13] initialized half of the filters with known motifs from JASPAR and saw an improved performance of DanQ, but they simultaneously increased the number of filters; It would therefore be interesting to test the sole effect of the custom JASPAR parameter initialization. The model might abandon these parameters and adjust them into different values, or it might find them useful for prediction and improve performance. Another way is to restrict the possible values of the convolution filters and design the CNN such that it cannot learn partial filters, but inherently model whole motifs in each filter [46]. This might limit the model's learning ability, but it could lead to a more easily interpretable model.

Training multiple models to become experts on subsets of the problem (ensemble method) could out-perform one single model [20]. In our case, the models could, for example, each cover a small range of fragment lengths. Further, transformer models would be natural next candidates for DNA language modelling, given their recent success [8]. Future work holds plenty of potential approaches for modelling DNA language more accurately. The development of more understandable models and tools to extract insight from neural networks, will be an important field of study in the imminent future, following the widespread and increasing application of artificial intelligence. It is essential to explain a model's decision process to establish trust and consider it valuable both in practical use as a predictor, and in research.

CHAPTER 6 CONCLUSION

In this thesis, we asked whether deep neural networks could predict gene regulation effects from ATAC-STARR-seq fragments of the Atlantic salmon. The main objectives were to test prediction performance, and to explore whether the models learned to recognize regulatory syntax.

Regarding performance, we revealed that the base mean was an important measurement of noise in the data, and saw that the sequences of greatest base mean were predicted with the highest correlation to the observed values. The models' performances were similar to the DeepSTARR model [1] and we discussed that data quality might limit prediction performance. We therefore suggest that longer and more overlapping fragments, decreased STARR-seq library complexity, and increased base mean overall, might provide better data for training deep neural networks.

Further, we extracted motifs from the trained models to assess what they had learned. The performances indicate that the models were able to find predictive features in the DNA sequences, and some extracted motifs did match known vertebrate transcription factors binding site motifs. We conclude that the models learned to read regulatory syntax, and that there is a potential to extract more knowledge from deep neural networks. In the future, deep model interpretation may become an important approach for regolomics analysis.

The motif findings of the two network architectures were different, which posed an interesting discussion about how different deep neural networks learn to represent sequence features. We point out the historical bias of comparing our motifs to known curated JASPAR motifs, and see a potential for deep models to discover novel enhancers and promotors in future work. To fully understand our deep models, more sophisticated and robust methods are needed, as well as systematic ways to validate the motif findings. This will be an important area of research in the future of genomics specifically, but also for deep learning in general. Future work may uncover alternative approaches and deep learning architectures fitting for DNA language modelling, the transformer being a suggested candidate based on recent studies.

BIBLIOGRAPHY

[1] Bernardo P. de Almeida et al. “DeepSTARR predicts enhancer activity from DNA sequence and enables the de novo design of synthetic enhancers”. In: *Nature Genetics* 54.5 (May 2022), pp. 613–624. DOI: `10.1038/s41588-022-01048-5`.

[2] Duc-Hau Le. “Machine learning-based approaches for disease gene prediction”. In: *Briefings in Functional Genomics* 19.5-6 (Dec. 2020), pp. 350–363. DOI: `10.1093/bfgp/elaa013`.

[3] Amani Al-Ajlan and Achraf El Allali. “CNN-MGP: Convolutional Neural Networks for Metagenomics Gene Prediction”. In: *Interdisciplinary Sciences: Computational Life Sciences* 11.4 (Dec. 2019), pp. 628–635. DOI: `10.1007/s12539-018-0313-4`.

[4] Raíssa Silva et al. “geneRFinder: gene finding in distinct metagenomic data complexities”. In: *BMC Bioinformatics* 22.1 (Feb. 2021), p. 87. DOI: `10.1186/s12859-021-03997-w`.

[5] William S. Klug et al. *Essentials of Genetics, Global Edition.* Pearson Education, May 2016. ISBN: 978-1-292-10893-3.

[6] Granton A. Jindal and Emma K. Farley. “Enhancer grammar in development, evolution, and disease: dependencies and interplay”. In: *Developmental Cell* 56.5 (Mar. 2021), pp. 575–587. DOI: `10.1016/j.devcel.2021.02.016`.

[7] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.* May 2019. DOI: `10.48550/arXiv.1810.04805`.

[8] Yanrong Ji et al. “DNABERT: pre-trained Bidirectional Encoder Representations from Transformers model for DNA-language in genome”. In: *Bioinformatics* 37.15 (Aug. 2021), pp. 2112–2120. DOI: `10.1093/bioinformatics/btab083`.

[9] “Base-resolution models of transcription-factor binding reveal soft motif syntax”. In: *Nature Genetics* 53.3 (Mar. 2021), pp. 354–366. DOI: `10.1038/s41588-021-00782-6`.

[10] Timothy L. Bailey and Charles Elkan. “Unsupervised learning of multiple motifs in biopolymers using expectation maximization”. In: *Machine Learning* 21.1 (Oct. 1995), pp. 51–80. DOI: `10.1007/BF00993379`.

[11] Jan Zrimec et al. “Controlling gene expression with deep generative design of regulatory DNA”. In: *Nature Communications* 13.1 (Aug. 2022), p. 5099. DOI: `10.1038/s41467-022-32818-8`.

[12] Babak Alipanahi et al. “Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning”. In: *Nature Biotechnology* 33.8 (Aug. 2015), pp. 831–838. DOI: `10.1038/nbt.3300`.

[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 (June 2016), e107. DOI: 10.1093/nar/gkw226.

[14] Xinchen Wang et al. “High-resolution genome-wide functional dissection of transcriptional regulatory regions and nucleotides in human”. In: *Nature Communications* 9.1 (Dec. 2018), p. 5380. DOI: 10.1038/s41467-018-07746-1.

[15] National Human Genome Research Institute. *Deoxyribonucleic acid (DNA).* URL: https://www.genome.gov/genetics-glossary/Deoxyribonucleic-Acid (visited on 03/28/2023).

[16] National Human Genome Research Institute. *Central dogma.* URL: https://www.genome.gov/genetics-glossary/Central-Dogma (visited on 04/20/2023).

[17] National Human Genome Research Institute. *Chromatin.* URL: https://www.genome.gov/genetics-glossary/Chromatin (visited on 03/28/2023).

[18] Jason C. Klein et al. “A systematic evaluation of the design and context dependencies of massively parallel reporter assays”. In: *Nature Methods* 17.11 (Nov. 2020), pp. 1083–1091. DOI: 10.1038/s41592-020-0965-y.

[19] Tyler J Hansen and Emily Hodges. “ATAC-STARR-seq reveals transcription factor–bound activators and silencers within chromatin-accessible regions of the human genome”. In: *Genome Research* 32.8 (2022), pp. 1529–1541.

[20] Sebastian Raschka and Vahid Mirjalili. *Python Machine Learning - Second Edition: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow.* 2nd edition. Packt Publishing, Sept. 2017. ISBN: 978-1-78712-593-3.

[21] Warren S. McCulloch and Walter Pitts. “A logical calculus of the ideas immanent in nervous activity”. In: *The bulletin of mathematical biophysics* 5.4 (Dec. 1943), pp. 115–133. DOI: 10.1007/BF02478259.

[22] Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton.* Cornell Aeronautical Laboratory, 1957.

[23] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. “Chapter 6: Deep Feedforward Networks”. In: *Deep Learning.* Vol. 1. MIT Press, 2016, pp. 164–223. URL: http://www.deeplearningbook.org.

[24] Jacob Schreiber et al. “A pitfall for machine learning methods aiming to predict across cell types”. In: *Genome Biology* 21.1 (Nov. 2020), p. 282. DOI: 10.1186/s13059-020-02177-y.

[25] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. “Chapter 7: Regularization for Deep Learning”. In: *Deep Learning.* Vol. 1. MIT Press, 2016, pp. 224–270. URL: http://www.deeplearningbook.org.

[26] Nitish Srivastava et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[27] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *Proceedings of the 32nd International Conference on Machine Learning.* Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. PMLR, July 2015, pp. 448–456. URL: https://proceedings.mlr.press/v37/ioffe15.html.

[28] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. “Chapter 9: Convolutional Networks”. In: *Deep Learning.* Vol. 1. MIT Press, 2016, pp. 326–366. URL: http://www.deeplearningbook.org.

[29] Aston Zhang et al. *Dive into Deep Learning.* 2023. arXiv: 2106.11342 [cs.LG].

[30] Cort J. Willmott and Kenji Matsuura. "Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance". In: *Climate Research* 30.1 (2005), pp. 79–82. DOI: 10.3354/cr030079.

[31] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, eds. *Automated Machine Learning: Methods, Systems, Challenges.* The Springer Series on Challenges in Machine Learning. Springer International Publishing, 2019. DOI: 10.1007/978-3-030-05318-5.

[32] Longwei Wang et al. "Explaining the Behavior of Neuron Activations in Deep Neural Networks". In: *Ad Hoc Networks* 111 (Feb. 2021), p. 102346. DOI: 10.1016/j.adhoc.2020.102346.

[33] Scott M Lundberg and Su-In Lee. "A Unified Approach to Interpreting Model Predictions". In: *Advances in Neural Information Processing Systems 30.* Ed. by I. Guyon et al. Curran Associates, Inc., 2017, pp. 4765–4774. URL: http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf.

[34] Zhuwei Qin et al. "How convolutional neural networks see the world — A survey of convolutional neural network visualization methods". In: *Mathematical Foundations of Computing* 1.2 (May 2018). Publisher: Mathematical Foundations of Computing, pp. 149–180. DOI: 10.3934/mfc.2018008.

[35] Jack Lanchantin et al. "Deep motif dashboard: visualizing and understanding genomic sequences using deep neural networks". In: *Biocomputing 2017.* World Scientific, Nov. 2016, pp. 254–265. DOI: 10.1142/9789813207813_0025.

[36] Michael Love et al. *DESeq2: Differential gene expression analysis based on the negative binomial distribution.* 2023. DOI: 10.18129/B9.bioc.DESeq2. URL: https://bioconductor.org/packages/DESeq2/.

[37] Tom O'Malley et al. *KerasTuner.* https://github.com/keras-team/keras-tuner. 2019.

[38] Daniel Quang and Xiaohui Xie. "FactorNet: A deep learning framework for predicting cell type specific transcription factor binding from nucleotide-resolution sequential data". In: *Methods.* Deep Learning in Bioinformatics 166 (Aug. 2019), pp. 40–47. DOI: 10.1016/j.ymeth.2019.03.020.

[39] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. *Learning Important Features Through Propagating Activation Differences.* Oct. 2019. DOI: 10.48550/arXiv.1704.02685.

[40] Avanti Shrikumar et al. *Technical Note on Transcription Factor Motif Discovery from Importance Scores (TF-MoDISco) version 0.5.6.5.* Apr. 2020. DOI: 10.48550/arXiv.1811.00416.

[41] Shobhit Gupta et al. "Quantifying similarity between motifs". In: *Genome Biology* 8.2 (Feb. 2007), R24. DOI: 10.1186/gb-2007-8-2-r24.

[42] Timothy L. Bailey et al. "The MEME Suite". In: *Nucleic Acids Research* 43.W1 (July 2015), W39–W49. DOI: 10.1093/nar/gkv416.

[43] Hung Viet Pham et al. "Problems and opportunities in training deep learning software systems: an analysis of variance". In: *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering.* ASE '20. Association for Computing Machinery, Jan. 2021, pp. 771–783. DOI: 10.1145/3324884.3416545.

[44] Benjamin J. Heil et al. "Reproducibility standards for machine learning in the life sciences". In: *Nature Methods* 18.10 (Oct. 2021), pp. 1132–1135. DOI: 10.1038/s41592-021-01256-7.

[45] Christoph Neumayr et al. “STARR-seq and UMI-STARR-seq: Assessing Enhancer Activities for Genome-Wide-, High-, and Low-Complexity Candidate Libraries”. In: *Current Protocols in Molecular Biology* 128 (2019). DOI: 10.1002/cpmb.105.

[46] Peter K. Koo and Sean R. Eddy. “Representation learning of genomic sequence motifs with convolutional neural networks”. In: *PLoS Computational Biology* 15.12 (Dec. 2019), e1007560. DOI: 10.1371/journal.pcbi.1007560.

[47] Peter Angel and Marina Schorpp-Kistner. “Jun/Fos”. In: *Encyclopedic Reference of Genomics and Proteomics in Molecular Medicine*. Springer, 2006, pp. 928–935. DOI: 10.1007/3-540-29623-9_4560.

[48] Julia I. Leu et al. “Interleukin-6-Induced STAT3 and AP-1 Amplify Hepatocyte Nuclear Factor 1-Mediated Transactivation of Hepatic Genes, an Adaptive Response to Liver Injury”. In: *Molecular and Cellular Biology* 21.2 (Jan. 2001), pp. 414–424. DOI: 10.1128/MCB.21.2.414-424.2001.

[49] Y Hayashi et al. “Liver enriched transcription factors and differentiation of hepatocellular carcinoma.” In: *Molecular Pathology* 52.1 (Feb. 1999), pp. 19–24. ISSN: 1366-8714.

[50] Hwee Hui Lau et al. “The molecular functions of hepatocyte nuclear factors – In and beyond the liver”. In: *Journal of Hepatology* 68.5 (May 2018), pp. 1033–1048. DOI: 10.1016/j.jhep.2017.11.026.

[51] Brian D. Reed et al. “Genome-Wide Occupancy of SREBP1 and Its Partners NFY and SP1 Reveals Novel Functional Roles and Combinatorial Regulation of Distinct Classes of Genes”. In: *PLOS Genetics* 4.7 (July 2008), e1000133. DOI: 10.1371/journal.pgen.1000133.

[52] Prasanna K Santhekadur et al. “The transcription factor LSF: a novel oncogene for hepatocellular carcinoma”. In: *American Journal of Cancer Research* 2.3 (Apr. 2012), pp. 269–285. ISSN: 2156-6976.

[53] Kevin A. Morano and Dennis J. Thiele. “Heat Shock Factor Function and Regulation in Response to Cellular Stress, Growth, and Differentiation Signals”. In: *Gene Expression* 7.4-5-6 (Sept. 2018), pp. 271–282. ISSN: 1052-2166.

[54] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems.* Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[55] J. D. Hunter. “Matplotlib: A 2D graphics environment”. In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[56] Michael L. Waskom. “seaborn: statistical data visualization”. In: *Journal of Open Source Software* 6.60 (2021), p. 3021. DOI: 10.21105/joss.03021.

[57] Charles R. Harris et al. “Array programming with NumPy”. In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[58] The pandas development team. *pandas-dev/pandas: Pandas.* Version 1.5.2. Feb. 2022. DOI: 10.5281/zenodo.7344967.

[59] Pauli Virtanen et al. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[60] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[61] Avanti Shrikumar et al. *kundajelab/tfmodisco: Bringing down Leiden memory use - patch 1.* Jan. 2022. DOI: 10.5281/zenodo.5909083.

APPENDIX A SOFTWARE AND HARDWARE

A.1 Software

Table A.1: Python modules used for data exploration, preprocessing and deep learning, along with the respective versions and purpose.

Module name	Version	Purpose of use	Reference
TensorFlow	2.5.0	Deep learning backend	[54]
KerasTuner	1.2.0	Hyperparameter tuning	[37]
Matplotlib	3.3.4	Data visualization	[55]
Seaborn	0.12.2	Data visualization	[56]
NumPy	1.19.5	Scientific computing	[57]
Pandas	1.1.5	Data wrangling	[58]
SciPy	1.5.4	Statistics	[59]
Scikit-learn	0.24.2	Model metrics and train_test_split function	[60]
SHAP	0.40.0	DeepExplainer for importance scores	[39]
TF-MoDISco	0.5.16.0	Motif discovery from importance scores	[61]

A.2 Hardware

The authors acknowledge the Orion High Performance Computing Center (OHPCC) at the Norwegian University of Life Sciences (NMBU) for providing computational resources that have contributed to the research results reported within this paper. URL (internal): https://orion.nmbu.no

Table A.2: Hardware specifications.

GPU	RAM	CPU type	Clock rate
NVIDIA Quadro RTX 8000	64 GB	EPYC 7302 16-Core	3.0 GHz

APPENDIX B
SOURCE CODE AND TRAINED MODELS

Table B.1: Table of source code with GitHub links and commit hashes.

Source Code	Link	Commit Hash
CNN_training.py	*GitHub* 🔗	48abd76
CNN_tuner.py	*GitHub* 🔗	f34416f
CNN_deepExplainer.py	*GitHub* 🔗	5f8368c
CNN_modisco.ipynb	*GitHub* 🔗	2293a27
CNN_conv_motifs.py	*GitHub* 🔗	5343d9a
Hybrid_training.py	*GitHub* 🔗	48abd76
Hybrid_tuner.py	*GitHub* 🔗	514d9d0
Hybrid_deepExplainer.py	*GitHub* 🔗	4a51f87
Hybrid_modisco.ipynb	*GitHub* 🔗	1f54089
Hybrid_conv_motifs.py	*GitHub* 🔗	5343d9a

Table B.2: Table of trained model files with GitHub links and commit hashes.

File name	Link	Commit Hash
CNN10bm.h5	*GitHub* 🔗	0a2798a
CNN25bm.h5	*GitHub* 🔗	0a2798a
CNN50bm.h5	*GitHub* 🔗	0a2798a
CNN100bm.h5	*GitHub* 🔗	0a2798a
Hybrid10bm.h5	*GitHub* 🔗	6495eea
Hybrid25bm.h5	*GitHub* 🔗	6495eea
Hybrid50bm.h5	*GitHub* 🔗	6495eea
Hybrid100bm.h5	*GitHub* 🔗	6495eea

APPENDIX C

CNN MOTIFS

The 33 motifs extracted from the first convolution filters of the CNNs and their best match (q-value < 0.05) with the JASPAR 2022 CORE non-redundant vertebrates motifs are shown. All 224 motifs for each of the CNNs are available on *GitHub* .

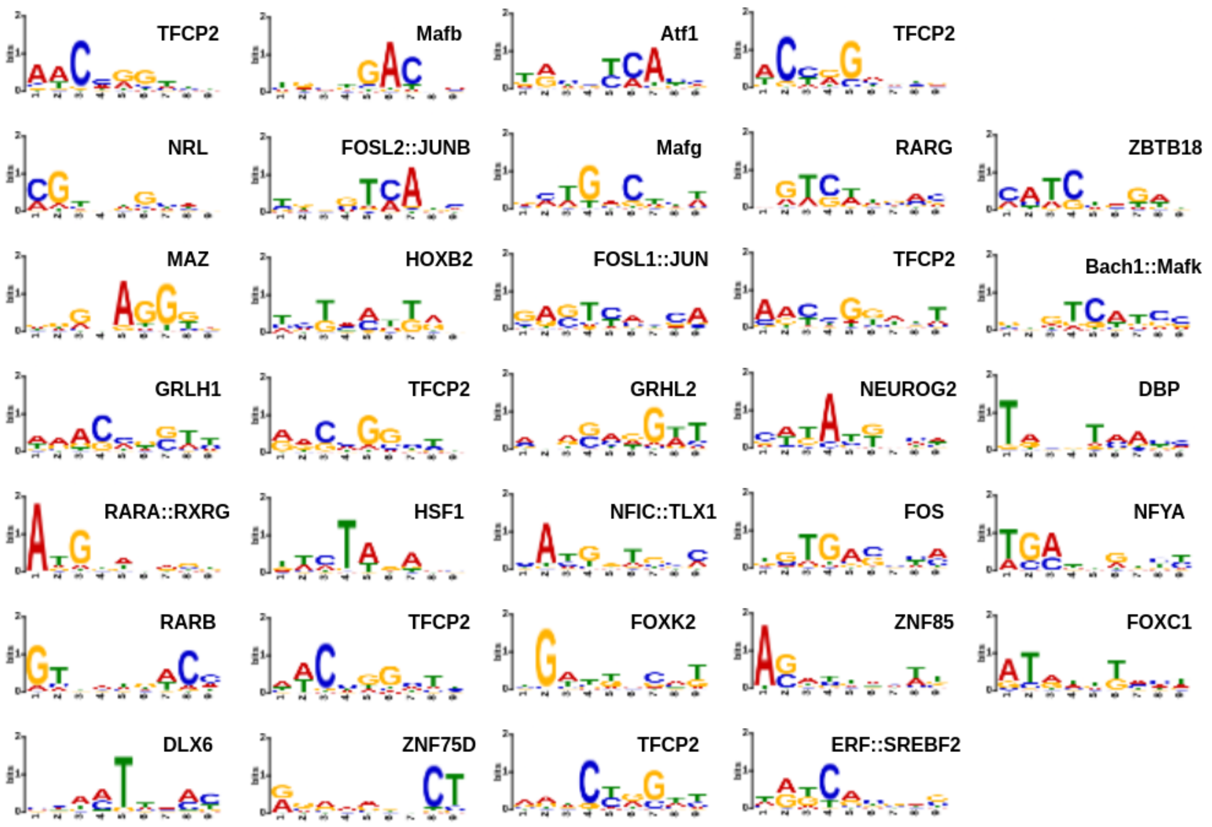


Figure C.1: Motifs from first convolution layer.

APPENDIX D
HYBRID CNN-LSTM MOTIFS

The 45 motifs extracted from the first convolution filters of the CNN-LSTMs and their best match (q-value < 0.05) with the JASPAR 2022 CORE non-redundant vertebrates motifs are shown. All 224 motifs for each of the CNN-LSTMs are available on *GitHub* .

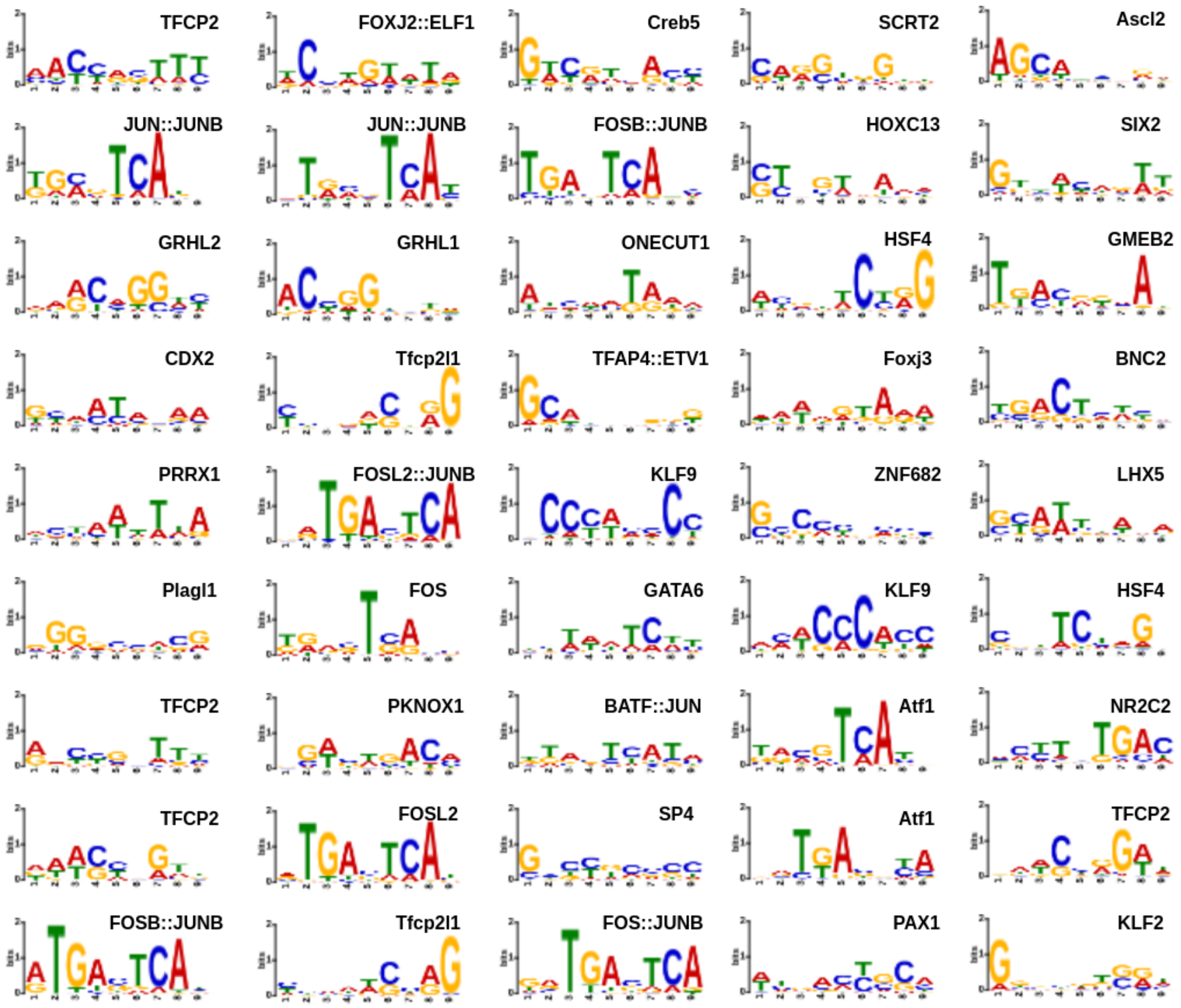


Figure D.1: Motifs from first convolution layer.