Explainable Deep Learning Models For Biological Sequence Classification

Dissertation zur Erlangung des Grades eines
Doktors der Naturwissenschaften (Dr. rer. nat.)
am Fachbereich Mathematik und Informatik
der Freien Universität Berlin

vorgelegt von
Stefan Budach

Dezember 2019

Erstgutachterin: Prof. Dr. Annalisa Marsico
Zweitgutachter: Prof. Dr. Christoph Lippert
Tag der Disputation: 11. Dezember 2020

Abstract

Biological sequences - DNA, RNA and proteins - orchestrate the behavior of all living cells and trying to understand the mechanisms that govern and regulate the interactions among these molecules has motivated biological research for many years. The introduction of experimental protocols that analyze such interactions on a genome- or transcriptome-wide scale has also established the usage of machine learning in our field to make sense of the vast amounts of generated data. Recently, deep learning, a branch of machine learning based on artificial neural networks, and especially convolutional neural networks (CNNs) were shown to deliver promising results for predictive tasks and automated feature extraction. However, the resulting models are often very complex and thus make model application and interpretation hard, but the possibility to interpret which features a model has learned from the data is crucial to understand and to explain new biological mechanisms.

This work therefore presents pysster, our open source software library that enables researchers to more easily train, apply and interpret CNNs on biological sequence data. We evaluate and implement different feature interpretation and visualization strategies and show that the flexibility of CNNs allows for the integration of additional data beyond pure sequences to improve the biological feature interpretability. We demonstrate this by building, among others, predictive models for transcription factor and RNA-binding protein binding sites and by supplementing these models with structural information in the form of DNA shape and RNA secondary structure. Features learned by models are then visualized as sequence and structure motifs together with information about motif locations and motif co-occurrence. By further analyzing an artificial data set containing implanted motifs we also illustrate how the hierarchical feature extraction process in a multi-layer deep neural network operates.

Finally, we present a larger biological application by predicting RNA-binding of proteins for transcripts for which experimental protein-RNA interaction data is not yet available. Here, the comprehensive interpretation options of CNNs made us aware of potential technical bias in the experimental eCLIP data (enhanced crosslinking and immunoprecipitation) that were used as a basis for the models. This allowed for subsequent tuning of the models and data to get more meaningful predictions in practice.

Acknowledgments

First and foremost, I would like to thank my supervisor Annalisa Marsico for giving me the opportunity to be part of the RNA Bioinformatics lab. Thank you for many interesting scientific discussions and advice and for granting me the freedom to pursue my own research ideas. I would also like to thank my office mates and lab colleagues Roman Schulte-Sasse, Lisa Barros de Andrade e Sousa, Sabrina Krakau, Evgenia Ntini and Brian Caffrey for making the PhD time an enjoyable one. My thanks further goes to Martin Vingron and the Computational Molecular Biology lab for many shared experiences over the years.

I wish to thank the IMPRS graduate program for funding and its scientific coordinators Kirsten Kelleher and Fabian Feutlinske for regular assistance and events. I am very grateful to the IT group of the MPIMG for providing excellent computing resources and for quick help when I crashed the GPU server yet again. Last but not least, I want to thank my family for their unending support and especially Jessica for continuous encouragement and for bearing with me during the final stages of the PhD.

Contents

1

Introduction

1.1 Explainable Machine Learning

Over the past years the field of machine learning has undergone a transformation from being a mostly research-focused discipline to seeing widespread adoption in all aspects of daily life. Especially through advancements in the branch of deep learning, machine learning is being popularized and used for product recommendation, language translation, image processing, credit assessment and many other things, often unbeknownst to human users. Deep learning describes a class of machine learning methods and algorithms based on artificial neural networks and has recently been applied very successfully to both supervised learning (learning based on labeled example data) and unsupervised learning tasks (learning based on unlabeled example data).

One well-studied and typical supervised problem is the task of detecting objects in images and the subsequent task of classifying the specific object, i.e. telling whether the object is a horse, a cat, a boat or something else. A human is able to perform this task by acquiring knowledge over time. By looking at many examples of each object and possibly through guidance of a more experienced human, one will eventually be able to tell different objects apart. Machine learning models are being trained in a similar way. Given labeled example images of every object category some mathematical model and algorithm can tweak itself over time to detect common patterns in the data which can be exploited to classify new, unseen images. Deep learning models have already been shown to outperform humans in such object classification tasks [1].

Which "patterns" differentiate a horse from a cat? An experienced human might mention the difference in body height or the mane of the horse, but what has a trained machine learning model learned in this regard? This problem of explaining why a machine learning model has made a certain decision is currently driving a highly active field of research. Consequently, it has already been shown that many models are not actually doing what a human would think they are doing, even when having a high classification accuracy. As an example, Figure 1.1 shows an image of a horse that was also correctly classified as containing a horse. It also shows, as a heat map, the result of an interpretation method highlighting in red color the parts of the image the machine learning model focused on to arrive at its classification. Surprisingly, the model seems to focus on the lower left corner of the image: a small copyright text. As it turns out this is the result of an oversight during the data collection as most training

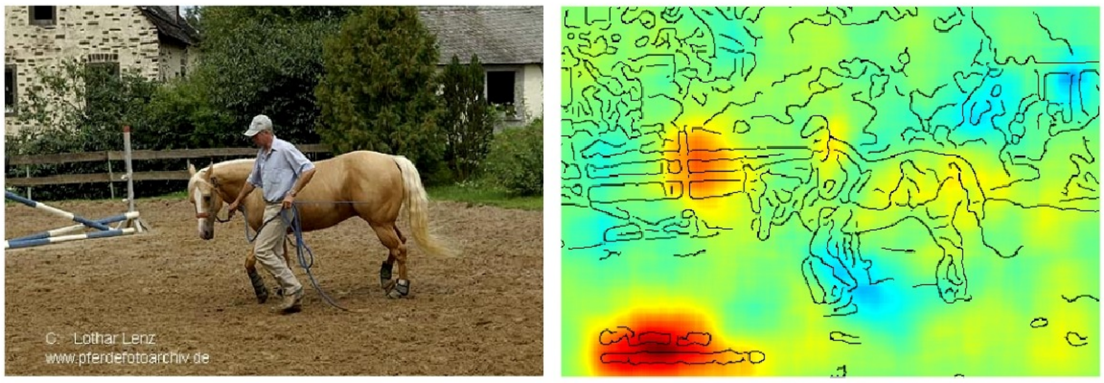


Figure 1.1: **Explaining Machine Learning Predictions.** A model that correctly classifies the left image as containing a horse is doing so by detecting a small copyright text not being present in other classes. The heat map shows the output of a method called layer-wise relevance propagation that highlights in red color which parts of an image are most relevant for the model prediction. The image is adapted from [2].

images of horses contained a copyright text while images of other classes did not [2]. Therefore, the model was right, but for the wrong reason.

Unintentional biases in data occur frequently and cause problems if not caught. Machine learning models will often pick up the "signal" resulting from these biases and are subsequently prone to deliver wrong predictions outside of the training context. In the mentioned example the error might be obvious in hindsight, but real-world data can become complex and unmanageable for humans. Likewise, the resulting impact of wrong predictions in this case might not be strong, but as machine learning predictions are increasingly used for decisions directly impacting human lives, e.g. in healthcare or finance, one has to be sure that predictions are right for the right reasons. To this effect, in another exemplary study [3], it was found that well-performing image models trained to detect the presence of pneumonia in chest X-rays were not able to generalize across multiple hospitals, because the models learned to focus on small details that were unique to a given hospital. Being aware of and fixing these issues before deploying models is crucial to avoid wrong diagnosis.

A final example shows that these issues are not restricted to image classification. Amazon uses machine learning for a variety of tasks, one such being the assessment of resumes of job applicants. After being leaked to the public that the used model discriminates against women [4], Amazon stopped using the tool for the time being. The used model was trained on data from previous years which inherently contained more male than female applicants and the model scores reflected this gender bias/imbalance. In this case the model raised ethical concerns. Methods to explain internal model properties can help to discover such issues more easily to allow tweaking if required.

In bioinformatics and in basic research stakes might not be that high. Nevertheless, because data biases are a common problem it is important to debug machine learning models during training to get reliable predictions. Biases often arise because of systematic technical limitations of experimental protocols (e.g. GC-content bias in RNA-seq data [5], hyper-ChIPable regions in ChIP-seq data [6], sequence-dependent base calling accuracy of Nanopore data [7]) and are ideally detected and managed during the data preprocessing. However, interpretable machine learning models can offer further sanity checks and act as bias detectors for new experimental protocols. Additionally, being able to explain why predictions (irrespective of bias) were made is essential to uncover new biological mechanisms and to advance knowledge. To this end, this thesis will focus on the interpretation of convolutional neural networks (CNNs), a class of deep learning models, for the classification of biological sequence data.

At the beginning of this PhD project, applications of deep learning within the bioinformatics domain were rare and their usefulness was not fully understood, due to their complexity and consequent "black box" nature. We contributed to the field by 1) developing a Python library that makes it easy for researchers to apply CNN models to sequence data without implementing them from scratch, 2) by exploring, evaluating and implementing different ways to interpret and visualize such networks through the usage of sequence motifs, structure motifs, motif enrichment and motif co-occurrence and 3) by applying networks to emerging research topics studying effects of DNA shape and modeling of experimental RNA-binding protein data. Especially with regard to RNA-binding proteins, insights into how proteins influence the RNA life cycle are still limited due to missing experimental data. Such data has only recently become available in larger quantities and we show that our CNN visualizations can help to better understand the role of individual proteins within the co- and post-transcriptional landscape. While experimental RNA-binding protein data in the form of eCLIP data is now becoming available for an increasing amount of proteins, it is only available for a very limited number of cell lines, but cell lines often differ substantially in which RNA transcripts are expressed. Thus, we also show that CNNs can be used to efficiently predict protein binding for transcripts for which experimental data is not yet available. Overall, extracting patterns and knowledge from data and predicting outcomes for unseen or experimentally unverified data are the two main purposes of machine learning, which we hope to illustrate during this thesis.

1.2 Thesis Outline

Chapter 2 introduces the biological background needed to understand the problems that machine learning is used for in this thesis. It explains how proteins interact with DNA and RNA, in the form of transcription factors and RNA-binding proteins, respectively, and highlights molecular properties that guide the process. Experimental methods that are able to detect interactions are presented as well. The data generated by these experiments will later form the basis for our machine learning models.

Chapter 3 introduces computational preliminaries, including the concepts of biological sequence classification and so-called sequence motifs that can be used as interpretation tools. Mathematical details of deep learning and CNNs are explained together with common interpretation methods.

Chapter 4 describes pysster, our Python software library for training and evaluation of CNNs, and shows how DNA-protein and RNA-protein interactions can be predicted by learning sequence and structure motifs with CNNs. Different ways of adding structure information for both DNA and RNA as model inputs are explored to improve interpretability and we visualize what every layer in a deep neural network is learning by adapting methods from Chapter 3 for our specific problem domain.

Chapter 5 presents a larger biological application by predicting RNA-binding of proteins for transcripts for which experimental data is not yet available. During this project the ability to visualize what exactly the models are learning helped to detect technical biases in the used eCLIP data, which otherwise might have gone unnoticed. Being aware of these biases allowed for subsequent tuning of the models and data to get more meaningful predictions.

Chapter 6 and *Chapter 7* conclude the thesis and briefly summarize machine learning-related contributions to other projects that were tackled during the time of the PhD, namely the interpretation of graph convolutional networks for cancer driver gene prediction and the dissection of microRNA regulation through modeling of expression quantitative trait loci.

2 Biological Preliminaries

Nucleobases and amino acids serve as the building blocks for all life on earth. When arranged in linear chains, these monomeric molecules form large polymers that are referred to as *biological sequences*: DNA (deoxyribonucleic acid), RNA (ribonucleic acid) and proteins. Together, DNA, RNA and protein molecules orchestrate the behavior of living cells. While the DNA sequence of multiple cells from the same organism is identical (ignoring mutations), the set of RNAs and proteins available at any given time is different and defines the specific function of cells.

2.1 Biological Sequences

The DNA sequence, or genome, of an organism is made up of just four different sequentially arranged nucleobases: adenine (A), guanine (G), cytosine (C) and thymine (T). When combined with a sugar molecule and a phosphate group, long strands can be created. In addition, A and T and C and G, respectively, are able to interact with each other by forming complementary base pairs through hydrogen bonds. Thus, the well-known double helix structure ("twisted ladder") is the result of two complementary single strands of DNA (see Figure 2.1). The human genome is comprised of about 3.2 billion base pairs and while it might not look like it, the order of the bases A, C, G and T is not random. Some small stretches of the genome (usually in the range of thousands or tens of thousands of base pairs) encode for genes. Genes are regions of the genome that form the blueprint for RNA. As DNA is a mostly static storage and located in the cell nucleus, RNA constitutes a more active form that can reach every region of a cell.

RNA is a single-stranded molecule and created through the process of transcription. During transcription, enzymes known as polymerases temporarily separate the double-stranded DNA helix within a gene region into individual strands. Subsequently, the polymerase uses one of the DNA strands as a template to synthesize an RNA copy out of bases that are complementary to the ones in the DNA. However, instead of T, its unmethylated form uracil (U) is used as the complementary base for A. The result of transcription is a primary transcript that can undergo several post-transcriptional (or co-transcriptional) modifications, one such being the process of splicing. Most genes are divided into alternating regions, so-called exons and introns, and intronic regions are usually spliced out of an RNA, i.e. they are removed from a transcript while the exons are joined together. Splicing is a very flexible process and it is also possible to skip individual exons, among other things [8], thereby creating (functionally) different transcripts from a single gene locus. In any case, the resulting transcript after splicing and other modifications is a mature transcript. One broadly differentiates two classes of transcripts: non-coding and coding transcripts. The majority of transcripts are non-coding, i.e. the RNA stage represents their final form and they function through interactions with other molecules. Based on transcript length and distinct functionality, non-coding RNAs are further divided into many sub-groups such as microRNAs ($\sim$ 22 bases short RNAs that repress other transcripts or mark transcripts for degradation [9]) and long non-coding RNAs (lncRNAs) (transcripts of length > 200 [10]). Especially lncRNAs are not well understood, as they are so far solely defined based on length, even though additional functional classification would be desirable and many researchers are currently working towards this goal.

Coding transcripts represent the other major transcript class and function as yet another blueprint, this time for proteins. Proteins are polymers composed out of chains of amino acids and some post-transcriptional modifications [11] and the presence of an open-reading frame lead to the translation of an RNA transcript into a protein. Thus, similar to how polymerases can create RNA from a DNA template, ribosomes can synthesize a protein from an RNA template. This process is called translation and the ribosome reads the bases of RNA as triplets, so-called codons, to assemble the proper amino acid chain (e.g. "ACG" encodes the amino acid threonine). A valid open-reading frame, a stretch of RNA initiated by a special start codon and terminated by a special stop codon, is required for successful translation. Resulting proteins have a variety of functions, one being the role of transcription factors.

2.2 Transcription Factors

Transcription factors (TFs) are proteins that are able to physically bind DNA to activate or repress the transcription of genes. They do not bind to the gene locus itself, but to regulatory regions such as promoters and enhancers nearby that are responsible for initiating transcription. TFs bound to such regions promote or block the recruitment and assembly of the polymerase complex. More than 2000 genes are estimated to encode TFs (around 10% of all protein-coding genes) and they can function alone or in combination with other factors. As TFs are essentially able to turn transcription of individual genes on and off, they are a major focus of gene regulation research [12].

2.2.1 *Protein-DNA Binding Affinity*

How does a TF know where exactly to bind within the 3.2 billion base pair long DNA? To recognize a specific binding site TFs possess at least on DNA-binding domain. Proteins fold into 3D structures containing individual domains and a DNA-binding domain is a structure that can recognize double- or single-stranded DNA. These domains are able to recognize very specific sequence patterns, for instance the well-studied TF TBP (TATA-binding protein) has a preference to bind "TATAAAA" sequences, the so-called TATA box often found in promoter regions [13]. However, DNA-binding domains are not limited to only one specific sequence pattern, they are able to recognize slightly similar sequences and in some TFs they do not have any preference and generally recognize DNA. Moreover, binding sites are generally short (6-12 base pairs on average) and are therefore expected to frequently occur just by chance within the DNA [14]. To circumvent this problem and to make binding and hence gene regulation more specific, TFs can have multiple DNA-binding domains and they often require the presence of other co-factors to collectively bind DNA. The accessibility of the DNA also plays a role, as the double helix structure is not always available for binding when it is wrapped around histone proteins. The complex formed of DNA and histone proteins is called chromatin and its purpose is the compaction of the otherwise very long DNA. DNA regions that are wrapped around histones are usually not accessible for TFs to bind, but epigenetic modifications of the histone proteins can cause unwrapping of specific DNA regions to allow for TF binding and subsequent gene transcription and represent another level of gene regulation (details can be found in [15] and [16]).

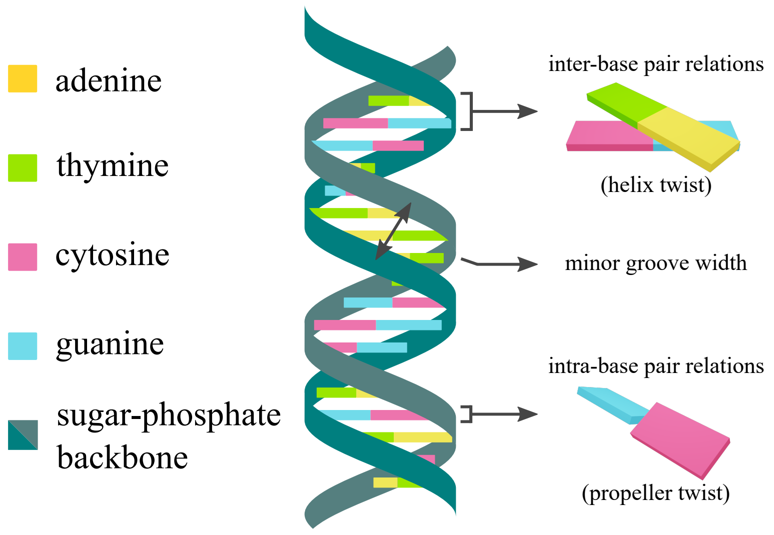


Figure 2.1: **The DNA Double Helix And Shape Features.** Two single strands of DNA form the by now familiar double helix structure. The local structure of the DNA can be characterized using so-called shape features which include the minor groove width (measured in angstrom) and multiple intra- and inter-base pair relationships (for instance helix twist and propeller twist, both measured in degrees). A total of 12 such relationships have been described previously [17].

Another aspect of protein-DNA binding affinity that recently started to attract more research is DNA shape [17, 18]. The shape of the DNA describes the local structure of the double helix and the relationship of bases and adjacent base pairs. DNA shape has been found to affect the binding in a similar way as the above mentioned sequence patterns do. Diverse properties and features are being used to describe the shape (see Figure 2.1) including minor groove width (distance between two opposing phosphates in the DNA backbone, measured in angstrom), propeller twist (amount to which bases within a base pair are rotated relative to each other, measured in degrees), helix twist (amount to which adjacent base pairs are rotated relative to each other, measured in degrees) and electrostatic potential [19] (how accessible is a mostly negatively charged stretch of DNA for the mostly positively charged amino acids that are known to directly interact with it, measured in volt). DNA shape features can be measured experimentally via X-ray crystallography and they can be derived computationally via Monte Carlo and molecular dynamics simulations [17]. Computational tools to predict shape features for a given DNA sequence are available and will be used later in the thesis [20].

2.2.2 *Experimental Detection of Protein-DNA Interactions*

A common experimental protocol to detect DNA binding sites for a protein of interest in vivo is ChIP-seq (chromatin immunoprecipitation sequencing) [21]. Following a genome-wide fixation of protein-DNA contacts through the use of formaldehyde, the DNA of cells can be fragmented and the fragments bound to a protein of interest can be immunoprecipitated (pulled down) by using a protein-specific antibody. The resulting DNA fragments can be isolated, sequenced and then computationally mapped to a reference genome to identify the approximate positions of binding sites. The computational analysis also involves the identification of significant binding sites (peak calling) by comparing the amount of mapped fragments at a given genomic position to a control experiment. These control experiments usually represent ChIP-seq runs missing the immunoprecipitation step ("input" control) or ChIP-seq runs using a non-specific antibody ("IgG" control) [22]. In Chapter 4 of this thesis, we will use already peak-called, public ChIP-seq data as the basis for TF binding site prediction models.

2.3 RNA-binding Proteins

Analogous to TFs binding DNA, RNA-binding proteins (RBPs) are proteins that are able to physically interact with RNA. In fact, some proteins are able to bind both DNA and RNA. RNAs are bound and regulated by RBPs throughout their entire life and RBPs can tune RNA functions by influencing splicing, RNA stability, cellular localization, degradation and other processes [23]. At the same time, RNAs can of course regulate the functions of bound RBPs as well [24]. RBPs and RNAs also often cooperate by forming large complexes comprised out of many different molecules such as the splicing and translation machinery [25]. About 1500 human RBPs are known [26], but until recently effective genome-wide experimental protocols to detect protein-RNA interactions have been missing, making RBPs relatively little understood compared to TFs.

2.3.1 *Protein-RNA Binding Affinity*

Binding of a protein to an RNA usually requires the presence of at least one RNA-binding domain, although it has recently been shown that some RBPs lack such a domain and can bind RNA through intrinsically disordered regions, among other things [24]. With about 3-5 bases in length the average RNA sequence pattern recognized by an RBP (if it has a preference at all) is even shorter than patterns recognized by TFs [26] and similar mechanisms are in place to make binding more specific, i.e. the presence of multiple RNA-binding domains, cooperative binding of multiple proteins, competitive binding and preference for certain RNA secondary structures.

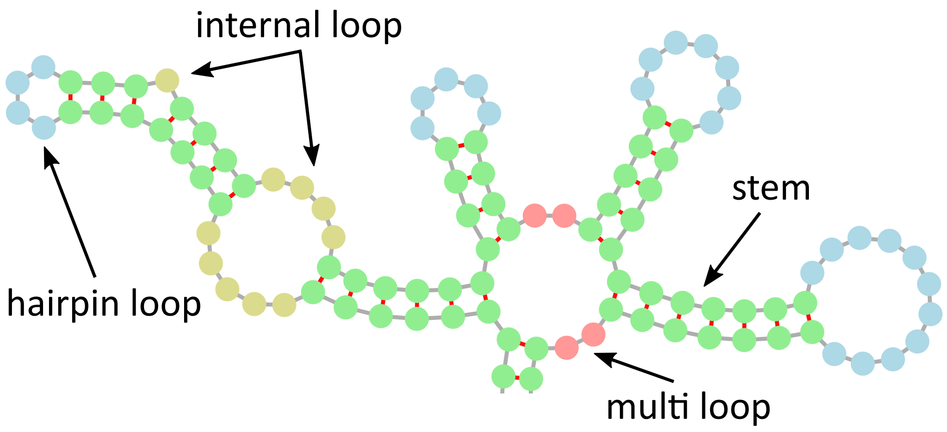


Figure 2.2: **RNA Secondary Structure Features.** Through intra-molecular base interactions RNAs form secondary structures. In the illustration circles represent bases, grey lines the RNA backbone and red lines hydrogen bonds between bases. The main structural features of RNAs are stems (continuously paired bases, shown in green), hairpin loops (unpaired bases connecting the two sides of a stem, shown in blue), internal loops (mismatches or small loops connecting two stems, shown in yellow) and multi loops (unpaired bases in loops with more than two outgoing stems, shown in red).

RNA is a single-stranded molecule, but the primary sequence of bases does not exist as a linear string in living cells. Rather, intra-molecular interactions between complementary bases lead to the formation of so-called secondary structures often visualized as 2D images (see Figure 2.2). The most important structural features that are recognized by different RBPs are stems (stretches of continuously paired bases), hairpin loops (small unpaired stretches connecting the two sides of a stem) and internal loops (small mismatches within an otherwise continuous stem). Secondary structures of RNAs are not static and change over time depending on cellular conditions and binding of molecules can even induce changes to the structure [27]. Genome-wide in vivo approaches measuring RNA secondary structures are feasible [28, 29], but they were only introduced recently and data are not yet available for a wide variety of cell lines and conditions. Because of this we are using computationally predicted secondary structures in the form of minimum free energy structures in later chapters. These predictions provide the theoretically most stable structure, but not necessarily the one an RNA adopts in reality at a given time [30].

2.3.2 *Experimental Detection of Protein-RNA Interactions*

By now, multiple methods exist that measure genome-wide RNA binding for a protein of interest in vivo. Most of them are based on the CLIP-seq protocol (crosslinking immunoprecipitation sequencing) [31] and for this thesis we will focus on one of its many variations called eCLIP (enhanced crosslinking and immunoprecipitation) [32]. The general idea of the protocol is similar to ChIP-seq and involves the crosslinking (fixation) of protein-RNA interactions through ultraviolet radiation, followed by RNA fragmentation, immunoprecipitation of the protein of interest, reverse transcription of isolated RNA into DNA and a final sequencing step. An input control experiment missing the immunoprecipitation is used to detect significant binding sites during the computational analysis of the data. The peak calling of both ChIP-seq and eCLIP data can be a very challenging task, due to potential biases and details of the protocols and the interested reader can learn more about the technicalities in [33] and [34]. Therefore, we will again use already processed, public CLIP data for our RBP models in Chapter 4 and Chapter 5, most of which was recently published as part of the ENCODE project [35] and represents the first large-scale profiling study of over 150 RBPs [36].

2.4 RNA A-to-I Editing

We briefly mentioned splicing as one of many post- and co-transcriptional modifications RNA sequences are subject to. Another such process is A-to-I editing: the act of converting an adenosine into an inosine (I). It is catalyzed by ADAR proteins (adenosine deaminase acting on RNA) which remove amino groups from A's turning them into I's [37]. In humans, ADAR RBPs possess a deaminase domain and at least one double-stranded RNA (dsRNA) binding domain, therefore editing frequently occurs in stem-rich regions of a transcript. With over 4.5 million events annotated in public databases, A-to-I editing is the most prevalent form of RNA editing [38], but its function is not fully understood. In all cellular processes, including RNA secondary structure formation and translation, I's behave as G's. Accordingly, editing is known to alter protein sequences if exonic regions of transcripts are affected [37]. This introduces yet another level of diversity and results in proteins not directly encoded in the original DNA gene locus. However, only 3% of editing happens in exonic regions. Editing overwhelmingly occurs in non-coding regions and, in addition, within so-called Alu repeats (89%) [39].

Alu sequences are repetitive elements specific to primates. They are ~ 300 base pairs long and over one million, almost identical copies can be found throughout the human genome [40]. Due to their repetitive nature, Alu elements form long regions of dsRNA, especially if two adjacent Alu's are transcribed together. Thus, they represent an ideal target for ADAR proteins. It has been observed that frequent editing of transcribed Alu's disrupts the double-stranded stem regions and thereby suppresses the interferon signalling pathway. This pathway initiates an immune response and is usually activated by the presence of long dsRNA, such as viral RNA. Consequently, preventing Alu-related auto-immune responses is thought to be one of the main functions of A-to-I editing [37, 41]. As edited A's are being read as G's also during normal sequencing, public databases with genomic locations of validated editing events are available [38] and later in this thesis, we will use an A-to-I editing data set to demonstrate the functionality of our Python CNN software library.

3 Computational Preliminaries

Sequence data generated by the ChIP and CLIP protocols mentioned in the previous chapter will form the basis for our deep learning models. In this chapter, we introduce mathematical details of convolutional neural networks and explain how they can be used together with this specific kind of input data. We also present multiple interpretation techniques for neural networks that can be used to detect so-called sequence motifs, the most common interpretation tool for the analysis of biological sequence data.

3.1 Sequence Motifs

For many years already, sequence motifs have been the tool of choice to draw conclusions from sequence data, as they inform about the binding preferences of proteins and as they can be used to estimate the impact of mutations that affect said binding preferences. The previously mentioned TFs and RBPs do not just bind a single specific sequence pattern, but they are able to recognize variations of a pattern. The RBP PUM2, for instance, is able to recognize the sequences "UGUAUAAU", "UGUAAAUA", "UGUACAUU" and many others (see Figure 3.1). All these patterns differ by one or more bases, but one can also see that some positions almost never change, while others vary frequently. This fact makes the use of a simple consensus sequence (a single "average" sequence showing the most common base for each position) often not suitable to summarize binding sites, as too much information is lost. Information about which positions are more stable, however, is important to estimate the impact of mutations, among other things. Consequently, sequence motifs (also called logos) [42, 43] have been proposed to better visualize a set of short sequences of equal length.

Sequence motifs allow for easy visual inspection of a set of sequences (Figure 3.1). Positions that vary frequently and therefore carry little information are depicted small (e.g. position 5 in Figure 3.1) and stable, informative positions are depicted larger (positions 1-3). To arrive at such a visual representation of a motif, the possible bases of a position are stacked on top of each other with heights proportional to their observed frequencies which are collected in a so-called position frequency matrix (a matrix of shape $m \, x \, n$ holding normalized base counts for every sequence position where m is the number of possible bases and n the length of the sequences, see Figure 3.1). The height of a complete stack is limited by the information content of a position. Given a set of equally-sized sequences, the amount of information $I(i)$ of position i, also binding sites

UGUAUAAU
UGUAAAUA
UGUACAUU
UGUGUAUA
UGUGAAUA
UGUACAUA
UGUAUAUA
UGUGAAUU

position frequency matrix

	1	2	3	4	5	6	7	8
A	0	0	0	0.63	0.37	1.0	0.13	0.63
C	0	0	0	0	0.25	0	0	0
G	0	1.0	0	0.37	0	0	0	0
U	1.0	0	1.0	0	0.38	0	0.87	0.37

sequence motif

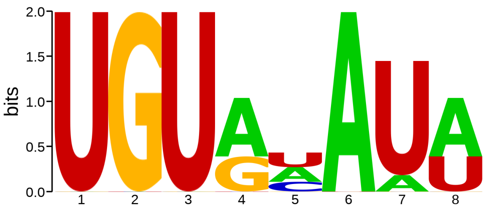


Figure 3.1: **A Sequence Motif Example.** The left part of the figure shows a non-exhaustive list of PUM2 binding site sequences. These sites can be summarized as a position frequency matrix by counting the occurrences of each base at each position and by dividing by the number of sequences. Based on the position frequency matrix a sequence motif can be computed with the help of Equation 3.1.

known as the Kullback-Leibler divergence and measured in bits, can be computed as follows [43]:

$$I(i) = -\sum_{b} f_{b,i} * log_2 \frac{f_{b,i}}{p_b} \tag{3.1}$$

where b represents the possible bases, $f_{b,i}$ is the observed frequency of base b at position i and p_b is the background frequency of base b. The background frequencies can either represent a uniform distribution of all possible bases (i.e. 0.25 in the case of the elemental 4-letter DNA and RNA alphabets) or they can be chosen differently to reflect prior knowledge about the origin of the used sequences (e.g. many promoter regions show higher G and C counts). The observed frequencies $f_{b,i}$, i.e. all entries of a position frequency matrix, are commonly artificially increased through small pseudocounts (and normalized again) to account for unobserved cases and to avoid frequency values of zero before computing the Kullback-Leibler divergence.

3.1.1 *Motif Finding And Sequence Classification*

Sequence patterns that can be used to compute a position frequency matrix are often not as neatly aligned as shown in Figure 3.1. Experimentally derived sequences that are assumed to share common motifs are usually long (hundreds of bases), but motif starting positions and the motifs themselves are unknown and therefore de novo motif finding tools have been developed that are able to discover overrepresented motifs in an unsupervised way. Many tools and algorithms have been proposed in the past [44], one of the most used ones being MEME [45] which discovers motifs by optimizing the information content of position frequency matrices using an expectation–maximization algorithm. Following the de novo discovery of motifs it is often of interest to computationally predict additional motif occurrences by using a position frequency matrix to scan sequences that were not experimentally covered. Tools such as RSAT [46] score each position of a sequence of interest based on similarity to a given motif. By also scoring background sequences a score threshold can be determined that enables the identification of statistically significant motif hits.

By connecting motif discovery and motif scanning into a pipeline it is certainly possible to perform a binary classification of sequences based on whether they contain a motif hit or not. However, if classification is the goal, for instance protein binding prediction for sequences not covered experimentally, dedicated supervised machine learning methods that blend feature extraction and scoring into a single model provide better predictive performance than the motif finding/scanning approach [47]. Additionally, machine learning methods can be applied to problems beyond binary decision making for example multi-class and multi-label classifications. To this end, different machine learning methods have been used in the past, such as string kernel support vector machines [48] and k-mer (short strings of length *k*) frequency approaches [49], as well as neural networks and CNNs [50] which will be the focus of the remaining thesis.

3.2 Artificial Neural Networks

Artificial neural networks (ANNs) are a class of machine learning methods and are loosely based on the nervous system of living beings. Similar to how neurons within the nervous system form networks to propagate and transform signals, ANNs are made out of connected nodes that are arranged in layers and propagate data from one layer to the next while applying nonlinear transformations. ANNs have already been proposed in the 1940s [51] and research has been dedicated to them ever since, but they have only recently seen widespread adoption. This is due to a strong increase in computing power (GPUs), the availability of much more training data and small but impactful changes to model details, such as random dropout of nodes and rectified linear unit activation functions, all of which made the backpropagation and gradient descent algorithms used for training of networks practically feasible.

3.2.1 *Multilayer Perceptrons*

The basic building block of ANNs are artificial neurons (Figure 3.2). Given n inputs $(x_1, ..., x_n)$ and n weights $(w_1, ..., w_n)$, a neuron computes the weighted sum of its inputs and applies a function f afterwards:

$$y = f(\sum_i^n w_i x_i) \tag{3.2}$$

Function f is called activation function and introduces non-linearity into the system. Accordingly, the output of a neuron is referred to as its activation. One possible activation function is the sigmoid, or logistic, function:

$$f(z) = \frac{1}{1 + e^{-z}} \tag{3.3}$$

which squashes the weighted sum into the $[0, 1]$ range and is especially popular for output neurons in the final layer of binary classifiers. Another popular activation is the rectified linear unit (ReLU) function:

$$f(z) = max(0, z) \tag{3.4}$$

which replaces all negative values with zero. ReLUs are one of the reasons why neural networks became popular again, because they introduce sparsity into networks (which helps to keep the often huge number of weights in a multilayer network under control), they are very fast to compute and because they lead to models with a high predictive performance [52].

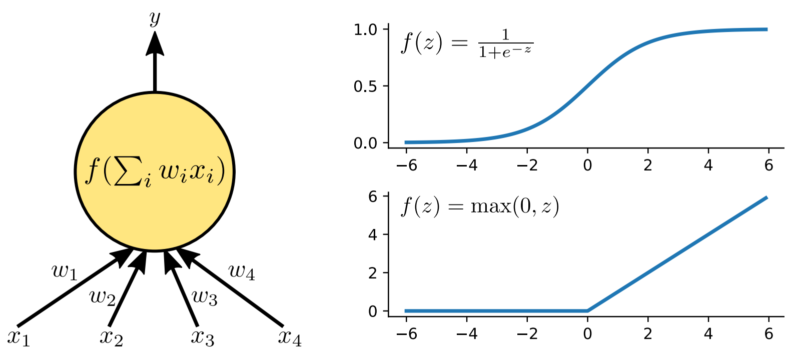


Figure 3.2: **Neurons And Activation Functions.** A neuron computes the weighted sum of its inputs and applies an activation function to introduce non-linearity into the computation. Frequently used activation functions are the sigmoid function (top plot) and the rectified linear unit function (bottom plot). The output y of a neuron is used as an input for neurons in the next layer of a multilayer network or as the overall network output, if that particular neuron is the final one.

Organizing multiple neurons into layers and connecting all neurons of a layer to all neurons in the subsequent layer leads to a so-called multilayer perceptron - the "default" neural network type (Figure 3.3). The output y of an individual neuron is thereby used as an input for all neurons in a subsequent layer. Each neuron, including the final output neuron of the network, performs the aforementioned computations, but the activation might differ between neurons. Today, ReLU functions are often used throughout the majority of a network [53], while the output neurons in the final layer use a function tailored to the specific classification task, e.g. the sigmoid function is appropriate for binary classifications. The individual layers of multilayer perceptrons are either called fully-connected layers or dense layers. Finally, yet another activation function frequently used for output layers containing more than one neuron is the softmax function [54]:

$$f(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \tag{3.5}$$

where z is a vector containing all outputs of a layer. This function is applied to every neuron output i individually and normalizes the output ensuring that all values are within the range $[0, 1]$ and form a probability distribution by summing up to one. This function is therefore useful for classification settings with more than two output classes (multi-class classification).

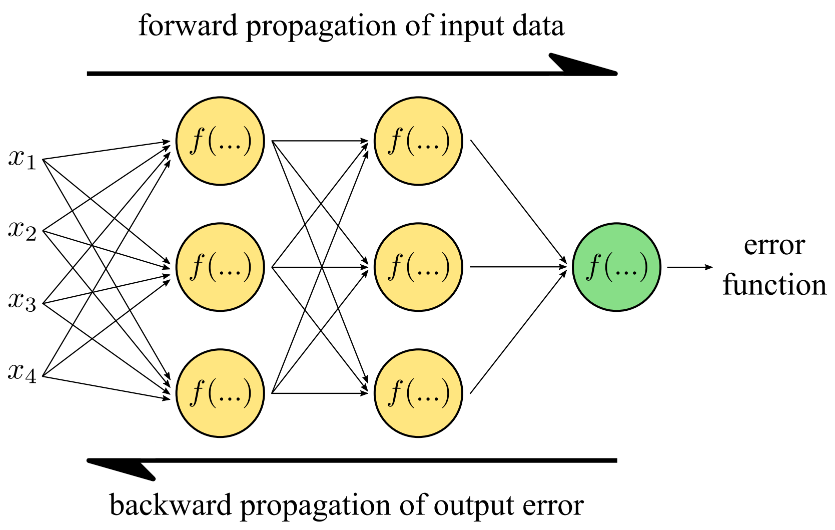

forward propagation of input data
x_1
x_2
x_3
x_4
$f(...)$
error function
backward propagation of output error

Figure 3.3: **A Multilayer Perceptron.** A "default" neural network is a collection of neurons arranged in layers in which every neuron passes its output as an input to all neurons in the subsequent layer. The example shows a 4-layer network consisting out of an input layer, two hidden layers (yellow neurons) and an output layer (green neuron). Each arrow connecting inputs/neurons in the visualization carries a different weight. Given some input data x, the weights are used to compute a forward pass through the network. Given the resulting network output and an expected output, the backpropagation algorithm (see Chapter 3.2.2) is used to walk backwards through the network to minimize the output error by adjusting the weights.

3.2.2 *Gradient Descent & Backpropagation*

In this thesis, neural networks are used for supervised learning tasks. Given a set of pairs (x, y) with $x \in X$ and $y \in Y$, the goal of supervised learning is to find a function $g : X \rightarrow Y$. In our case, x represents an input (e.g. a DNA sequence) and y the desired network output (e.g. a label indicating if the sequence is bound by a protein or not). If we would only have a single neuron in our "network", the function g is equal to Equation 3.2. Accordingly, in bigger networks, g would be a nested combination of Equation 3.2. *Learning* then means to find weights w that minimize the difference between the desired output y and the actual network output, denoted as $\hat{y}$ from now on. The similarity between y and $\hat{y}$ can be measured using a loss function (also called error or objective function). Many loss functions are possible, but in case of a binary classifier and if a sigmoid activation is used in the last neuron, the binary cross-entropy loss is a popular choice[1]:

$$L(y, \hat{y}) = -(y * log(\hat{y}) + (1 - y) * log(1 - \hat{y})) \tag{3.6}$$

where $\hat{y}$ is within the range $[0, 1]$ (because of the sigmoid activation) and y is either zero or one. The bigger the difference between these two variables, the higher the cross-entropy. One method to minimize such a loss function is gradient descent. Gradient descent involves the computation of the partial derivatives of L with respect to all individual weights w_i of a network, defined by the mentioned function g. The partial derivative of a function represents the slope of that function for a specific w_i. Thus, the derivative indicates how a change in w_i influences L. The gradient of a function is defined as a vector containing all partial derivatives and the gradient always points in a direction that increases the respective function [54]. Therefore, since our goal is the minimization of L, we want to adjust the network weights by moving in the direction of the negative gradient. This is called gradient descent and can be summarized as follows [54]:

$$w^* = w - \epsilon \nabla_w L(w) \tag{3.7}$$

where w is the original set of weights, w^* the adjusted set of weights, $\nabla_w L(w)$ the gradient and ϵ the learning rate, a hyperparameter (a parameter that has to be set by the user before the training) that controls how large the change of the weights should be.

Knowing all of this, backpropagation is a method to efficiently compute the gradient $\nabla_w L(w)$ in the above equation by decomposing the nested neural network function (which is composed out of many individual neuron computations) and recursively

1 `https://github.com/tensorflow/tensorflow/blob/r1.14/tensorflow/python/keras/backend.py`, accessed: 9.8.2019

employing the chain rule used for derivatives in calculus. It moves backwards through the network and computes the partial derivatives of neurons of higher-level layers first before it moves to previous layers, thereby propagating the loss throughout the network to reach every weight. Details of the chain rule application depend on the particular loss and activation functions being used and can be found in [54] and [55].

Taking everything together, a neural network is trained by an iterative process. Network weights are usually randomly initialized before computing the first forward pass through the network which leads to a first loss value. Due to the random initialization of weights this value will most likely be random as well. After updating the weights through backpropagation and gradient descent, another forward pass can be performed using the new weights. This should now lead to a lower loss value which one can try to minimize once again. The described process is then repeated until the loss function converges and it usually also makes use of mini-batch training. Instead of looking at all the data at once, the training data is shuffled, split into batches and fed batch-wise through the network, that is, a certain number of input samples will be forwarded through the network and their average gradient will be determined before a weight update is performed. A full training iteration is called epoch in neural network jargon and indicates that every batch has been seen by the network. For every training iteration, shuffling and splitting of the training data is repeated. Similar to the learning rate, the batch size is another hyperparameter of the model. Looking only at small batches of data has the advantage that gradients are noisier compared to the gradient of the complete data set which can help the model to better generalize to data other than the training data [56]. Nonetheless, smaller batches require more frequent weight updates and have a negative impact on the computational runtime performance and the batch size parameter therefore has a strong influence on the convergence of the training process.

In general, mini-batch training is a form of stochastic gradient descent (in contrast to "normal" gradient descent), as the gradient of random batches only approximates the real gradient of the complete data set. Nowadays, training of neural networks is almost exclusively performed by backpropagation and gradient descent combined with mini-batch training. The main disadvantage of the procedure is its tendency to only find local minima of loss functions (due to those being non-convex) which is, however, not a frequent problem in practice and for large networks [57].

3.3 Convolutional Neural Networks

Dense multilayer perceptrons are very performant classifiers, but they are not a good fit for the sequential DNA and RNA string data we are working with. One problem is the number of weights that would be needed to model biological sequence data. For instance, to use a DNA sequence of length 200, one might treat each position as a distinct network input (i.e. distinct x_i in the aforementioned explanations). Assuming that we want to use a network with three hidden dense layers and 100 neurons per layer, we would need $20,000$ weights to connect the input to the first layer, $10,000$ to connect the first to the second layer, $10,000$ for layer two and three as well and 100 weights for the final connection to the output neuron. However, having a large number of trainable weights negatively affects the runtime performance, requires a substantial amount of training data and is in turn prone to overfitting. Another problem, and arguably a bigger one, is the fact that a multilayer perceptron treats each input x_i independently. This means, that we are losing valuable information about the relation of adjacent bases that form sequence motifs.

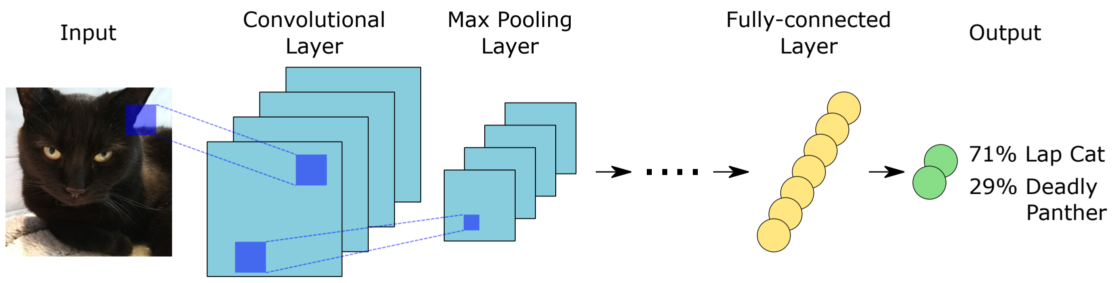


Figure 3.4: **Standard Convolutional Neural Network Architecture.** Originally, CNNs have been invented for the task of image classification. Receptive fields within the convolutional and pooling layers are able to detect small patterns in the input and by hierarchically combining the learned patterns, CNNs can classify images with high accuracy.

For sequential data, CNNs are a viable alternative, as they drastically reduce the number of weights and because they capture the dependencies between adjacent input positions by automatically learning sequence motifs (or something that we interpret that way) to classify the given data. The basic architecture of a CNN consists of a variable number of convolutional and max pooling layers followed by a variable number of fully-connected layers. As we will see later in the thesis, the convolutional portion of the network is responsible for automatically extracting features from the input data (in our case sequence motifs) while the fully-connected layers are responsible for classifying the data based on the learned features. CNNs have been and continue to be a very successful architecture for different types of problems, first and foremost image classification (Figure 3.4). Conceptually, CNNs are based on the visual cortex of living beings in that they consist of multiple receptive fields that learn to recognize small patterns. Combining many receptive fields in a hierarchical fashion enables the detection of larger patterns and eventually allows for the complete understanding of the image or scenery that is being looked at [58, 59]. For image classification, object detection and other computer vision tasks, CNNs have become the de facto standard [60]. In the following, we will explain the details of CNNs and their application to sequential string data, as this is the data we are interested in.

3.3.1 *Convolutional Layers & Pooling*

Before one can apply a CNN to biological sequence data, the data has to be transformed into a numeric representation. Therefore, DNA (or RNA) sequences are usually one-hot encoded, that is, each position in a sequence is treated as a categorical variable of size n and represented by a vector of length n containing exactly 1 one and zeros otherwise (Figure 3.5). Vectors of all positions are then concatenated into a single matrix. In the case of a DNA sequence of length 100 the result is a 4×100 matrix. The one-hot encoded matrix is equal to a position frequency matrix that is based on a single sequence.

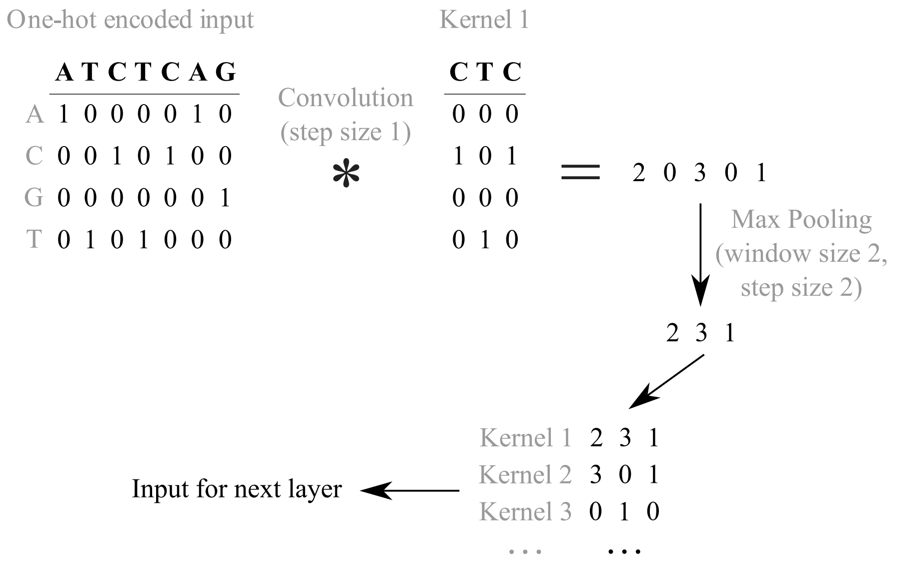


Figure 3.5: **Convolution And Max Pooling.** To obtain numeric input suitable for a CNN, input strings are one-hot encoded ("ATCTCAG" in the example). Kernel entries are usually randomly initialized with real-valued numbers, but for the sake of the example kernel 1 shows a perfect "CTC" kernel. Kernels scan a sequence using a sliding window approach by computing the sum of the element-wise products at each step and the vector $(2, 0, 3, 0, 1)$ is the result of applying Equation 3.8 to the example. Applying max pooling with window size two and step size two further leads to the vector $(2, 3, 1)$. As layers consist of multiple kernels, the output of every kernel within the same layer is summarized into a single matrix which is used as an input for the subsequent network layer.

As opposed to fully-connected layers, convolutional layers are not a collection of neurons, but a collection of so-called convolutional kernels that take on the duty of receptive fields. A kernel is a matrix and acts as a feature detector. In our case, a kernel is also comparable to a position frequency matrix in that each row represents a possible base and each column a position of the motif. The number of columns of a kernel, and therefore the length of a motif, is a hyperparameter. For the sake of the example, the kernel in Figure 3.5 shows only perfect zero/one entries. In practice, the entries of kernels represent the network weights w, i.e. kernels are randomly initialized with real-valued numbers and hopefully converge to something meaningful at the end of the backpropagation training. In a convolutional layer there are no weights connecting kernels to every input and a kernel operates by moving over the input in a sliding window fashion ("convolving") and by computing the sum of the element-wise products (dot product) between itself and the input window at each input position i:

$$c_i = f(\sum_k \sum_l A_{kl} * B_{kl}) \tag{3.8}$$

where A and B are matrices representing an input window and a kernel, respectively, where k and l are row and column indices, respectively, and where f and c are the activation function (e.g. ReLU) and activation vector, respectively.

The step size of the convolution approach is usually one, resulting in overlapping input windows (e.g. an input of length seven and a kernel of length three produce an output vector c of length five, see Figure 3.5). Kernels in convolutional layers make use of so-called parameter sharing by using the same weights regardless of the current input window. This additionally differentiates kernels from neurons in fully-connected layers which use one weight per input position. By sharing a fixed set of weights across the input sequence, kernels can detect a pattern multiple times and independent of its location in the input sequence. This leads to the effect that CNNs are equivariant to translation [54], i.e., if patterns in an input sequence shift or change, the kernel output vector c shifts or changes in the same way.

Before the resulting vector c is passed as an input to the next part of the network, a pooling operation is performed. A popular pooling method is max pooling, another sliding window approach, which, given a window and a step size, slides over c and only keeps track of the maximum value in each window, thereby effectively reducing the size of c (see Figure 3.5). Pooling down-samples the data to improve the runtime performance of the network and to decrease the amount of needed weights in the fully-connected layers at the end of the network. Since high values in vector c imply high similarity of the kernel to a particular input position, only keeping track of the maximum value and deleting smaller values also helps the network to generalize to different data by decoupling a kernel from an exact input position. Thus, the model becomes "approximately invariant to small translations of the input" [54], that is, small changes in the input likely do not affect the output vector of the max pooling operation.

The number of kernels in a convolutional layer is another hyperparameter. Kernels (including pooling) operate independently of each other and the output vectors of all kernels are concatenated to form a matrix C of size $m \times n$ where m is the number of kernels and n the length of vectors c after pooling (Figure 3.5). This matrix is used as the input for the next network layer, which is often another convolutional layer. Conceptually, this matrix is similar to the original one-hot encoded input matrix, but now rows represent kernels/full motifs (in contrast to individual bases) and while columns still represent input positions, there is no direct mapping to the original input possible anymore, because of the max pooling (e.g. position $C_{1,1}$ and position $C_{2,1}$ might not refer to the same position in the original input, depending on the results of the max pooling operation). As a side note, the form of convolution applied to sequential string data is often referred to as 1D convolution, whereas image classification uses 2D convolution. 2D convolution performs the exact same computations presented here, but the input of image classifiers is usually interpreted as a 2D matrix (containing individual pixels) and kernels and pooling operations do not just slide over the input from the left to the right (as shown in our example in Figure 3.5), but additionally from the top to the bottom such that the result is another 2D matrix that can be interpreted as an image.

3.3.2 *A Complete Convolutional Network*

The convolutional/pooling layer that is directly connected to the input is usually followed by at least one additional convolutional/pooling layer. Each layer operates on the output of the previous one and has its own kernels and hyperparameters, but they all perform the same operations, thereby learning bigger and bigger motifs over time. Every pooling layer also increases the effect of the mentioned translational invariance. At some point, the output matrix of a convolutional/pooling layer is flattened (row- or column-wise) and fed into a fully-connected layer which performs the final classification task (Figure 3.6). In Chapter 4, we will visualize what every kernel and neuron in such a CNN is learning.

The number of weights in the convolutional part of a CNN is independent of the length of the input sequences. Weights depend on user-controllable hyperparameters such as number of kernels, length of kernels and pooling window size and step size. The number of weights in the first fully-connected layer partly depends on the length of the input sequence, however, convolutional/pooling layers substantially reduce that length by essentially compressing sequences. For instance, applying a kernel of length 10 to an input of length 200 results in an output vector of length 191 (200-10+1) and applying a typical pooling operation of window size two and step size two halves

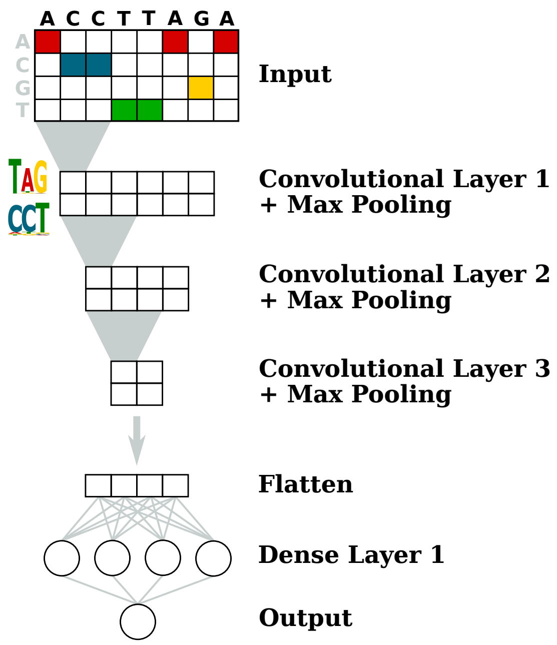


Figure 3.6: **A Convolutional Neural Network For Sequence Classification.** A basic CNN consists of a variable number of convolutional layers augmented through max pooling layers (here exemplarily using two kernels of length three in the first layer) followed by a variable number of dense layers. Convolutional layers are able to hierarchically extract features from the data and the dense layers are able to perform the desired classification task utilizing these features.

that number to 96. Each additional convolutional/pooling layer in the network further reduces the length until eventually the first fully-connected layer is reached. Because of the length reduction and because pooling helps the network to only focus on the important parts of the input, a single, small fully-connected layer at the end is often enough to perform the classification task, resulting in the fact that CNNs need less weights than comparable multilayer perceptrons.

3.3.3 *Network Regularization*

Even though the number of trainable weights in a CNN is comparably low, there is still a considerable amount left and the network still has to be regularized to avoid overfitting and to improve the convergence of the backpropagation algorithm. The main regularization method is dropout [61]. The idea of dropout is to randomly and temporarily remove kernels and neurons from the network during training. Different randomly selected kernels and neurons are being removed in each training iteration, thus, not all parts of the network are being adjusted every time. It is also possible to apply dropout to the network input, e.g. by randomly setting a small amount of columns in the one-hot encoded input to zero. This prevents the network from focusing too much on specific patterns in the data. Dropout adds a form of random noise to both the data and the network itself, making it more robust by intuitively averaging over many slightly different network architectures. Dropout is only used during training and not for predicting new data. It is applied to individual layers and in small amounts, e.g. one might randomly turn off 20% of units in a specific layer.

Another regularization method is early stopping [62]. Here, the idea is to stop the backpropagation when the loss of a validation data set has stopped improving for a certain number of iterations. During training of a network, it is good practice to split the available data into a training and a validation set. The network is only trained on the training set and the validation set can be used to decide on when to stop training. By monitoring the loss on both the training set and the validation set, it is often observable that at some point the validation loss starts to plateau or even increases again while the training loss still improves (Figure 3.7). This indicates the beginning of overfitting and can be used to dynamically stop training to preserve the generalizability of a network.

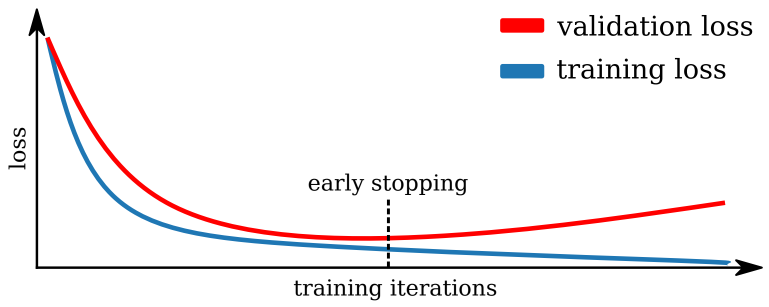


Figure 3.7: **Early Stopping.** The loss of the training and validation data sets will often start to diverge at some point during network training. To avoid overfitting to the training data, the training can be stopped at that point. The number of training iterations needed to determine when exactly to stop is yet another hyperparameter and usually reflects the number of iterations without validation loss improvement.

One last exemplary regularization approach is the restriction of network weights. In later chapters of the thesis, we will be using a max-norm regularization [61], that is, given a vector w representing the weights associated to a kernel or neuron, the network will be optimized such that $||w||_2 \leq d$ holds. d is a hyperparameter and prevents individual weights from becoming too large and more evenly distributes the available information in the input data over all weights to make better use of the full model capacity.

3.3.4 *Feature Visualization Methods*

Neural networks automatically extract features from the input data that are indicative of the classification problem at hand. If the goal of network training is not just a high predictive performance, but also to gain new insights into a problem, feature visualization methods can be applied to extract learned features from a trained network model. Moreover, visualization methods can be helpful for debugging by looking for (unintentional) biases in the data. A hint in that direction can be a model that is learning features which a human with strong domain knowledge would not expect to be learned or which they would not expect to be in the data in the first place. Feature visualization helps to detect such cases and therefore improves the fine-tuning of data or model.

Specifically for biological sequence data, the DeepBind paper [50] was the first to introduce a method for visualizing sequence motifs learned by kernels from the first convolutional layer of a CNN. While kernel matrices have the shape of a position frequency matrix, their entries are more or less arbitrary. A high value within a column of a kernel indicates that the respective base is the most likely one, but there are no restrictions ensuring that kernel entries are always positive or that individual columns form a probability distribution by summing to one (as is the case in position frequency matrices). The DeepBind authors proposed to visualize a kernel by extracting subsequences from the training data that lead to high kernel activations. Given an input sequence s and a kernel k with corresponding output vector c (as defined by Equation 3.8 and before pooling has been applied), the objective is to find the position of the maximum value in c:

$$j = \underset{i}{\operatorname{argmax}}\, c_i \tag{3.9}$$

Position j represents the part of input s that is most similar to the kernel matrix and if c_j is above a chosen threshold, the subsequence $s_{j \dots j+m-1}$ can be extracted from the input, where m is the length of kernel k. The procedure is repeated for all input sequences and yields a set of equally-sized subsequences that can be used to compute a sequence motif in the usual way.

Unfortunately, the above visualization method can only be applied to kernels of the very first convolutional layer. The pooling layers destroy the direct connection between input sequence and kernel output, therefore it is not possible anymore to map the maximum value of a kernel output of downstream layers to a corresponding position in an input sequence. One possible method to visualize kernels (and also neurons) in every layer of a network is called visualization by optimization [63]. Instead of computing a summary statistic over the input as the above method, visualization by optimization generates a single input sequence that maximizes the output of a kernel or neuron. During training of a neural network we usually start with a fixed input and randomized network weights to minimize the network loss function with respect to the weights. After the successful training, when fixed network weights have been obtained, we can accordingly start with a randomized input to maximize the output of a specific kernel or neuron anywhere within the network with respect to the input, that is, entries of a randomly initialized input matrix now depict the weights w we want to learn. This allows to visualize what every unit in a network is reacting to. Analogous to the network training, the optimization is an iterative process involving backpropagation and, since we want to maximize an objective, gradient ascent:

$$w^* = w + \epsilon \nabla_w L(w) \tag{3.10}$$

where w represents a set of weights, w^* the adjusted set of weights, $\nabla_w L(w)$ the gradient and ϵ a learning rate. Instead of moving in the negative direction of the gradient as is the case during gradient descent, we are simply going with the gradient. Many variations of this method have been applied to image data (see [63] for an overview) and in Chapter 4 we will adopt it for biological sequence data.

3.3.5 *Attribution Methods*

The two methods explained in the previous subsection are model-centric approaches that try to give insights into the inner workings of a network by explaining individual network units. This thesis focuses on model-centric methods. In addition to explaining internal network properties, it can, of course, also be of interest to explain why a specific input has been classified in a certain way. Input-centric approaches, also called attribution methods, try to tackle this problem. In this thesis, we will not apply attribution methods, but for the sake of completeness we present their general idea.

Attribution methods explain which features of a specific input were most relevant for the network prediction. For a given DNA sequence, these features would be its individual positions. Multiple methods, such as saliency maps [64], DeepLIFT [65] and layer-wise relevance propagation (LRP) [66] exist. They all try to achieve a similar goal, but use different mathematical means to do so. LRP, for instance, computes a backward pass through a network and redistributes the network output $\hat{y}$ over the input such that:

$$R_i^{(l)} = \sum_j \frac{c_i w_{ij}}{\sum_i c_i w_{ij}} R_j^{(l+1)} \tag{3.11}$$

where R represents the relevance of unit i and j in layer l and $l + 1$, respectively, and where c is the output of unit i and w_{ij} the weight connecting unit i and j. Given that the relevance of the output neuron is $\hat{y}$, this assures that:

$$\sum_d R_d^{(1)} = \hat{y} \tag{3.12}$$

is satisfied for the input layer (layer 1). Relevance values obtained that way can be both positive or negative and indicate if the presence of the feature had a positive or negative impact on the prediction. Example applications of input-centric interpretation methods on biological sequence data can be found in [50] and [67].

3.3.6 *CNNs For Biological Sequence Analysis*

Deep learning and especially CNNs have already been successfully applied in bioinformatics research and are now widely used to tackle various questions. This subsection gives a brief overview of notable publications published in the previous years and should also provide some insight into why we decided to write our own CNN software library in Chapter 4.

To the best of our knowledge, DeepBind (2015) [50] represents the application of CNNs for biological sequence data that popularized their usage in our field. In this paper, the authors showed that the standard CNN architecture that has been established for computer vision tasks can also be applied to sequence data. By modeling ChIP-seq and other experimental data, the presented CNN models were able to automatically learn sequence motifs without the need for time-consuming feature engineering and their predictive power outperformed most methods previously presented as part of the DREAM5 challenge for predicting protein binding sites [68]. As described in Section 3.3.4, the DeepBind paper also introduced a way to visualize first-layer kernels as sequence motifs. In addition, it introduced "mutation maps" as an interpretation option for neural networks. By performing perturbation tests (e.g. systematically changing individual bases within a sequence) and by comparing the network predictions for wildtype and perturbed sequences, the authors were able to deduce the importance of every sequence position. Trained models and visualizations are available online and the source code of DeepBind is available as well, however, it is undocumented and not usable without significant time investment. We actually asked the authors about how to use DeepBind for our own data and were told that it would be easier to implement everything from scratch. In support of the authors, software libraries like Tensorflow [69] were not yet available when DeepBind was published. Since CNNs require GPUs as hardware for effective training, which are cumbersome to program for, it was not an easy task to provide software that runs on a variety of hardware. Today, Tensorflow and other libraries abstract these hardware requirements away and have substantially simplified the distribution of related software.

Basset (2016) [70] is another CNN library and was explicitly designed for training models on one's own data. It makes use of the standard CNN architecture and in their paper, the authors presented the prediction of DNA accessibility (open versus closed chromatin) from DNase-seq data as an example application. With regard to model interpretation they offer the visualization of first-layer kernels as sequence motifs and the option to perform mutation map analysis. One short-coming of Basset is its restriction to sequence data, as no other additional input data can be used and while model hyperparameters can be changed, no built-in hyperparameter search or optimization is available.

Deep learning models are very flexible, because different kinds of layers and units can be freely combined and DeepCpG (2017) [71] makes use of this point. The goal of DeepCpG is the detection of cytosines possessing an additional, temporary methyl group (such cytosines are frequently followed by a guanine and therefore abbreviated as "CpG"). Methylated cytosines depict a form of genome and transcription regulation, as, for example, heavily methylated promoter regions repress transcriptional activity. In their work, the authors present a multimodal model, i.e., a model that uses separate inputs that are being joined together at some point. The DeepCpG network consists of two modules: a convolutional neural network block processing DNA sequence information and a recurrent neural network block that processes single-cell methylation profiling data based on scBS-seq and scRRBS-seq. Specifically, the latter block performs an embedding operation to model CpG dependencies within individual cells and is followed by recurrent layers to model dependencies between different cells. Recurrent layers are yet another type of layer (in addition to dense and convolutional layers) and able to learn long-range dependencies in data, much more so than convolutional layers whose kernels are restricted to a defined range of adjacent input positions. The interested reader can learn more about recurrent layers in [54]. At some point, DeepCpG concatenates the outputs of both network blocks and feeds them into two dense layers to predict the methylation state of cytosines in individual cells. The final model was shown to outperform previous methods and through the visualization of first-layer kernels as sequence motifs, the authors could learn which proteins are associated with elevated methylation levels and general methylation variability. Software has been made available to train models using one's own single-cell methylation data.

Finally, iDeepS (2018) [72] represents a deep learning model that also incorporates RNA secondary structure data and is able to learn both sequence and structure motifs to predict RBP binding sites. The iDeepS authors also follow a multimodal approach and train two convolutional blocks in parallel, where one block is applied to sequences based on CLIP-seq data and where a second convolutional block is applied to the respective secondary structure predictions. The outputs of both convolutional blocks are merged at some point to apply an additional recurrent layer followed by a dense layer for classification. First-layer kernels of both convolutional blocks can be individually visualized as motifs, however, it is not possible to tell which sequence and structure motifs belong together, as the convolutional blocks processed the data separately. Additionally, while the authors provide source code to reproduce the results and to train models on other data, its functionality is very limited and it is not possible to adjust hyperparameters (number of layers, number of kernels, length of kernels, etc.), as the described network architecture is fixed.

3.4 Performance Measurements

To conclude the chapter, we will explain how the predictive performance of supervised classification models can be evaluated. In a binary classification setting, the classes are generally referred to as positive class and negative class (or background) and the last-layer outputs of our CNNs can be interpreted as the probability that a given input sample belongs to a certain class. Directly working with predicted probabilities has the advantage that these values carry information about the confidence of the prediction, but if the goal is to further assign discrete class labels a probability threshold is required. Given a threshold (e.g. 0.5 such that inputs above 0.5 are classified as positive and inputs below or equal to 0.5 as negative) and a collection of inputs to predict (e.g. held-out test data not used during training to avoid overfitting), one can compute a so-called confusion matrix, a 2×2 matrix that summarizes the predictive performance by reporting the true positives (TP, number of positive samples predicted to be positive), false positives (FP, number of negative samples predicted to be positive), false negatives (FN, number of positive samples predicted to be negative) and true negatives (TN, number of negative samples predicted to be negative) (see Table 3.1).

		True Labels	
		Positive	Negative
Predicted Labels	Positive	TP	FP
	Negative	FN	TN

Table 3.1: **Confusion Matrix.** Count data describing true positives (TP), false positives (FP), false negatives (FN) and true negatives (TN) can be used to assess the performance of a binary classifier for a specific probability threshold.

Based on the confusion matrix, additional measurements can be defined. Frequently used are the true positive rate (TPR, portion of correctly predicted positive samples) [73]

$$TPR = \frac{TP}{TP + FN} \quad , \tag{3.13}$$

the false positive rate (FPR, portion of negative samples incorrectly predicted as positive) [73]

$$FPR = \frac{FP}{FP + TN} \tag{3.14}$$

and the precision (portion of positive predictions that are truly positive) [73]

$$Precision = \frac{TP}{TP + FP} \quad . \tag{3.15}$$

To get a visual overview of the performance, so-called receiver operator characteristic (ROC) curves can be constructed by plotting the TPR (y-axis) against the FPR (x-axis) at different probability thresholds. The ideal ROC curve has high TPR and low FPR, i.e. the curve should tend towards the top-left corner of the plot. ROC curves can further be condensed into a single number by computing the area under the ROC curve (auROC), a value between 0 and 1 where higher values indicate better performance across a range of thresholds. A classifier that randomly assigns class labels creates a straight line across the main diagonal in ROC plots with a respective auROC value of 0.5 and can be considered as a baseline [74].

Finally, if classes are highly imbalanced (much more samples are available for one of the classes), precision-recall plots are preferred over ROC plots by plotting the precision (y-axis) against the TPR, also called recall (x-axis). ROC curves are insensitive to the class balance and can be misleading in such cases [73, 74], because TPR and FPR metrics evaluate the classes separately. Precision, however, takes predictions for both classes into account. In practice, this means that a model that is evaluated using both balanced and imbalanced held-out test data will show near-identical ROC curves, but different precision-recall curves [73, 74]. Generally, the ideal precision-recall curve has high precision and high TPR, i.e. the curve should tend towards the top-right corner. The area under the precision-recall curve (auPRE) can be computed analogous to the auROC and the random baseline performance is given by a horizontal line that varies according to the portion of the positive class (e.g. if 10% of samples are positive, the baseline is at y = 0.1)

4 Learning Sequence And Structure Motifs with CNNs

Neural network-based classification methods have increasingly been shown to outperform traditional machine learning methods, such as random forests and support vector machines, for the analysis of biological sequence data [70, 71]. However, when we started to work with neural networks and especially CNNs, there was no easy way to use these methods for one's own data. Existing implementations were either hard to reuse [50], focused on specific problems [71] or lacked comprehensive data integration and interpretation options [70, 72]. To this end, we developed *pysster* (a *PY*thon *S*equence-*ST*ructure classifi*ER*) [75], an open-source Python library for training and interpretation of CNNs on biological sequence data and this chapter demonstrates the main features of our library. While most of the upcoming sections focus heavily on network visualization and biological interpretation aspects, predictive performance is, of course, equally important and will be discussed in more detail at the end in Section 4.4 and Chapter 5.

4.1 Pysster

Pysster was designed to enable more researchers to easily apply, tune and evaluate CNNs on biological sequences using only a handful of lines of code and this chapter describes and showcases its features by modelling ChIP-seq, CLIP-seq, A-to-I editing and artificial data. Before showcasing specific examples, however, this subsection takes a bird's eye view of the package features and some implementation details.

4.1.1 *Network Architecture & Feature Overview*

THE BASE MODEL. Pysster's basic network architecture mirrors the established CNN architecture described previously: a variable number of convolutional/max pooling layers are followed by a variable number of dense layers. All hidden layers, except the output layer, use ReLU activations to achieve a high predictive performance. The network architecture and individual layers can be freely adjusted using a collection of hyperparameters, such as number of layers, number of kernels/neurons, length of kernels, mini-batch size and regularization parameters. A full list of hyperparameters and usage considerations can be found in Section A.1 and Section A.2, respectively. These appendix sections also list default parameters that we found to perform well, both in terms of predictive performance and training convergence, for a variety of tasks.

To simplify the identification of the most suitable hyperparameters for a given data set, grid search tuning has been implemented, i.e. given a set of hyperparameters and respective value ranges, the model architecture reaching the lowest loss on validation data will be automatically found for the user.

REGULARIZATION. To regularize the network, dropout is applied after the input layer and after every max pooling and dense layer. Network weights in all layers are subject to a max-norm constraint and early stopping is implemented with respect to the loss on validation data. We also regularize the learning rate of the network weight optimization. Similar to early stopping, we halve the learning rate if the validation loss did not improve for a certain number of training iterations. This allows the backpropagation to take big steps in the beginning of the training while taking smaller, more precise steps when the loss is close to a minimum. This regularization was not always found to be useful in the past [76], but for our data sets adjusting the learning rate in such a way was beneficial for the training convergence time (see Section A.2 for details on learning rates and network optimization). Adjusting learning rates in some defined manner is also referred to as learning rate scheduling.

CLASSIFICATION. Pysster models are able to perform multi-class and single-label or multi-label classifications. To achieve this, different output layer activation functions and loss functions are used. The output layer of a model always contains as many neurons as input classes are available and each output neuron represents a different class. For single-label classifications (i.e. each input sequence belongs to exactly one class) a softmax activation is used in the output layer to ensure that output values form a valid probability distribution. This can be combined with a categorical cross-entropy loss, the generalized form of the binary cross-entropy introduced in the previous chapter that scales to more than two classes [54]:

$$L(y, \hat{y}) = -\sum_{i}^{C} y_i \, log(\hat{y}_i) \tag{4.1}$$

where C is the number of classes and y a one-hot encoded vector representing the true class label. Together, softmax activation and categorical cross-entropy loss can perform a single-label classification for an arbitrary number of classes. For multi-label classifications (i.e. each input sequence can belong to multiple classes simultaneously) a sigmoid activation together with a binary cross-entropy loss is applied to every output neuron individually. Thereby, the model can predict an arbitrary number of classes at the same time which is not possible with a softmax activation. The final loss value is in this case represented by the sum of all binary cross-entropy losses. Irrespective of the classification mode, we make use of class weighting in our models, that is, to tackle potential class imbalances loss values of minority class training samples are multiplied by a constant factor according to the class frequencies. For example, given a training set of $1,000$ class A sequences, 200 class B sequences and 100 class C sequences, loss values of class B and C sequences will be multiplied by five and ten, respectively.

INPUT DATA. The fundamental input data the models operate on are strings, for example DNA sequences over the $[A, C, G, T]$ alphabet. Alphabets are user-defined and models are therefore applicable to DNA, RNA, protein and other custom data. Moreover, it is possible to add arbitrary, additional handcrafted data both on a per-base and on a per-sequence basis. Examples of the first kind will be shown throughout this chapter, while additional data on a per-sequence basis will be utilized in Chapter 5.

INTERPRETATION. Finally, an important focus of our Python library is the interpretability of networks. For this purpose, we extended previous visualization methods to also report information on positional and class enrichment of motifs and motif co-occurrence. Additional input data, in the following examples in the form of RNA secondary structure and DNA shape, can be visualized as well to further increase interpretability. By adapting the "visualization by optimization" scheme introduced in Section 3.3.4 to our specific problem domain we are also able to visualize what every kernel and neuron in every network layer is learning.

4.1.2 *Implementation Details*

Pysster is implemented in Python and compatible with Python 3.5+. It is MIT licensed and available on GitHub at `https://github.com/budach/pysster` and the Python Package Index (PyPI) at `https://pypi.org/project/pysster`. The code is extensively documented and comes with an application programming interface (API) documentation and tutorials in the form of Jupyter notebooks. Continuous integration of the GitHub repository ensures that all unit tests are executed on every code change. By building on top of Tensorflow [69] and Keras [77], users can train models on both CPU and GPU without having to change any code. In addition, pysster offers a number of convenience functions such as prediction of RNA secondary structures via RNAfold, RNAplfold and forgi [78, 79] and database comparison of learned motifs via Tomtom [45]. A brief tutorial with code examples of pysster can be found in Section A.3.

4.2 Sequence & Other Training Data

In the following sections we use publicly available sequence data to highlight classification problems that CNNs are suitable for and to demonstrate different network visualization methods. Peak-called ChIP-seq data were generated by the ENCODE Consortium [35] and we use data for the TFs CTCF (accession number ENCSR560BUE), JUN (ENCSR569XNP), CEBPB (ENCSR701TCU), E2F1 (ENCSR563LLO) and SRF (ENCSR041XML). Further ChIP-seq peaks for the glucocorticoid receptor TF (GR) are taken from EBI ArrayExpress (accession number E-MTAB-2955 [80]). Genome-wide DNA minor groove width data are based on GBshape [81] and additional DNA shape features are predicted using the DNAshapeR library [20]. Processed CLIP data for a collection of RBPs are taken from [82], as this publication also provides matching background data suitable for binary classification (binding sites of an RBP of interest are classified against a random selection of binding sites of other RBPs) and we additionally predicted RNA secondary structures for binding sites using RNAfold and forgi. Lastly, experimentally validated genomic positions of RNA A-to-I editing events are based on the REDIportal database [38].

4.3 Network Interpretation

The following section demonstrates the network visualization and biological interpretation capabilities of our pysster models. For this purpose, we model two different classification problems: the discrimination of different TF binding sites and the discrimination of repetitive/non-repetitive RNA A-to-I editing events. We first present interpretation options for first-layer kernels and sequence-only models and show subsequently how additional data in the form of RNA secondary structure and DNA shape can be visualized as well without breaking said interpretation options. Afterwards, we present visualizations for all network layers using an artificial data set containing embedded and known motifs.

4.3.1 *Discrimination Of Transcription Factor Binding Sites*

The first model we look at has been trained to discriminate GR binding sites from CTCF binding sites in a binary classification setting. As will become clear later, we chose these specific proteins because they allow us to illustrate multiple interpretation aspects in a concise manner. For both the GR class and CTCF class we randomly selected 20, 000 sequences of length 300 centered at a ChIP-seq peak summit, that means, sequence motifs are expected to be located around position 150 within these sequences. Prior to the network training, data were split randomly into 70% training (14, 000 per class), 15% validation (3, 000) and 15% test (3, 000). A pysster model with 20 kernels per convolutional layer and default hyperparameters otherwise was then trained on the training set while the validation set was used for learning rate scheduling and early stopping. The final predictive performance on the test set amounted to 0.974 auROC. As mentioned, more detailed predictive results for similar classification settings can be found in Section 4.4. In this case, we (ab-)use the high predictive performance only as another validation that the learned network features are relevant.

After the network training we can now visualize motifs learned by kernels of the first convolutional layer. Kernels are visualized analogous to the method introduced

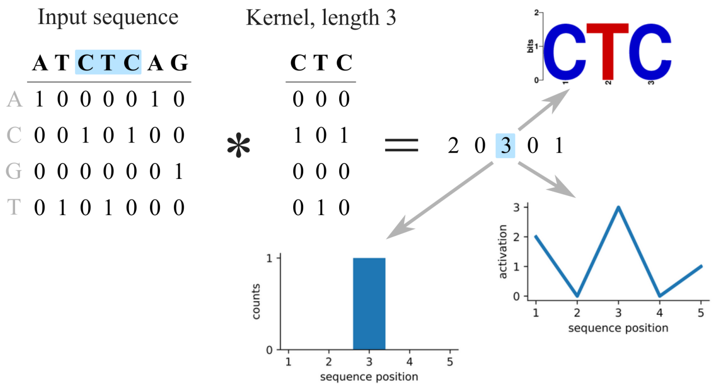


Figure 4.1: **First-Layer Kernel Visualization Principle.** Kernels can be visualized by looking for the position of the maximum activation. This position points towards the subsequence in the input that is most similar to the kernel. Plotting the position of the maximum activation (ideally over many input sequences) as a histogram or directly plotting the complete activation vector also informs about the positional enrichment of the subsequence.

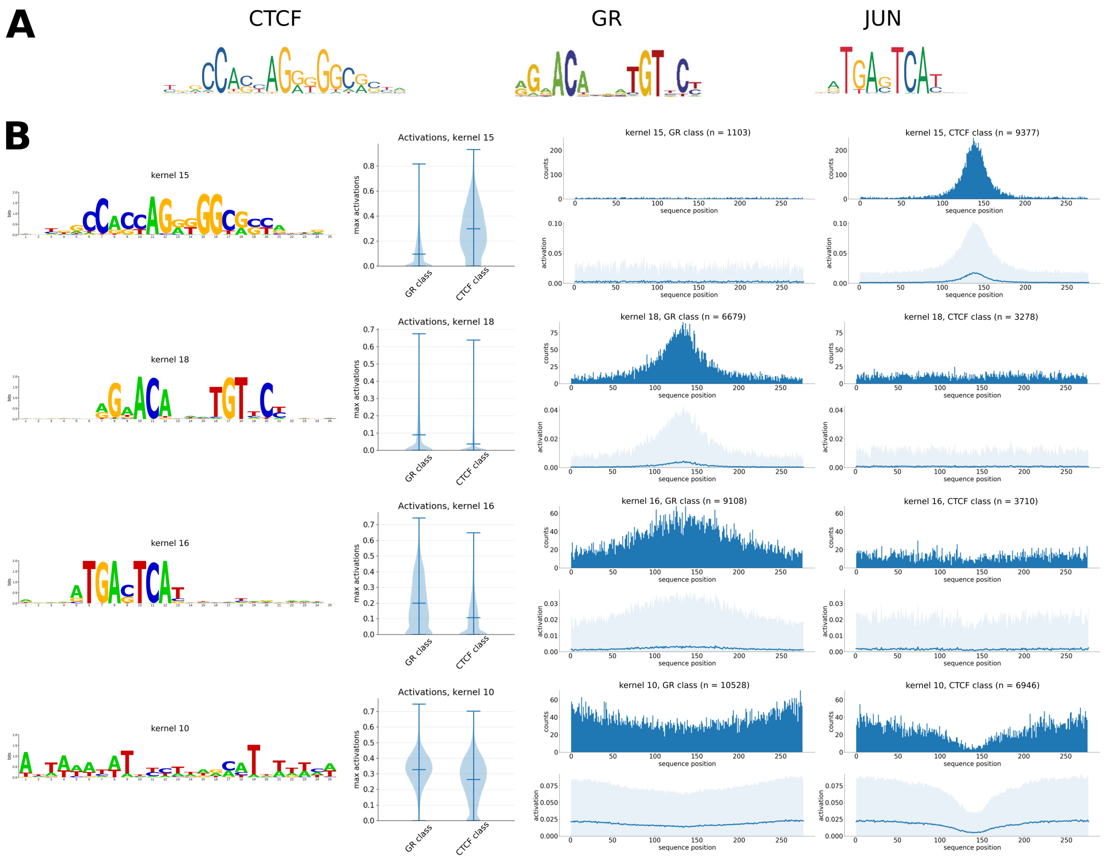


Figure 4.2: **Example Kernels Of The GR/CTCF Model. (A)** Known sequence motifs for CTCF, JUN (both [83]) and GR [84]. **(B)** Four kernels (length 25) and derived sequence motifs, global class enrichments (violin plots, showing maximum activation densities and means) and positional enrichments per class (histogram and activation plots, the latter show mean and standard deviation).

previously: subsequences of the length of the kernel that lead to high kernel activations above a threshold are extracted from input sequences and subsequently used to compute a position frequency matrix and a motif. However, we extend the previous method by also keeping track of the position of the maximum activation for each input sequence to gain information about the positional enrichment of motifs (Figure 4.1). Doing all of this separately for every input class further gives information about the global class enrichment of every motif. Figure 4.2 shows these information visualized for four first-layer kernels and the model is able to recover known sequence motifs and their locations for CTCF, GR and JUN, a major binding partner of GR whose binding motif is expected to be overrepresented near GR binding sites [80]. Positional enrichment of motifs can be visualized by either plotting the positions of the maximum activations as a histogram (in Figure 4.2 using a bin size of one, i.e. the highest point in the histogram shows the most likely motif start location) or by directly plotting the complete activation vectors (depicted below the histograms in Figure 4.2 using the mean and standard deviation over the vectors). Histograms of maximum activations and direct plots of activation vectors contain the same information, but looking at the activation vectors can be favorable if histograms are sparse due to highly imbalanced classes or low amounts of input sequences.

In general, a sequence is only considered for visualization if its maximum activation is above some threshold. Instead of having a fixed threshold, we dynamically determine thresholds for every kernel by computing the average maximum activation per class. The highest average across classes is considered as the threshold for all classes, for example the threshold for kernel 15 in Figure 4.2 is ~ 0.3 according to the violin plots that show average maximum activations and their densities for both classes. We found this threshold to deliver the expected motifs and locations across all studied data sets. The violin plots used to visualize the maximum activations per class also inform about the global class enrichment of a motif (higher values indicate a stronger motif signal). Interestingly, and in addition to known motifs, Figure 4.2 also shows a kernel that learned to recognize AT-rich sequences depleted near the center of both GR and CTCF sequences. Depleted "motifs" are a feature we commonly observe in TF and RBP models and the location of the depletion usually overlaps with the location of other, enriched motifs (e.g. kernel 10 is substantially depleted at CTCF motif locations represented by kernel 15 in Figure 4.2). Thus, looking at all first-layer kernel visualizations provides a convenient overview of the general sequence composition of a data set and is an advisable practice to likewise learn about the features the CNN uses to differentiate the classes.

Looking at individual kernel visualizations is already insightful, but it does not tell us whether different motifs found in the same class (e.g. GR and JUN motifs in our example) are present within the same sequences or whether their signals we see in the histograms represent distinct sets of sequences. Maximum activations of

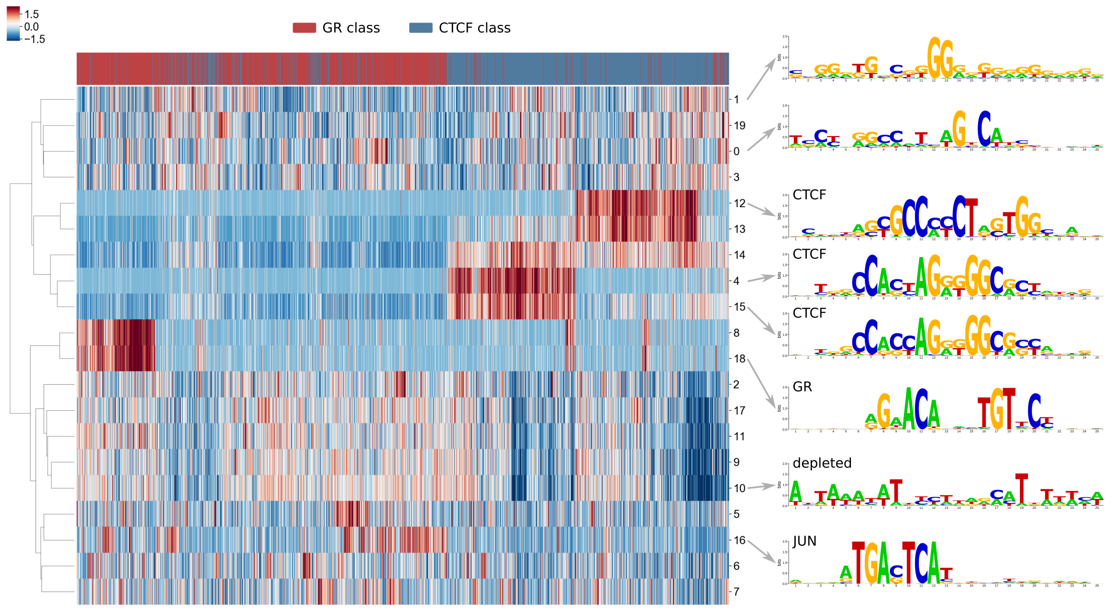


Figure 4.3: **Kernel Clustering Of The GR/CTCF Model.** Normalized maximum activation values of sequences (columns) and first-layer kernels (rows) can be hierarchically clustered to gain insights into motif co-occurrence and class enrichment.

kernel-sequence pairs can therefore further be used to gain information about motif co-occurrence by computing a hierarchical clustering of both sequences and kernels (see Figure 4.3). Clustering kernel-sequence pairs according to their maximum activations ideally places co-occurring kernels close to each other. In practice, it is often observable that strong motifs are learned by multiple kernels and accordingly, these will be close to each other in the clustering (e.g. kernel 4 and 15 in Figure 4.3 both learned the CTCF motif). Looking at the clustering also makes it apparent that two kernels (12 and 13) learned the reverse complement motif of CTCF and consequently do not co-occur with the other CTCF kernels. GR and JUN co-occur within a small subset of GR sequences, however, the kernels seem to mostly be independent (kernel 18 and 16).

A number of kernels (e.g. kernel 0 and 1) learn low-complexity motifs and show neither a preference for a particular class, according to the clustering, nor any kind of global or positional enrichment, according to their respective violin plots and histograms. This brings up the problem of feature importance, i.e. which kernels were most important for the classification or which kernels should a researcher focus on? The GR/CTCF example only used 20 kernels, but studying all kernel visualizations can become time-consuming for bigger models. There is no consensus on how to compute feature importance for CNNs, but to simplify the model evaluation we implemented an importance score and users are presented with an ordered list of kernels when performing visualizations. Kernels can be ordered according to their class-wise average maximum activation values and we define the importance score of a kernel as the difference between the highest average maximum activation and the smallest average maximum activation (higher differences indicate higher importance, see also violin plots in Figure 4.2). Formally, given a CNN model with n input classes and a vector x of length n holding the average maximum activations of every class for first-layer kernel k, the importance score of kernel k is defined as

$$max(x) - min(x). \tag{4.2}$$

The idea is that kernels showing large activation differences across classes are more important for the network to deliver correct predictions. Such kernels also tend to show specific positional enrichment in at least one class according to our experience. In the end, it is important to keep in mind that interpretation efforts, such as the clustering and importance scoring, are intended to give a rough guide on where to start looking for potentially interesting biological insights. While the visualizations presented so far might appear fairly comprehensive, one should never blindly trust a computational prediction without considering one's domain knowledge of the data and problem at hand.

4.3.2 *Direct Visualization Of First-Layer Kernel Matrices*

Kernel matrices of the first convolutional layer have the shape of position frequency matrices (columns are sequence positions and rows are alphabet characters), but the numeric entries of their columns do not form probability distributions and therefore cannot be directly used for the visualization of a sequence motif. However, since we are able to utilize the maximum kernel activations to find matching subsequences in the input sequences, this implies that numeric kernel entries are not completely arbitrary and that higher values are more important. It also implies that it is possible to normalize kernel matrices to create valid position frequency matrices, for instance by individually applying the softmax function (see Equation 3.5 from Chapter 3) to every column of a kernel. To this end, we investigated whether such a direct visualization of kernel matrices as sequence motifs is a feasible alternative to the "detour" visualization presented so far.

We found that the quality of directly visualizing kernels through application of softmax normalization strongly depends on the random weight initialization of kernels (Table 4.1). By default, weights in pysster models are initialized by sampling from a uniform distribution with lower and upper bound -0.05 and $+0.05$, respectively. Applying the described column-wise softmax normalization to kernels of the GR/CTCF model after the network training resulted in matrices with all values close to 0.25

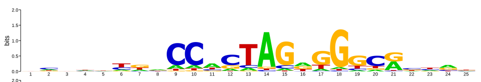
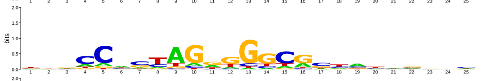
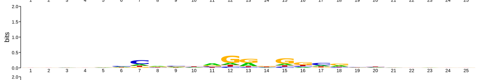
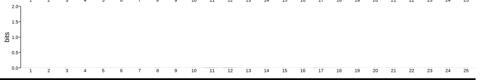
UNIFORM WEIGHT INITIALIZATION	TRAINING ITERATIONS	AUROC	EXAMPLE MOTIF
±1.5	∼ 144	∼ 0.954	
±1.0	∼ 140	∼ 0.957	
±0.5	∼ 123	∼ 0.968	
±0.05	∼ 61	∼ 0.974	

Table 4.1: **Influence Of Weight Initialization On Direct Kernel Visualization.** Visualizing first-layer kernels by softmax-normalizing kernel columns only shows acceptable results if very large boundaries are chosen for the uniform distributions used to initialize kernel weights. However, this negatively impacts both predictive performance and the number of training iterations (shown numbers are averaged over five network trainings). The example motifs show CTCF kernels.

and sequence motifs with essentially zero information content (see example motifs in Table 4.1). Increasing the bounds of the uniform weight initialization helped to spread out the weights and resulted in higher information content motifs, but at the expense of predictive performance (0.974 auROC for ±0.05 bounds, 0.954 auROC for ±1.5 bounds) and a substantial increase in training iterations until early stopping (61 iterations for ±0.05 bounds, 144 iterations ±1.5 bounds). The largest distribution boundaries we were able to test were ±1.5, for larger bounds the network got "stuck" and loss values did not change over time anymore. In addition, we had to remove the max-norm weight regularization from the network for all tests using larger than default bounds to allow weights to grow in the first place. Still, resulting motifs do not look as convincing as motifs created through the "detour" visualization, which is not affected by the random weight initialization because the specific magnitudes of kernel weights and resulting activation vectors do not influence the extraction of subsequences from the input.

To avoid the mentioned downsides of larger initialization boundaries it is certainly possible to train a network with default settings and to artificially increase final kernel entries by simply multiplying them with a constant factor before softmax normalization. This way, both negative and positive entries diverge from each other and sequence motifs tend to have higher information content. However, we also found that the needed multiplication factor is different for every model and even for kernels within the same model. Fine-tuning the information content of motifs, especially if it is unknown what motifs to expect in a data set, is not an advisable practice and we therefore prefer the visualization method presented in the previous section, as it delivers more consistent results and, as a byproduct, also informs about class and positional enrichment.

4.3.3 *Discriminating RNA A-to-I Editing Events & Adding RNA Structure Information*

Proteins binding to DNA/RNA might do so by recognizing specific structural elements in addition to the sequence. While the primary sequence composition of a stretch of DNA/RNA substantially dictates its structure, there is not necessarily a bidirectional mapping possible between sequence and structure and differing sequences are able to form the same structure and vice versa. Accordingly, we are interested in incorporating structural information into our predictive CNN models. In this subsection, we switch to an RNA-based classification problem, the discrimination of repetitive/non-repetitive RNA A-to-I editing events, to describe how RNA secondary structure information can be added as a model input to improve the biological interpretability of the problem.

Generally, RNA secondary structure can be encoded as a categorical feature: a base is either paired or unpaired, or, more precisely, a base can be located in a stem context (paired) or in a hairpin, internal or multi loop context (all unpaired). Computational tools that predict the RNA secondary structure for a given sequence usually output a dot-bracket string (see Figure 4.4A) where a dot indicates an unpaired base and opening and matching closing brackets a base pair. Such strings can further be annotated to assign a precise structural context to unpaired bases. The result is another string (of the length of the original RNA sequence), often over the $[H, I, M, S]$ alphabet where H represents a hairpin loop, I an internal loop, M a multi loop and S a stem region (see Figure 4.4A). Since CNNs operate directly on (one-hot encoded) strings, it is possible to train a model on a secondary structure string (dot-bracket or annotated) instead of the RNA sequence. To use both sequence and structure information at the same time, one could also train a multimodal CNN, for example by designing a model architecture that trains two convolution blocks in parallel, one on the sequence strings and another on the structure strings, and that combines these two blocks before the dense block of the network. This would, however, obstruct the network interpretability. As there is no direct relation between kernels from the sequence and structure blocks, it would not be possible to tell which sequence and structure motifs belong together.

To overcome this problem, we combine sequence and structure strings before the model training to simultaneously learn sequence and related structure motifs from a single model input. Given a sequence string over the four-character alphabet $[A, C, G, U]$ and the corresponding structure string over the four-character alphabet $[H, I, M, S]$, we join the strings into a single intermediate string over an arbitrary alphabet of size 16 representing all possible combinations of characters of the two original alphabets. The new

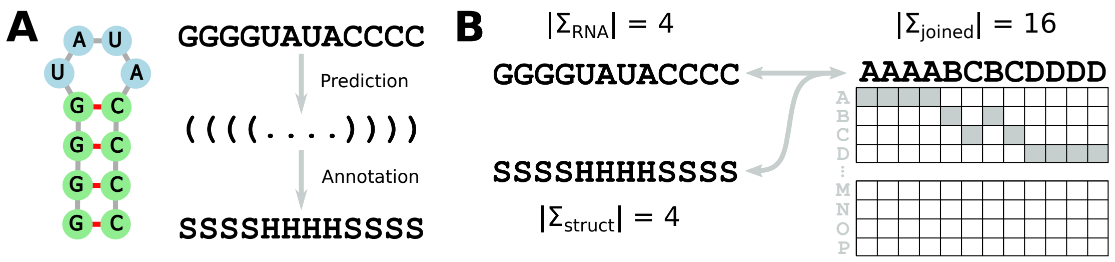


Figure 4.4: **RNA Secondary Structure & Model Input Encoding. (A)** Secondary structure prediction tools often output dot-bracket strings to represent paired and unpaired bases. These strings can be converted into annotated strings indicating more precise structural contexts (*H* meaning hairpin loop, *S* stem region, *I* internal loop, *M* multi loop). **(B)** To enable the simultaneous detection of sequence and structure motifs with pysster, sequence and structure strings are joined into a single new string over an arbitrary alphabet before network training (e.g. using the first 16 letters of the English alphabet). The new string is subsequently one-hot encoded and used as the CNN input.

string is then one-hot encoded and used for model training (see Figure 4.4). Combining strings this way has the advantage of not impairing first-layer kernel visualizations. After the network training we can still extract subsequences from the intermediate strings and then decode these subsequences into the two original strings. Thereby, we can visualize two sequence motifs, one motif over the $[A, C, G, U]$ alphabet and one over the $[H, I, M, S]$ alphabet.

To demonstrate the capability of the approach to capture known motifs the following paragraphs present a binary classification model that discriminates A-to-I editing in repetitive Alu regions from editing in non-repetitive regions. A-to-I editing sites are strongly enriched in repetitive Alu sequences which, when transcribed into RNA, form very similar secondary structures that our model should be able to detect. Input data for the A-to-I editing model has been extracted from the REDIportal database [38] which collects experimentally validated editing events in humans and other species and genomic positions of edited A's in humans are already annotated as being located in Alu repeats or non-repetitive regions. To detect A's in Alu repeats that are more prone to editing than others and to detect how editing patterns differ in non-repetitive regions, we designed a binary classification model by randomly selecting 50,000 edited sites from Alu regions and 50,000 from non-repetitive regions. Sites were then extended into both directions to form sequences of length 301, that is, all sequences in both classes have an edited A at position 151. Secondary structures for all sequences were predicted and annotated using RNAfold and forgi, respectively, and sequence and structure strings were joined as described. Finally, data was split into 70% training, 15% validation, 15% test and the model was trained with default parameters. Somewhat expected, predictive performance on the held-out test set amounted to 0.999 auROC due to the strong Alu signal and is again only treated as another validation for the found motifs.

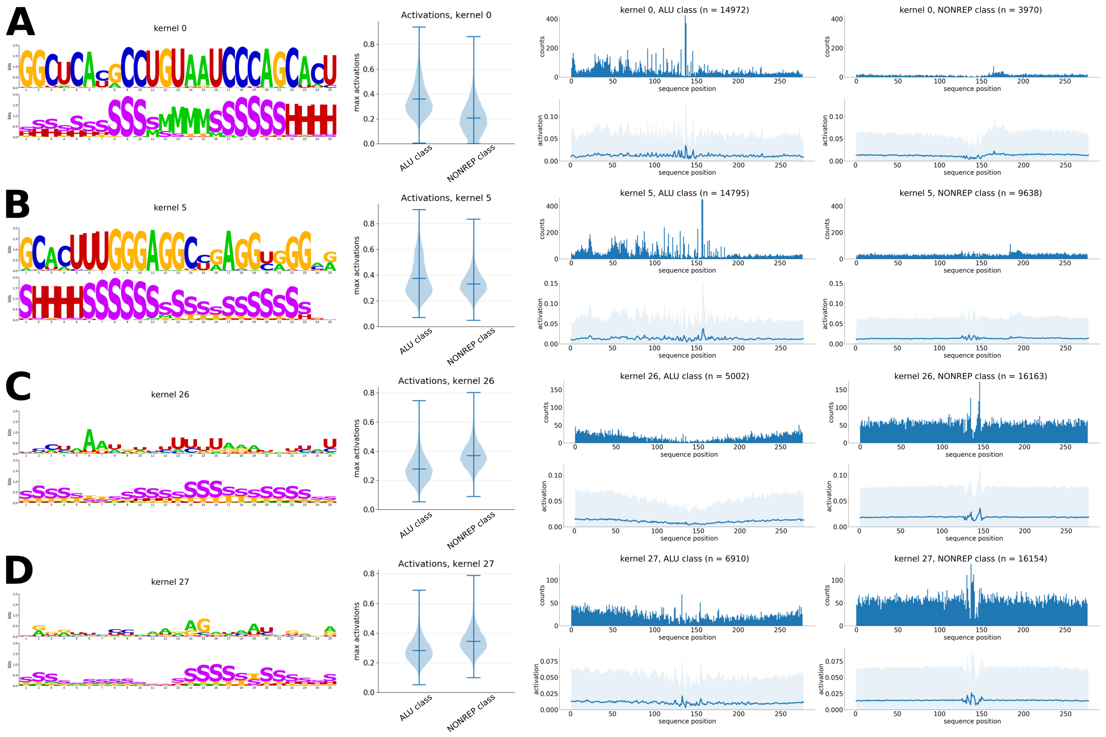


Figure 4.5: **Example Kernels Of The A-to-I Editing Model.** Motifs **(A)** and **(B)** are enriched in the Alu class. Spikes in the histograms and activation plots mark motif starting positions. Since all sequences have an edited A at position 151 this indicates that bases 14 and 15 in **(A)** are often edited. The positional enrichment of motif **(B)** indicates that it is mostly located downstream of edited A's. Motifs **(C)** and **(D)** are enriched in the non-repetitive class and show a general preference for stem regions and a slight preference for a G immediately downstream of the edited A.

Figure 4.5 presents four noticeable kernels that learned to recognize known motifs. Kernels overrepresented in the non-repetitive class show low-complexity sequence motifs, but a strong enrichment for stem regions in the respective structure motifs (Figure 4.5C and Figure 4.5D). This is in line with previous studies which found a general editing preference for stems, but no distinct sequence motif other than an enrichment of G's at the position immediately downstream of edited A's [39]. The model kernel in Figure 4.5D was able to learn this information. Overall, the Alu signal is very dominant in this model and most kernels accordingly have learned to recognize small Alu stretches. The information content of the visualized motifs is usually very high for both sequence and structure (kernels A and B in Figure 4.5). Since all sequences were anchored at an edited A at sequence position 151, the positional enrichment plots can be used to pinpoint exact editing locations. The kernel in Figure 4.5A predominantly starts slightly upstream of sequence position 151 indicating that motif positions 14 and 15 might be often edited. At the same time, the kernel in Figure 4.5B predominantly starts slightly downstream of sequence position 151 indicating that the edited A is not part of the motif itself. Interestingly, the end of the former motif and the start of the latter motif seem to be identical. Indeed, mapping the motifs to the consensus secondary structure of a transcripted Alu repeat leads to an unambiguous match (Figure 4.6) and the A's that the model predicted to be edited have, in fact, previously been shown to be among the most frequently edited sites of Alu repeats [85].

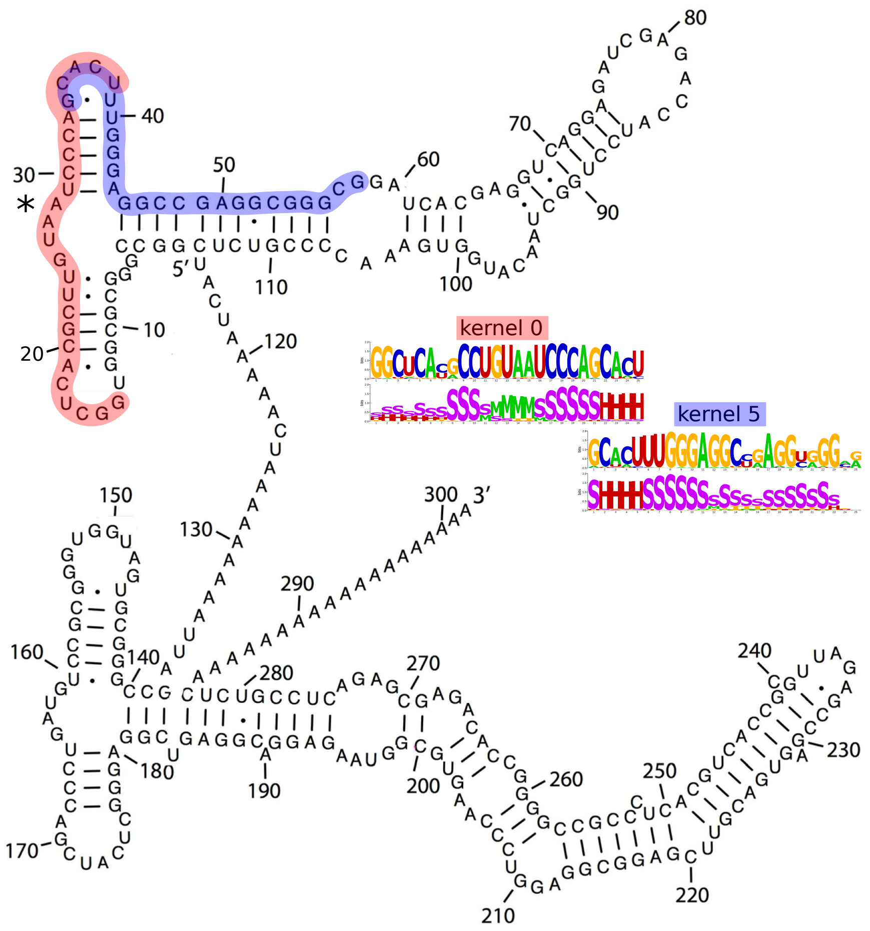


Figure 4.6: **Learned Motifs Match Alu Sequence And Structure.** Mapping learned motifs to a consensus Alu repeat and combining the results with the positional enrichment information of said motifs from Figure 4.5 suggests Alu positions 27 and 28 to be frequently edited, which has been shown previously [85]. The consensus Alu structure has been adapted from [86].

Taken together, the A-to-I editing example demonstrates our ability to effectively learn sequence and structure motifs simultaneously. During the interpretation process additional structure motifs can be used both to strengthen biological insights (locations of edited A's in Alu repeats) and to allow conclusions to be drawn in the first place (enrichment of stems in non-repetitive editing regions). In general, our Python library allows the combination of strings over arbitrary alphabets (including special characters for e.g. dot-bracket strings) and users can therefore freely encode and visualize custom data.

4.3.4 *Adding DNA Shape Information*

The previous section described how RNA secondary structure information can be added to a CNN. Compared to RNA secondary structure, DNA shape information are thought to have a similar influence on protein binding site recognition and thus adding this information to a CNN promises benefits for model interpretation. However, the features used to describe DNA shape are much more complex. While RNA secondary structure can be encoded as a single categorical feature, over a dozen features describing DNA shape are known, all of which are on a continuous scale. Below, we explore two different methods to supplement sequence-based CNN models with DNA shape information by extending our previously described GR/CTCF model. The first method seeks to preserve the familiar motif visualization by simplifying individual shape features whereas the second method directly adds multiple shape features to a model and we show that both methods produce visualizations that reproduce the current DNA shape literature knowledge.

4.3.5 *Discretization Encoding Of DNA Shape*

The alphabet and string joining approach used for RNA secondary structure encoding proved to deliver meaningful results. Therefore, the first DNA shape encoding attempts to fit individual shape features into this existing framework. As a feature of choice, we will look at the minor groove width (MGW), the distance in angstrom (Å) between two opposing phosphates in the DNA backbone, and the already established motifs from the previous GR/CTCF model will be used for illustrations. Only one piece is needed to fit a shape feature into the existing framework - a string representing said feature. To this end, we discretize the otherwise continuous MGW measurements. Using the DNAshapeR tool we first predicted the MGW for all bases and sequences of the GR/CTCF model. Based on genome-wide MGW data from the GBshape database [81] (which uses DNAshapeR internally) we subsequently determined the first, second and third quartiles (Q_1, Q_2, Q_3) of the MGW distribution. Numeric MGW predictions of bases of model inputs were then replaced by one of four different characters: values below Q_1 were replaced with an A, values between Q_1 and Q_2 with B, values between Q_2 and Q_3 with C and values above Q_3 with D (see Figure 4.7 for an example). Thereby, we obtain a string of the length of the original DNA sequence over the $[A, B, C, D]$

alphabet where *A* represents very low MGW values, *D* very high values and *B* and *C* more average ones.

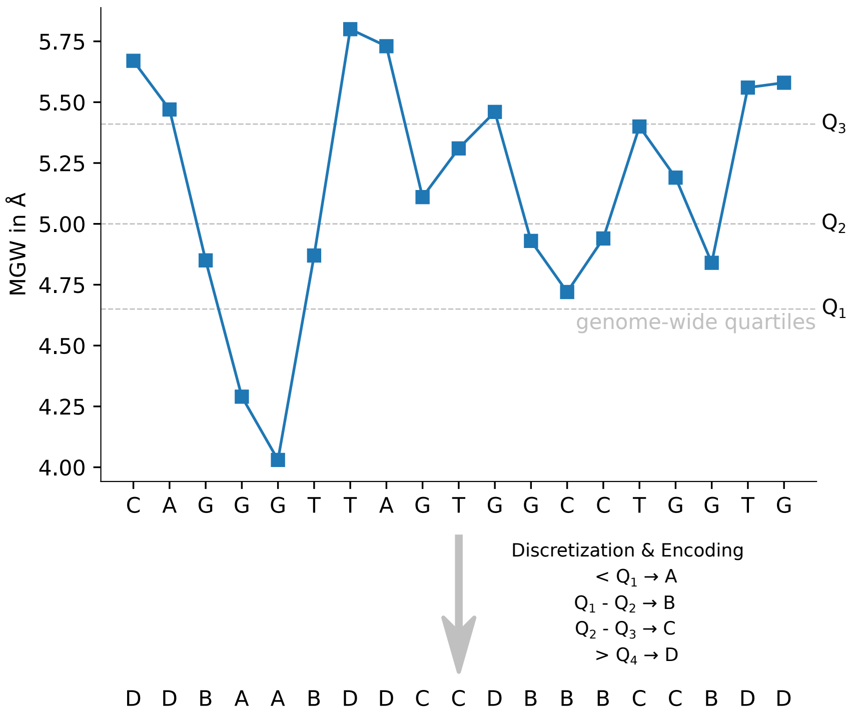


Figure 4.7: **MGW Feature Discretization.** By using the quartiles of genome-wide MGW predictions as thresholds, MGW predictions (y-axis) of individual model input sequences (x-axis) can be discretized into four groups. Groups are encoded using a four-letter alphabet, leading to a final string representation that can be used as a CNN input.

Analogous to the RNA secondary structure encoding, the MGW string can now be used as a model input in addition to the DNA sequence string. We retrained the GR/CTCF model and Figure 4.8 shows example motif visualizations for CTCF, GR and JUN. The appearance of the sequence motifs is not affected by adding MGW information. GR does not show a particularly strong MGW preference, but CTCF and especially JUN display high information content MGW motifs. CTCF appears to prefer a high MGW in the center of its motif, while JUN prefers a very low MGW. Interestingly, many motif positions with high MGW information content show a comparatively low sequence information content (position 16 and 23 for CTCF, 6 and 16 for GR, 14 for JUN), signalling that shape information might supplement binding site specificity. Overall, the model (20 first-layer kernels) did not learn MGW motifs without accompanying sequence motif.

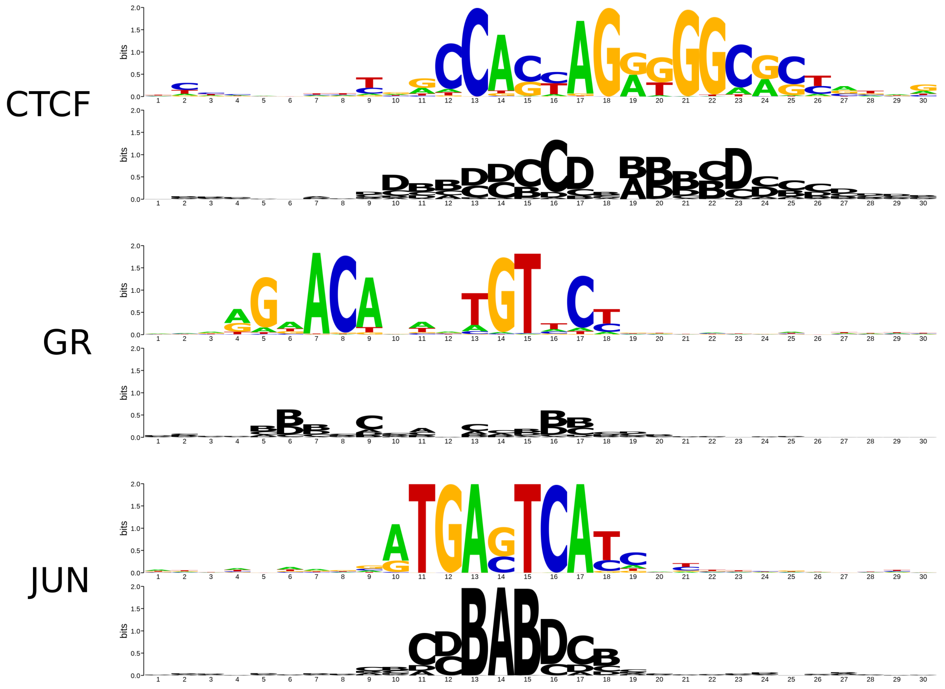


Figure 4.8: **MGW Discretization Example Motifs.** The GR/CTCF model was retrained using MGW "ABCD" strings as additional input. Sequence motifs for CTCF, GR and JUN do not change and MGW motifs show preferences for particular MGW ranges.

4.3.6 *Direct Encoding Of DNA Shape*

Encoding DNA shape features as discretized strings produces familiar motif visualizations, but adding more than one feature at once is impractical due to exploding sizes of joined alphabets. MGW is a major shape feature, but many other features describing the relation of bases within a base pair or the relation of adjacent base pairs can be predicted using the available tools. To enable the addition and visualization of multiple features at once we tested a different encoding and directly added numeric shape predictions (e.g. the predictions from the y-axis in Figure 4.7) as rows to the one-hot encoded sequence matrix (see Figure 4.9). Prior to their addition, individual shape predictions have been standardized (by subtracting their mean and dividing by their standard deviation). Adding multiple features this way enables the simultaneous learning of sequence motifs and shape patterns and does not affect our first-layer kernel visualization strategy. During kernel visualization we still look for the subsequences in the input, or in this case more precisely, for the columns in the input matrix that maximally activate a kernel (see Figure 4.9). The rows corresponding to the one-hot encoding can then be used to receive a sequence for motif visualization and rows corresponding to DNA shape features can be individually visualized by plotting the values of the respective columns as line plots (such visualizations are called *pattern* from now on). Pysster offers users the option to automatically standardize arbitrary input features, but will visualize such features using the non-standardized values in output plots.

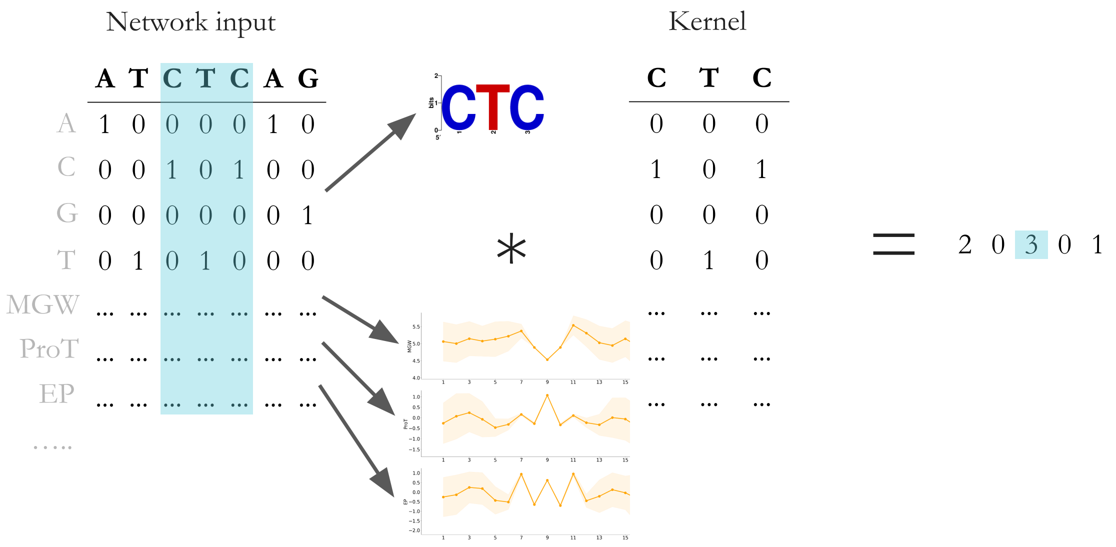


Figure 4.9: **Direct Shape Feature Encoding.** By concatenating further features as individual rows to the one-hot encoded input matrix multiple features can be added to the network at once. Thereby, kernels can be visualized using the established strategy: input columns that maximally activate a kernel (light blue accentuation) can be extracted and rows belonging to the one-hot encoding are still being used to create sequence motifs, while other rows are individually plotted using summary statistics. As this figure technically only shows a single network input, orange lines and shades indicate mean and standard deviation over many inputs for illustrative purposes. *ProT* symbolizes the propeller twist feature and *EP* electrostatic potential.

As a first test, we retrained the GR/CTCF model with only the numeric MGW feature as additional input. Resulting CTCF, GR and JUN motifs and MGW patterns are shown in Figure 4.10A. The figure also shows a comparison of the MGW patterns with the previously learned "ABCD" motifs from Figure 4.8. Both visualizations are very similar in that they show the same general trend regarding low/high MGW values for each position and in that the information content in the motifs and the standard deviation in the patterns correlate. High information content positions always show a comparatively low standard deviation while low information content positions exhibit a very high standard deviation. Therefore, the simplified "ABCD" string motifs constitute a familiar and effective visualization. To demonstrate that the features learned by our model can recover known shape patterns, Figure 4.10B and C show heatmap visualizations provided by the TFBSshape database [87] for CTCF and JUN (GR was not available). In both cases patterns and motifs learned by our models match the database entries.

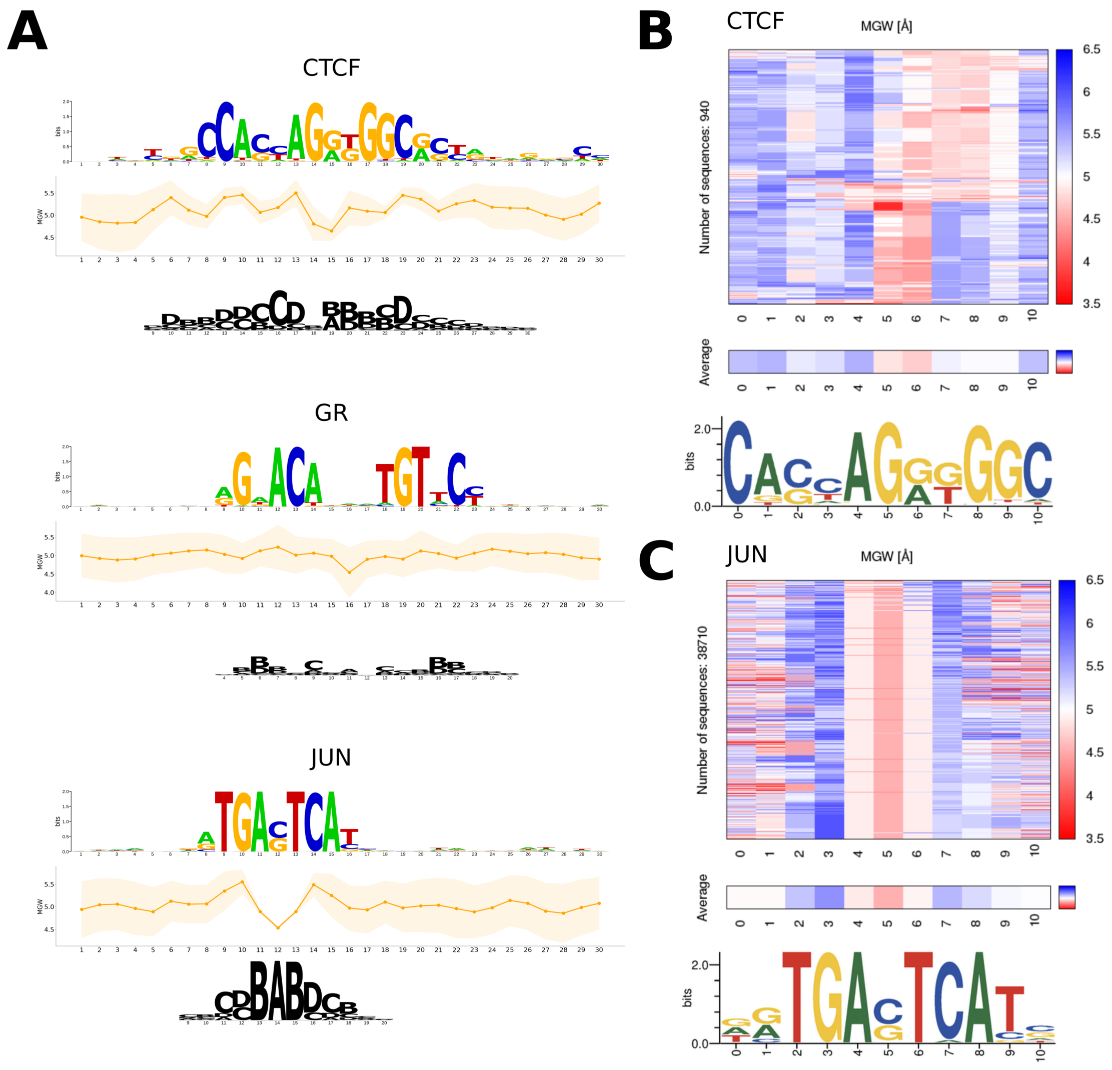


Figure 4.10: **MGW Pattern And Motif Comparison.** Line plots below motifs in **(A)** show mean and standard deviations for learned MGW patterns. Below that, "ABCD" string motifs from Figure 4.8 are shown as a comparison. Screenshots in **(B)** and **(C)** depict known MGW patterns for CTCF and JUN from the TFBSshape database (GR was not available).

The big advantage of the direct encoding is that it is possible to add an arbitrary number of additional features at once. Thus, we retrained the GR/CTCF model yet again with five predicted shape features - MGW, Roll, propeller twist (ProT), helix twist (HelT) and electrostatic potential (EP) - and Figure 4.11 illustrates the resulting visualizations for CTCF, GR and JUN. Notably, not all features are equally important for every protein (or position), for example while MGW does not appear to be particularly important for GR, the Roll and EP features show specific patterns for this protein.

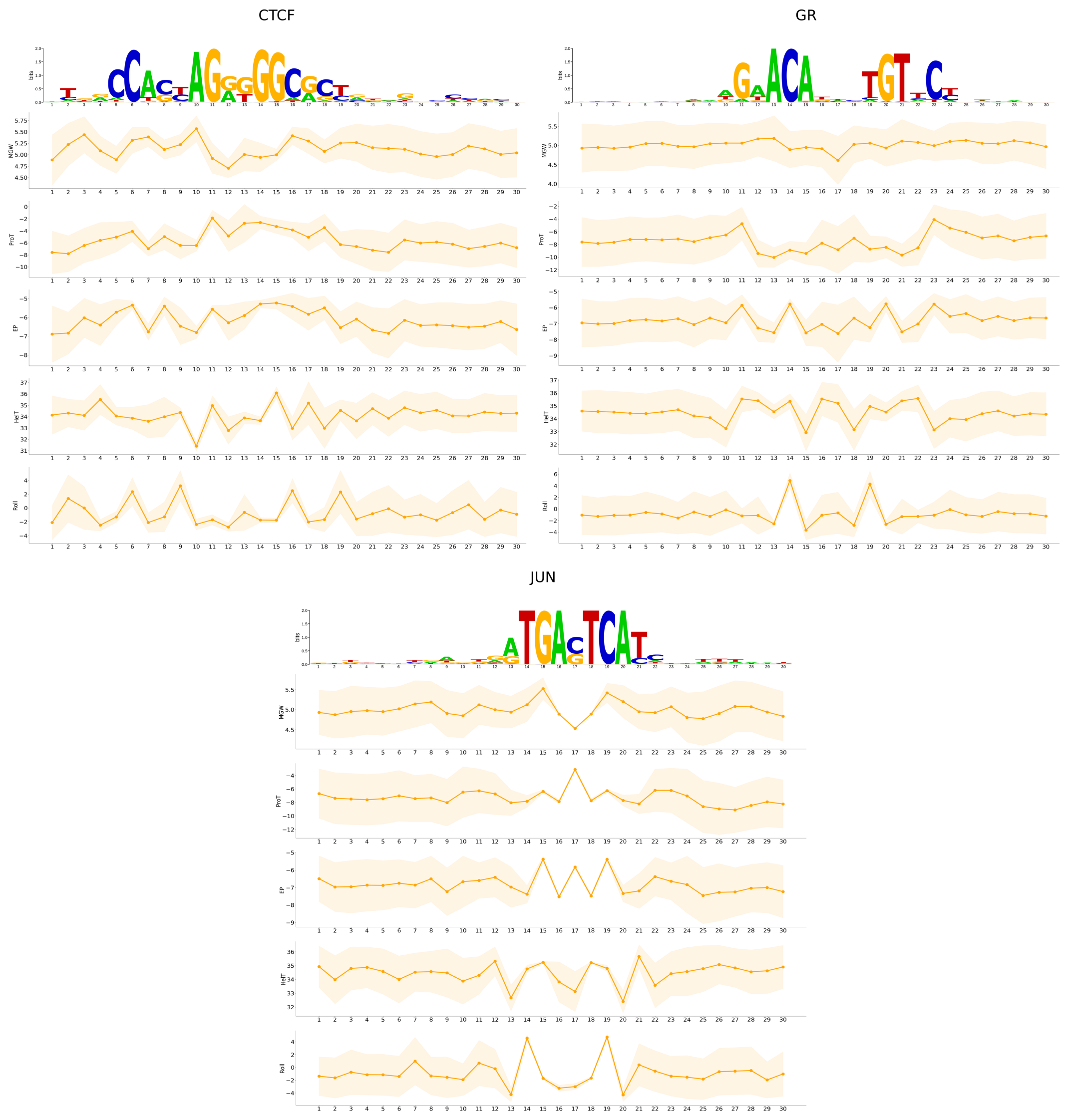


Figure 4.11: **Adding Multiple Shape Features At Once.** Features visualizations for the GR/CTCF model trained with five shape features. In order: MGW, propeller twist (ProT), electrostatic potential (EP), helix twist (HelT) and Roll.

In summary, we demonstrated that our models can recover known DNA shape patterns and are additionally able to utilize multiple features at the same time to aid the model interpretation process. Example visualizations for other DNA binding proteins can be found in Section A.4 and predictive performance results for all tested proteins and shape features can further be found in Section 4.4. Analogous to the RNA secondary structure encoding, usage of DNA shape as input is only an example. Users of our Python library are free to use custom data in the presented way.

4.3.7 *Visualizing All Network Layers By Optimization*

The visualizations presented so far were all concerned with only the first convolutional layer - the layer directly connected to the input. Mapping positions of the maximum kernel activations to input positions, however, is not applicable anymore to downstream layers in the network, because the max pooling operations break the direct connection to the input. To still visualize downstream layers of a network, visualization by optimization can be used (see Section 3.3.4). It has previously been applied to image data and could illustrate how networks hierarchically extract features from inputs. Starting layers of image classifiers, for instance, usually learn to recognize individual colors and line directions while later layers learn textures and more complex patterns until at some point shapes are learned that resemble input objects (see Figure 4.12). We adapted the visualization by optimization approach for biological sequence data and show in the following that a comparable feature hierarchy is also learned for our particular input data.

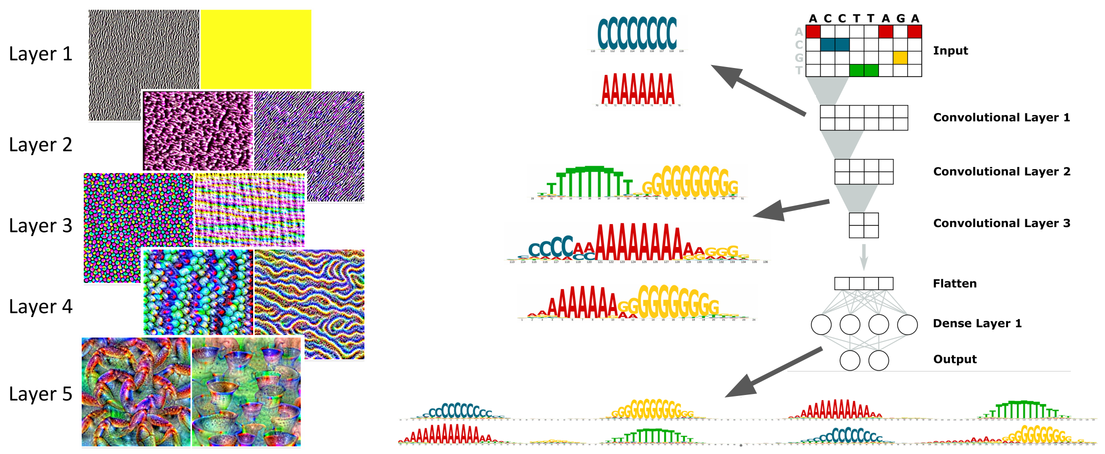


Figure 4.12: **Visualization By Optimization Overview.** Visualizing what every unit (convolutional kernel or neuron) in every layer of a CNN is learning assists in understanding how features are hierarchically extracted during network training. Images on the left side of the figure are adapted from [88] and depict exemplary convolutional kernel visualizations from different layers of an image classification model (two per layer).

Given an already trained network, the idea of visualization by optimization is to generate an input that maximizes the output of a specific network unit. Such an input depicts the optimal input a unit might react to. Units can be individual kernels, neurons or combinations thereof and applying this idea to every network unit creates a collection of inputs that depict the hierarchical feature extraction and classification process. To generate an optimal input for a network unit a gradient ascent optimization is applied that maximizes the unit output with respect to a randomly initialized input. Unit outputs are either the activation vectors of convolutional kernels or the real-valued output numbers of neurons. In the case of the sequence-only GR/CTCF model a network input would be a matrix of shape 4×300 and entries of that matrix would now be the weights that are adjusted during the gradient ascent training.

More formally, the objective function we are trying to optimize can be stated as:

$$L = \max(c) \tag{4.3}$$

where c is the activation of a kernel or a neuron. Since kernel activations are vectors, they can be optimized in different ways and we found that only optimization of the maximum value leads to satisfying results. Neurons only produce a single output number and are trivial to optimize. Given a randomly initialized input through a set of weights w, we perform the gradient ascent optimization as follows:

$$w^* = w + \epsilon g(\nabla_w L(w)) \tag{4.4}$$

where w^* is the adjusted set of weights, ϵ the learning rate, $\nabla_w L(w)$ the gradient and g a function that normalizes the gradient with respect to the $L2$ norm[1]:

$$g(x) = \frac{x}{\sqrt{\sum_i x_i^2}}. \tag{4.5}$$

We mentioned earlier that the direct visualization of a kernel matrix is not advisable, but for the visualization by optimization procedure to work we essentially have to do exactly this. The generated input is a position frequency matrix-like object, just as a convolutional kernel, and we now have to normalize it in some way to directly visualize it. Visualization attempts of image classifiers are confronted with a similar problem in that they have to manually find appropriate normalization or regularization schemes that create "viewable" images. We found the $L2$ normalization to be helpful for our data, as it keeps the partial derivatives in the gradient small and close to each other, which in turn results in a more consistent gradient ascent process. Moreover, it affects the learned weights such that a final column-wise softmax normalization without any further scaling is usually enough to produce suitable sequence motif visualizations. Overall, the optimization process is controlled by three hyperparameters: the learning rate (0.02 by default), the number of gradient ascent iterations (600 by default) and the boundaries of the uniform distribution used to initialize the input (± 0.1 by default). Higher values for all parameters can be used to increase the information content of the generated input. They can also be tuned if the iterative gradient ascent scheme is not converging for a particular model.

1 `https://keras.io/backend/`, accessed: 20.9.2019

We will now demonstrate the method using an artificial data set that contains defined motifs. Therefore, a positive and a negative class were created by randomly sampling sequences over the DNA alphabet and by implanting motifs into the positive class. The positive class consists of two kinds of sequences: one half (5,000 sequences) contains a "CCCCCCCCCC" and a "GGGGGGGGGG" motif, always starting at the exact same positions within a sequence, while the other half (5,000 sequences) contains an "AAAAAAAAAA" and a "TTTTTTTTTT" motif, likewise always starting at the same positions. Figure 4.13 shows the two kinds of positive input sequences visualized as full-length motifs. One can see that the C and G and A and T motifs are always co-occurring, respectively, while G and A, for example, never co-occur. The negative class (10,000 sequences) does not contain any motifs and all sequences in both classes are of length 140.

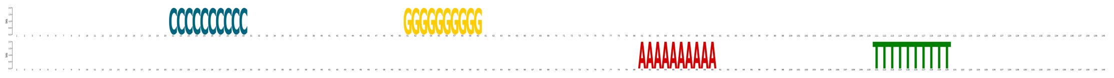


Figure 4.13: **Positive Class Of the Artificial Data.** Random sequences of length 140 from the DNA alphabet were sampled to create the positive class. Afterwards, motifs were implanted at exact locations. One half of the positive class therefore contains a C and G motif, while the other contains an A and T motif.

We trained a small CNN comprised out of two convolutional layers with 20 kernels of length eight per layer, a dense layer with five neurons and an output layer with two neurons on the data. Although we already have a visualization method for first-layer kernels, we will now look at said layer, because the existing visualizations can be utilized to validate the new approach. We would expect the visualizations to be similar. Figure 4.14 shows the results of the visualization by optimization approach for all kernels of the first layer and the reference visualization for one of the kernels. Motifs are identical and we can now be more confident that results obtained from downstream layers are correct. Generally, it is noticeable that all optimized inputs of first-layer kernels only possess eight adjoined positions with an observable information content, which is desirable as eight is the length of our kernels. We mentioned already that only optimization of the maximum activation leads to this result (Equation 4.3). Optimization of other metrics, such as mean activation or sum of the activation vector, created motifs that were stretched out over the complete 140 bases long input, which we did not consider to be useful (e.g. the input of the first kernel in Figure 4.14 would show high information content A's for all 140 positions instead of just eight positions).

One can further notice that the locations of the four implanted motifs in the gradient ascent visualization do not match the locations in the real input. Indeed, re-running the optimization of all first-layer kernels results in the same motifs, but different random locations yet again (not shown). This is due to convolutional layers being equivariant to translations in the input as discussed earlier in the thesis. Kernels can therefore detect motifs irrespective of their location in the input and the gradient ascent-derived inputs reflect this property.

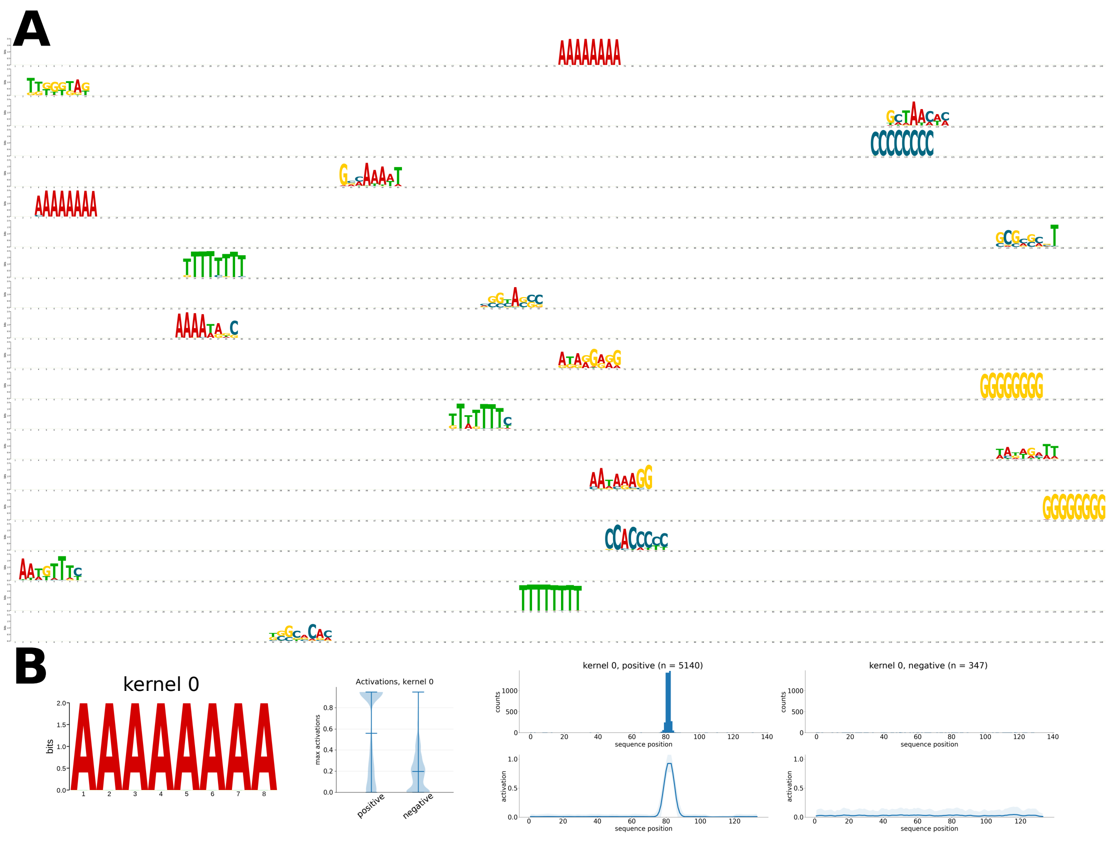


Figure 4.14: **Visualization Of The First Convolutional Layer.** Every row in **(A)** shows the generated input for one of the 20 kernels of the first layer. Comparing the generated input of the first kernel to the visualization created by the default approach in **(B)** indicates a strong agreement.

Figure 4.15 illustrates the generated inputs for all 20 kernels of the second convolutional layer. The second layer appears to learn combinations of first-layer motifs. Combinations usually comprise two previous motifs and feature no spacing between motifs, possibly due to the max pooling applied after the first convolutional layer. It is also noticeable that even motif combinations that never occur in the real data are being learned (e.g. combinations of the A and G motif). In addition, motifs are much longer than the kernel length eight instructs, presumably because the second convolutional layer operates on an input matrix whose rows conceptually already represent full motifs instead of individual bases. Again, the network is not yet learning exact motif locations, re-running the optimization yields the same motifs at random locations (not shown).

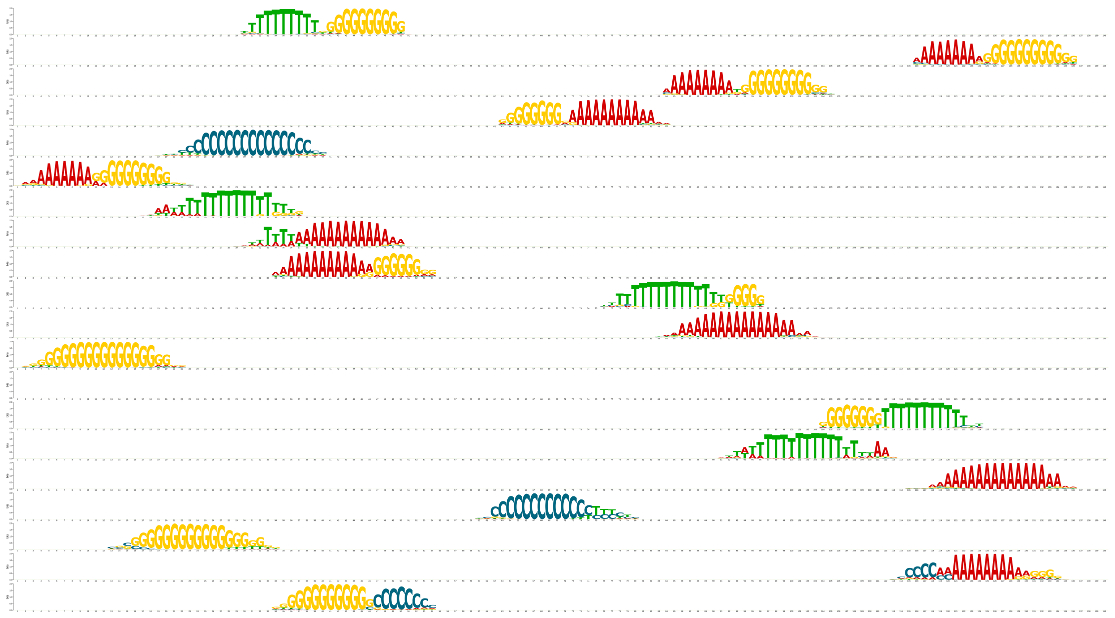


Figure 4.15: **Visualization Of The Second Convolutional Layer.** Kernels in the second convolutional layer learn combinations of first-layer motifs. However, some of these combinations do not exist in the real input data the model was trained on.

Applying visualization by optimization to the five neurons in the dense layer generates inputs as shown in Figure 4.16 and we finally see exact motif locations and correct motif spacing. Interestingly, some neurons seem to learn inputs that contain all four motifs at the same time, resembling a consensus input. The generated input for the second to last neuron shows motifs at locations where we expect motifs, but the order is not correct. To understand this, we can look at the last network layer, the output layer.

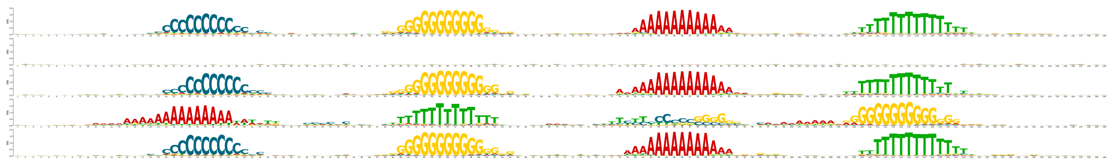


Figure 4.16: **Visualization Of The Dense Layer.** Generated inputs for the five neurons in the dense layer show exact motif locations and correct motif spacing for some neurons. Neuron four (from the top) appears to learn exact locations, but an incorrect order. The optimization for neuron two did not converge.

The output layer consists of two neurons and each neuron represents one class. Therefore, visualizing the output layer can help to understand what the network considers to be the most representative sequence for each class. For all previous layers we maximized unit activations. The output layer uses a softmax (or sigmoid) activation for classification which saturates at one. To allow for an effective maximization we replaced the activation with a linear activation ($f(x) = x$) before applying gradient ascent and results are shown in Figure 4.17. The generated input for the neuron of the positive class shows all four motifs with correct locations and spacing. We likely already observe the same generated input in the previous dense layer, because the data set is very easy to learn. The neuron representing the negative class is more interesting. The real negative input was completely random and did not contain any motifs, nevertheless, the generated input shows clear motifs and locations. Notably, motifs and locations are comparable to motifs from the positive class, but the order of motifs is different. Apparently the network is forced to somehow model a negative sequence and it has to do so by using motifs from the positive class (since there are no motifs in the negative class) and by arranging the motifs in a way that cannot possibly form a positive class sequence. Shuffling the order of motifs seems to be the solution chosen by the network to learn the classification task. Indeed, retraining a different model on the same data from scratch and applying the visualization scheme leads to a different motif order for the neuron representing the negative class.

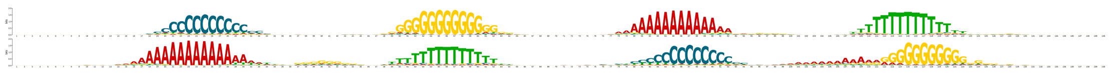


Figure 4.17: **Visualization Of The Output Layer.** The generated input at the top represents what the positive class neuron assumes an average positive sequence to look like. The generated input at the bottom represents the same for the negative class neuron. As there were no motifs in real negative inputs, the network appears to reorder motifs from the positive class to model a sequence that is never found in the positive class (and neither in the negative class).

In summary, the presented results indicate that visualizations of internal network layers have to be interpreted very cautiously. Convolutional layers beyond the first one are learning motif combinations that never actually occur in the real data. Similarly, visualizations of negative class neurons in the output layer generated inputs that do not exist in the real data. On top of that, the contrived artificial data set was completely

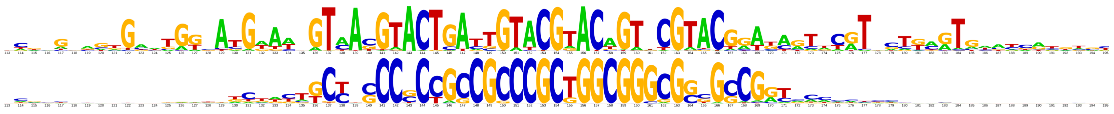


Figure 4.18: **Output Layer Visualization Of The GR/CTCF Model.** Generated inputs for the GR and CTCF class neurons (top and bottom, respectively) only show the general sequence composition of binding sites and resemble consensus sequences. The motifs only show the center region around position 150 (full-length inputs of length 300 have been omitted for brevity). As binding sites are enriched around position 150, but are not precisely located at the same positions, we only see a general preference for GC-rich regions in the CTCF class and individual di- and trinucleotides from the GR motif (possibly JUN motif) in the GR class.

free of noise. A real, noisy biological data set is much harder to interpret, but even when looking at a comparatively simple classification, such as the GR/CTCF model, the usefulness of the visualizations become questionable (see Figure 4.18 for output layer visualizations of the GR/CTCF model). While visualization by optimization can be useful for image classification models (as the human brain is very efficient at recognizing patterns even in very noisy images) we are not convinced of its value for biological sequence data. Nevertheless, it is very interesting to see the inner workings of such a network at least once.

4.4 Network Performance

The models presented so far were useful to demonstrate visualization and interpretation aspects of CNNs in a concise manner. The last section of this chapter focuses on additional binary classification settings to investigate both predictive and training runtime performance as well as their dependence on additional DNA shape and RNA secondary structure input. All measurements were performed on a 64-bit Linux machine with two E5-2697Av4 CPUs (32 cores, 64 threads) and an NVIDIA Titan X GPU. In addition, all results are based on pysster version 1.2.1 with TensorFlow version 1.14 as back-end.

4.4.1 *DNA Shape Performance*

We first looked into the performance of TF and DNA shape models by classifying TF binding sites versus genomic background regions in a binary classification setting for six different proteins. As before, sequences of length 300 centered at ChIP-seq peak summits were used for individual TFs, but instead of classifying one TF against another TF we now used the flanking regions of peaks for the background class. To this end, we randomly selected regions of length 300 located downstream and upstream of peaks (selected regions and peaks do not overlap) such that we received balanced classes (20,000 sequences per class). Classes were further randomly split into distinct sets of 70% training, 15% validation and 15% test data. This splitting was repeated 50 times and all presented measurements are medians of the resulting 50 models. Models were trained using kernel length 30 and default hyperparameters otherwise.

To obtain baseline predictive performance results, we trained sequence-only models for each protein (TF versus its respective flanking background). Subsequently, we added DNA shape features, individually and jointly, to assess their predictive performance impact (Table 4.2). We observe that baseline results are already very high (0.978 auROC for the CEBPB model being the highest, 0.806 auROC for GR being the lowest) and that adding DNA shape features has essentially no consequences. Depending on the TF, some DNA shape features seem to perform better than others,

TF	SEQUENCE BASELINE	MGW "ABCD"	MGW	PROT	EP	HELT	ROLL	ALL FEATURES
CTCF	**0.965**	0.959	0.964	**0.965**	0.964	0.964	0.963	0.964
CEBPB	**0.978**	0.973	**0.978**	**0.978**	**0.978**	0.977	0.977	0.977
JUN	**0.969**	0.961	0.968	**0.969**	**0.969**	0.968	**0.969**	0.968
E2F1	0.83	0.826	0.83	0.829	**0.832**	0.828	0.828	0.829
SRF	0.906	0.899	0.906	**0.907**	**0.907**	0.903	0.903	0.904
GR	**0.806**	0.771	0.8	0.804	0.802	**0.806**	0.797	0.797

Table 4.2: **DNA Shape Feature Performance With Flanking Regions Background.** Table entries are median test set auROC measurements and each based on 50 models trained on different training/validation/test data splits. The "all features" column represents models using all five examined DNA shape features at the same time (minus the MGW "ABCD" string). Bold values depict maximum row entries.

but since we often have to look at the third decimal place to find auROC differences, this has no practical meaning. We also tested the impact of the MGW "ABCD" string discretization encoding, which leads to a persistent performance reduction, pointing to a loss of information during the discretization. Here, with 0.771 auROC, GR shows a meaningful performance difference compared to the baseline (0.806) and compared to the direct addition of MGW to the input matrices (0.8) and the very weak MGW "ABCD" motif observed earlier for GR (Figure 4.8) might be a potential explanation, as additional input data does not necessarily causes better performance if it does not carry additional extractable patterns. For all other proteins, however, differences are much smaller (e.g. CTCF: 0.965 baseline, 0.959 MGW "ABCD" string, 0.964 direct MGW addition).

TF	SEQUENCE BASELINE	MGW "ABCD"	MGW	PROT	EP	HELT	ROLL	ALL FEATURES
CTCF	0.98	0.977	**0.981**	0.98	0.98	**0.981**	0.98	**0.981**
CEBPB	0.979	0.976	0.981	0.978	0.979	0.978	**0.983**	0.982
JUN	0.974	0.971	0.976	0.978	0.977	0.976	0.977	**0.979**
E2F1	0.906	0.902	0.913	0.913	0.913	0.912	0.919	**0.924**
SRF	0.935	0.936	0.939	0.942	0.942	0.935	0.941	**0.948**
GR	0.912	0.91	0.916	0.909	0.91	0.914	**0.921**	0.92

Table 4.3: **DNA Shape Feature Performance With Dinucleotide Shuffled Background.** Table entries are median test set auROC measurements and each based on 50 models trained on different training/validation/test data splits. The "all features" column represents models using all five examined DNA shape features at the same time (minus the MGW "ABCD" string). Bold values depict maximum row entries.

To further investigate the predictive performance impact of DNA shape features, we also tested another common background class. Everything else being the same, we replaced the flanking regions background with a dinucleotide shuffled background, that is, sequences from the TF classes were shuffled such that their dinucleotide content remained constant. DNA shape features were then predicted for the shuffled sequences and model results can be found in Table 4.3. For this background, one can notice a slightly higher positive impact of DNA shape features (often already observable in the second auROC decimal place) and the negative impact of the "ABCD" string discretization seems to be moderated as well (e.g. GR: 0.912 baseline, 0.91 MGW "ABCD", 0.92 all features). This might be caused by the fact that, while dinucleotide shuffled backgrounds are popular, they nevertheless represent a more artificial setting than the flanking regions background and are easier to distinguish from real sequences.

TF	SEQUENCE BASELINE	MGW "ABCD"	MGW	PROT	EP	HELT	ROLL	ALL FEATURES
CTCF	100	**72**	111	108	99	106	102	93
CEBPB	**58**	74	71	81	92	70	79	98
JUN	72	**67**	83	83	84	72	88	82
E2F1	71	**55**	88	121	110	81	105	105
SRF	**62**	63	71	68	70	**62**	111	98
GR	72	**59**	103	105	95	77	103	82

Table 4.4: **DNA Shape Training Runtime Performance.** Table entries depict median GPU training times in seconds and are each based on 50 models trained on different training/validation/test data splits (using dinucleotide shuffled backgrounds). 28,000 training and 6,000 validation sequences (all length 300) were used for each split. Bold values highlight minimum row entries.

With respect to the runtime performance, we observed that adding more DNA shape features tends to result in longer training times (Table 4.4). Models operate on $m \times n$ input matrices, where n is the length of a sequence and where m is bound by the dimensionality of a sequence's one-hot encoding and the number of additional features. Therefore, adding more features can be expected to raise training times. In spite of this, it is also observable that adding features can occasionally improve the training time (e.g. CTCF: 100 seconds baseline, 93 seconds all features) and in no case did the more than doubling of the input (four-dimensional one-hot encoding + five additional shape features) led to a doubling of training time. Even more interestingly, using the "ABCD" string discretization, i.e. a 16-dimensional one-hot encoded input, produces the fastest training times for four out of six proteins (e.g. 72 seconds for CTCF compared to its 100 seconds baseline). In these cases, adding further input benefits the convergence of the training and less training iterations are required before the automated early stopping takes effect. Unfortunately, there is no consistent pattern noticeable in our used data that would explain, or allow to predict whether a particular kind of additional data benefits training convergence.

In summary, since DNA shape features are predicted solely based on sequence and therefore correlate with said sequence, it is not obvious what kind of predictive performance impact to expect, but it is certainly possible that shape features do not supply significant new information to a complex CNN model, especially given the already very high predictive results of baseline models. Conversely, although adding DNA shape features increases the biological interpretability of CNN models, we found only marginal predictive performance improvements, if any. The runtime performance shows stronger variations and can be affected both positively and negatively through additional features.

4.4.2 *RNA Secondary Structure Performance*

To measure the predictive performance for RNA sequences, we chose data sets from the ssHMM publication [82], a tool for the unsupervised extraction of RNA sequence/structure motifs. Here, the authors already prepared filtered RBP data sets suitable for binary classifications by providing CLIP-seq based RBP binding site peak locations for a collection of proteins. Binding sites of a protein of interest can then be classified against a balanced, random selection of binding sites of all other proteins. For our pysster models, we used the provided genomic peak locations, which were already split into positive and background class for each protein, to extract sequences of length 200 centered at peak summits. We predicted and annotated their respective secondary structures as described earlier and trained models with kernel length eight and default hyperparameters otherwise. Reported results are again medians based on 50 different data splits and models. RBP motifs are usually short and we therefore chose kernel length eight to briefly compare learned motifs with the literature knowledge. We also used the opportunity to compare pysster's performance with GraphProt [89], a frequently used classifier for RNA sequence/structure data. Compared to CNNs, GraphProt represents a very different machine learning approach and extracts both sequence and corresponding RNA secondary structure binding preferences using a graph-kernel approach and subsequently uses the extracted features to classify inputs with a support vector machine.

Table 4.5 shows predictive performance and runtime results for a number of proteins with known sequence or structure preferences. Predictive pysster results on the held-out test data are very high (on average ~ 0.931 over all proteins) and outperform GraphProt for all tested proteins (~ 0.816 over all proteins). In terms of runtime, both GPU and CPU-only runs of pysster compare favourably with GraphProt. While running pysster requires an average of $\sim$ 80 seconds on the GPU and $\sim$ 165 seconds using only the CPU, running GraphProt takes $\sim$ 67 minutes on average. This is due to GraphProt using only a single thread, while pysster is fully parallelized and makes more efficient use of the available hardware.

RBP	PYSSTER AUROC	GRAPHPROT AUROC	PYSSTER (GPU) SECONDS	PYSSTER (CPU) SECONDS	GRAPHPROT SECONDS
PUM2	0.948	0.853	49	102	2503
NOVA	0.965	0.881	58	101	2531
QKI	0.967	0.888	45	90	2014
SRFS1	0.94	0.87	199	457	10730
TAF2N	0.913	0.7	74	157	3960
IGF2BP	0.855	0.701	52	81	2400

Table 4.5: **RBP Performance: pysster & GraphProt.** All pysster measurements are medians based on 50 models that used different data splits. GraphProt (version 1.1.7, with motif length 8 and default parameters otherwise) was only executed once due to its high runtime. All runtimes include RNA secondary structure predictions, because GraphProt automatically computes these internally. We used a utility function from the pysster library to predict structures via RNAfold. The total amount of training and validation sequences (all length 200) for each RBP model are (in row order) 10368, 11132, 8700, 43472, 17302 and 10260.

The classification of RBP binding sites represents a more canonical RNA-based classification task compared to the already presented RNA A-to-I classification and while this section of the thesis focuses on predictive and runtime performance, we therefore also want to briefly investigate the learned motifs. To this end, Table 4.6 shows motif visualizations from pysster, GraphProt and ssHMM, a hidden Markov model-based tool that learns motifs in an unsupervised fashion from the protein of interest class and with regard to the learned sequence/structure motifs, all three tools can recapitulate existing knowledge.

Next, we examined how the addition of RNA secondary structure information affects both predictive and runtime performance of pysster. Since secondary structure is represented as a string (over the four-letter $[H, I, M, S]$ alphabet), we trained models that either utilized only sequence information, only structure information or both. To incorporate the fact that structure strings usually show long runs of the same character and potentially favor longer motifs, we measured the runtime of a three-model grid search for all input data (kernel length 8, 16 or 24). For every input type the grid search was repeated 50 times with different data splits and predictive performances are based

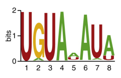
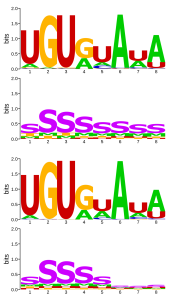
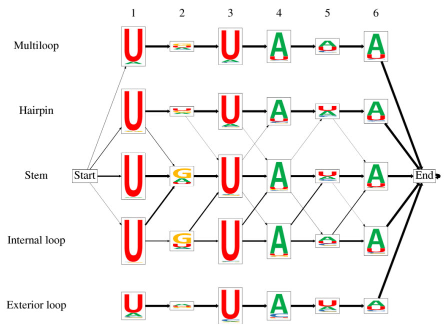
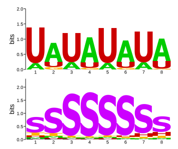
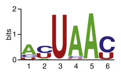
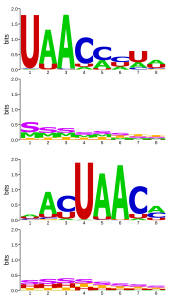
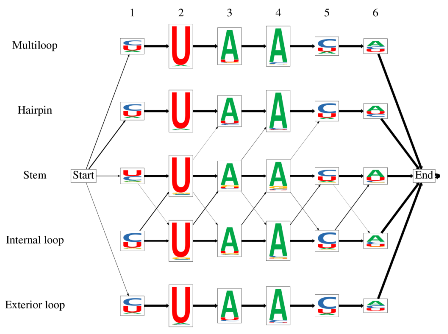
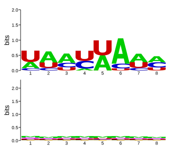
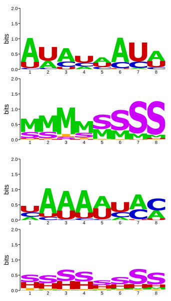
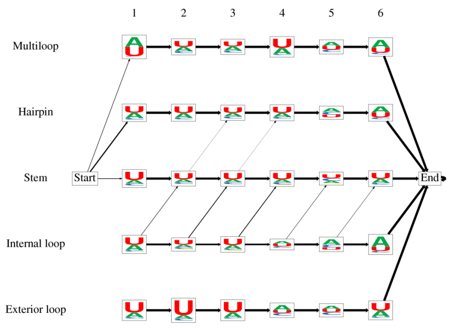
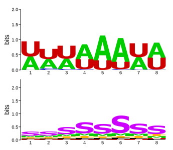
RBP & LITERATURE	PYSSTER (TOP 2)	SSHMM	GRAPHPROT
PUM2  [90]			
QKI  [90]			
TAF2N AU-rich stems [91]			

Table 4.6: **Motif Comparison Of pysster, ssHMM & GraphProt.** Pysster produces a user-defined number of motifs and only the two motifs with the highest importance score showing a local enrichment near the sequence center are shown in the table. GraphProt only learns a single motif per classifier and does not report any additional information. ssHMM, an unsupervised hidden Markov model-based motif finder, can learn a motif from the protein of interest class alone and in its visualization the thickness of the arrows indicates the most likely structural path through the sequence motif.

on the top-performing models only. The results shown in Figure 4.19 indicate that structure-only models consistently deliver the lowest predictive performance, while sequence-only models deliver the highest. The joined sequence/structure encoding into 16-dimensional one-hot encoded inputs tends to achieve slightly lower performance than the sequence-only baseline, similar to what we observed in the DNA shape "ABCD" string discretization. For example, the RBP PUM2 achieves a median auROC of 0.58 with structure-only input, 0.957 auROC with sequence-only input and 0.955 auROC with the joined 16-dimensional input. This trend holds true for all six tested RBPs and the importance of the sequence in the joined encoding seems to dominate the structure information, as the predictive performance is not simply the average of the respective separate models.

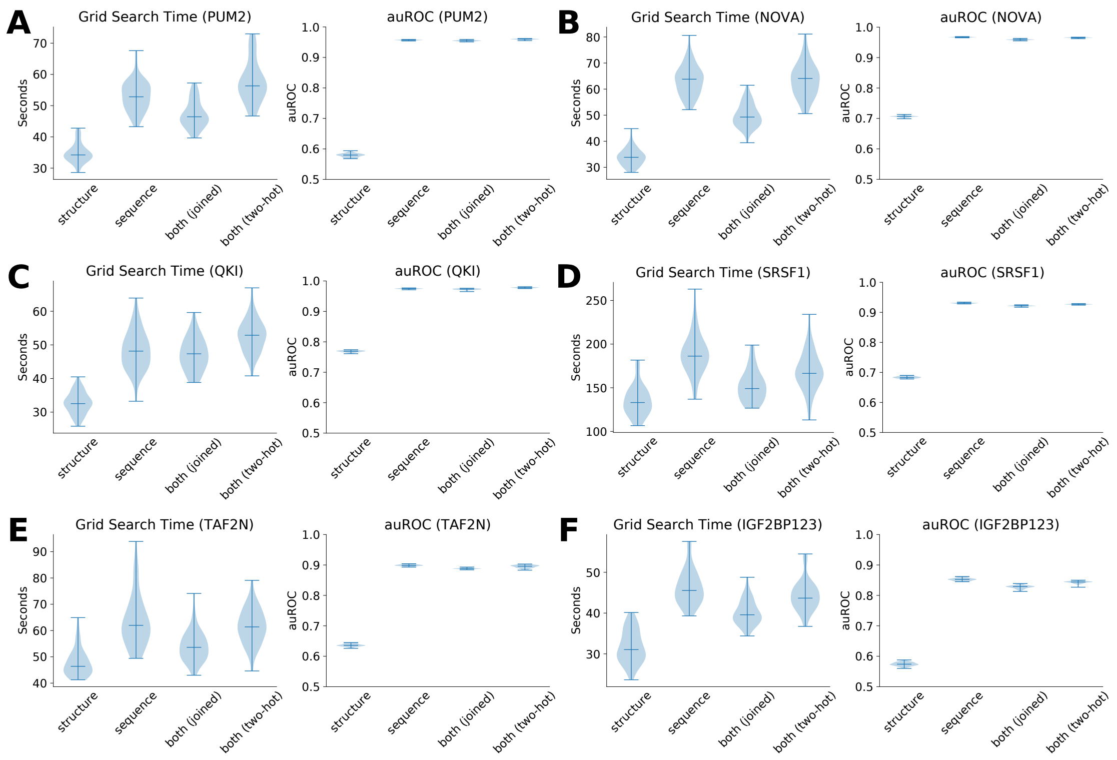


Figure 4.19: **Performance Effects Of RNA Secondary Structure.** Structure-only, sequence-only and combined inputs (joined 16-dimensional one-hot encoding and eight-dimensional two-hot encoding, see main text below) were tested to assess model performance for six different proteins. Per input, 50 grid searches were performed on different data splits (on the GPU). While predictive performance (auROCs) is very stable, grid search runtime varies substantially due to early stopping.

With respect to the training runtime, one can observe that, even though structure-only and sequence-only models use the same amount of input, structure-only models converge considerable faster (e.g. PUM2: 34 seconds using only structure, 52 seconds using only sequence, see Figure 4.19). Given the low predictive performance results of structure-only models, this might be due to the low amount of "signal" in the data which hinders validation loss improvements and consequently leads to fast early stopping (experimenting with relaxed early stopping did not improve the predictive performance of structure-only models). Interestingly, models using the considerably larger 16-dimensional joined input exhibit a lower training runtime than the sequence-only baseline models for all proteins (e.g. PUM2: 46 seconds versus 52 seconds baseline). We already observed a similar trend for the DNA shape discretization encoding and decided to explore this in more detail. To this end, we additionally tested a simple "two-hot" encoding for our RPBs, that is, we separately created four-dimensional one-hot encodings for sequence and structure strings and concatenated these into $8 \times n$ input matrices. In terms of training time, this input encoding is slower than the 16-dimensional encoding and comparable to sequence-only models (e.g. PUM2: 56 seconds versus 52 seconds baseline, see Figure 4.19). The predictive performance of two-hot encoded models is slightly higher than results from the joined encoding (e.g. PUM2: 0.96 auROC versus 0.955 auROC).

It is hard, or likely not possible at all, to quantify why the 16-dimensional input encoding leads to faster training convergence compared to four-dimensional or eight-dimensional inputs. Models based on the 16-dimensional inputs usually show the lowest predictive performance (excluding structure-only models), however, differences are marginal and manifest only in the second or third decimal positions of auROC measurements. Differences in training runtimes are more noticeable, and therefore arguably more relevant in practice, where data sets can become much larger or grid searches more comprehensive than in our examples. Accordingly, we found the joined input encoding to represent a convenient middle ground.

4.5 Discussion

In this chapter we presented the main capabilities of pysster, our open-source Python library for training and interpretation of CNNs on biological sequence data. We demonstrated that CNN models possess a high predictive performance and that various feature visualization and interpretation methods can deliver biological insights for the problem and data at hand. In addition to using only sequence information, our models can handle arbitrary, supplemental data in a way that still allows for visualization and interpretation, which we exemplified by using DNA shape and RNA secondary structure information. Overall, by providing an easy-to-use programming interface and automated hyperparameter tuning for model training (see code examples in Section A.3), this software library is intended to simplify the usage of deep learning models for other researchers.

One of the fundamental, and still not fully solved, problems that biological sequence classification methods try to solve is the prediction of protein-DNA and protein-RNA interactions. TF and RBP binding sites on DNA and RNA, respectively, are usually short and therefore expected to be found all over the genome. Since we used simple sequence-only models, supplemented by structure/shape information, this has implications for the usefulness of said models in practice, because they will predict all potential binding sites, irrespective of whether they are actually bound in a given cell and at a given time. Binding also depends on whether the DNA is accessible in a specific cell type and under the given conditions and it depends on whether co-binding proteins are available, the latter being valid for both DNA and RNA. To get meaningful predictions in practice and to avoid false positives, these simple models have to be further supplemented with other information such as DNA accessibility and correlation with binding sites for other proteins. Some data, for example experimentally validated accessibility and binding signals of relevant proteins, can be directly added to the input of our models analogously to shape information. However, since predictions are usually used to tackle the absence of experimental data (or to pinpoint effects that are worth to be experimentally validated) it also likely requires researchers to merge and overlap results of multiple predictive models.

One of the types of additional information that promise to improve results of sequence-only models are DNA shape and RNA secondary structure. Nonetheless, our results found no meaningful predictive performance improvements when adding such information. This might be due to some inherent property of shape/structure data, due to methodical issues of neural networks or due to data encoding before model training. Both shape and structure predictions are purely based on sequence and strongly correlated with sequence. It is therefore not completely unexpected that they do not add actionable information to the network and might even lead to a loss of predictive performance. Observations similar to ours made by previous studies point into that direction. For example, the iDeepS authors [72] also experimented with augmenting sequence-only RBP deep learning models with secondary structure information. In their case, they used a multimodal model, that is, they separately applied convolutional neural layers to sequence and structure strings before merging of results and further application of recurrent and dense layers. Compared to a sequence input-only variant of the described architecture, their predictive results likewise showed no practically meaningful differences (both positive or negative) for the tested proteins (Figure S3 in [72]). GraphProt, representing a very different machine learning approach, also generally reports little changes, however, it found significant improvements for a collection of RBPs (Figure 6 in [89]). IGF2BP is one of those proteins and was also tested by us during this chapter, but our models did not improve by using secondary structure data. This might be due to our much higher baseline performance, but given that we tested multiple input encodings and that the iDeepS authors report similar results for their neural network architecture, this points towards CNNs not taking full advantage of the available structure information. Thus, further research is required and looking into alternative network architectures, such as graph convolutional networks (see also Chapter 7), and using experimentally derived structure information that promise more variability than predictions might be interesting starting points.

Regarding DNA shape features, our observations are very similar to the RNA secondary structure case. Shape predictions are again purely based on sequence alone and correlations might influence their effectiveness in improving predictive model performance. Using our input encoding and dinucleotide shuffled backgrounds for TF binding prediction we found small predictive performance improvements for all tested proteins. DNA shape is relatively little studied, compared to RNA secondary structure, but our observation is in line with a previous study that used decision trees and a gradient boosting classifier to examine different input encodings that combined sequence and shape information. They likewise used a dinucleotide shuffled background and pre-defined position frequency matrices to find the regions of input sequences that were most similar to the position frequency matrix before classification. Regions were then one-hot encoded and shape features were added to receive matrices analogous to our input matrices and results only showed very small improvements compared to sequence-only models (see "4-bits" encoding in Data S3 in [92]). However, replacing the one-hot encoding of bases with position frequency matrix scanning scores led to significant classification improvements of more than 0.1 area under the precision-recall curve for individual TFs (see "PSSM" encoding in Data S3 in [92]). To be fair, using pre-defined motifs for classification (a single motif in the cited case) substantially shifts the difficulty of the problem towards the motif finding and scanning, but their results also show, as in the RNA secondary structure problem, that CNNs might not take full advantage of the data or that the data encoding is not appropriate and that further research is required to improve predictions.

Overall, applying deep convolutional networks to biological sequence data yields well-performing models from which extensive interpretations can be derived, as presented in this chapter. Given that they also can be efficiently trained on GPUs using large amounts of input data, they present themselves as a promising method to tackle large-scale projects. To this end, the next chapter of the thesis will cover such a project and will detail the prediction of RBP binding sites for 100 proteins in a cell line for which experimental data is not available. During this project, the ability to visualize what exactly the networks are learning helped us to be aware of potential biases in the training data to tune model predictions.

5 Predicting RNA-Binding Protein Binding Across Cell Lines

5.1 Motivation

Predictive models are often used to substitute for missing experimental data. While having experimentally validated data is, of course, the ideal to strive for, monetary costs of experiments and time constraints can be prohibiting factors. Concerning binding of TFs and RBPs, ENCODE and other public databases provide raw experimental and already computationally processed data that is free to use, however, especially for RBPs the available data are still very limited. At the time of writing, ENCODE provides eCLIP data for more than 150 proteins, but only for the HepG2 (human liver cancer) or K562 (human leukemia) cell lines. For one collaborative project we undertook during the PhD time, we were interested in the binding of RBPs in the MCF-7 (human breast cancer) cell line. Specifically, we were interested in interactions between RBPs and long intergenic non-coding RNAs (lincRNAs). LincRNAs represent a subset of the lncRNA class that does not overlap with protein-coding gene annotations and lincRNAs perform a wide range of functions through interactions with DNA, proteins and other RNAs (more details about their functions can be found in Section 7.1 and [93, 94]).

The most important motivation for this project was the fact that lincRNAs are expressed in a very cell line-specific fashion, much more so than protein-coding transcripts [94]. This means that many lincRNAs expressed in MCF-7 cells are simply not expressed in HepG2 and K562 cells for which eCLIP data is available. As there are so far no public eCLIP data available for the MCF-7 cell line, we decided to train predictive models based on the ENCODE eCLIP data to still get information about protein-RNA interactions for these transcripts. The main objective of this RBP binding prediction task was the training of well-performing models for a large collection of RBPs. In addition, since many eCLIP data sets have only been published recently and since consequently not much is known about many RBPs, using pysster's visualization options to learn more about protein-RNA interactions for specific proteins was a secondary objective. Therefore, this chapter describes the iterative model training and fine-tuning process that was needed to achieve high predictive performance and interpretable models in practice. The predictive models trained in this chapter represent a small part of a bigger project led by Evgenia Ntini tackling the functional classification of lincRNAs and a summary of the broader biological context of the complete lincRNA project can be found in Section 7.1.

5.2 Data Sources & General Data Preparation

Since 2016, the Gene Yeo lab is publishing eCLIP data as part of the ENCODE consortium for an ever-growing amount of RBPs in HepG2 and K562 cell lines. To date, data sets for 161 proteins are available and the data used in this chapter were downloaded from ENCODE on September 6, 2018 and comprise 100 RBPs known to be expressed in the MCF-7 cell line. Table A.2 shows the ENCODE accession numbers and original cell lines for all RBPs. Most RPBs were only available for either HepG2 or K562, but in case an RBP was available for both cell lines we used the cell line with the higher data quality (as indicated by the ENCODE audit categories) or HepG2 by default.

For each RBP, we downloaded two bed files containing the called peaks for the two available biological replicates. From these two bed files, we selected all peaks that overlapped with peaks from the respective other replicate by at least one base. Subsequently, peaks with a log-fold enrichment over the input control sample smaller than two were removed. As suggested by Dominguez et al. [26], the 5′ end of each peak was considered as the exact protein binding site. GENCODE [95] gene annotations version 24 for the GRCh38 human genome assembly were used to summarize binding locations and to identify transcripts that overlap with protein binding sites.

5.3 Predicting RBP Binding Sites

In the following, we will present multiple model training approaches using different input data to obtain increasingly better performing models. We will show that models that appear to perform well in artificial training and test settings are not necessarily the models that perform well in practice and that it is important to design the training data such that it represents the "real world" as close as possible. In this case, "real world" means that we want to predict the probability of being bound by a given RBP for every base of a transcript.

While a high predictive performance is the main objective of the project, we will also show that our neural network feature visualizations can help to better understand RBP biology and that they can be used to detect potential technical biases in the eCLIP data. Throughout this chapter, results will be showcased using a small selection of proteins for the sake of brevity. Results for the final models and all 100 proteins can be found in the appendix and will be referenced accordingly.

5.3.1 *Binary Classification Models*

To obtain baseline models and to get familiar with the data, we first trained CNN models performing a binary classification. For each protein we trained a model that is able to distinguish between protein binding site sequences (the “positive” class from now on) and background sequences (the “lincRNA” class). The positive class consists of sequences of length 400 centered at protein binding sites (5′ end of peaks) that overlap with annotated genes. Taking long flanking regions of binding sites into account improves the predictive performance of models, but at the same time negatively affects training times. Given the amount of RBPs and sequences we need to process, we found sequences of length 400 to strike a reasonable balance. Similarly, we used a maximum of 50, 000 sequences for the positive class if more than that amount passed the log-fold enrichment over the input threshold.

For the background class, and to obtain balanced classes, we selected an equal amount of sequences of length 400 that we randomly sampled from lincRNAs that overlap with a binding site of the protein of interest. These sampled sequences were chosen such that they do not overlap with sequences from the positive class. The positive class could not be restricted exclusively to lincRNA binding sites, as only very few sites would haven been left ($< 1,000$) for most proteins, prohibiting the training of effective deep learning models. Note that we do not make use of secondary structure information in any of the models presented in this chapter. As shown in the previous chapter, use of RNA secondary structure predictions likely does not increase the predictive performance of models. In addition, prediction of secondary structure has to be performed for every sequence of length 400 individually, because predictions for full-length transcripts are very unreliable. Given the amount of sequences used throughout this chapter, prediction of RNA secondary structures would have been the computational bottleneck.

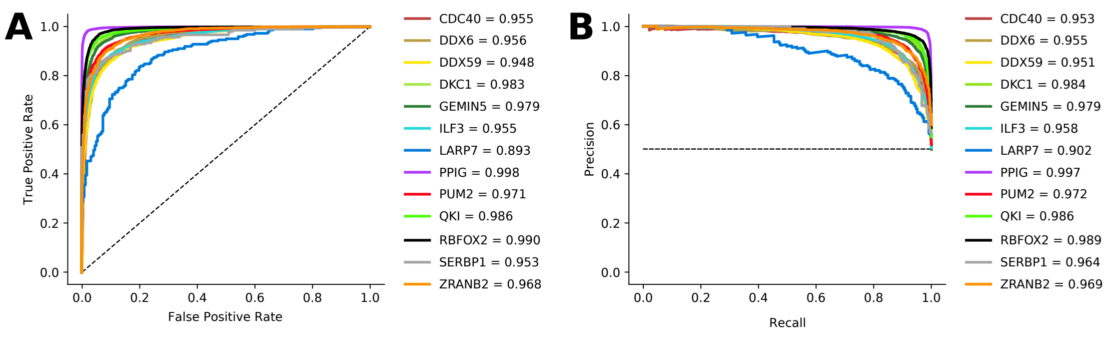


Figure 5.1: **Performance Of Binary Classification Models.** auROC values in **(A)** and auPRE values in **(B)** computed on held-out test data indicate a high predictive performance for all proteins. Balanced classes have been used for all proteins and dotted lines illustrate the performance of a random classifier.

We now used the collected sequences (randomly split into 70% training, 15% validation and 15% held-out test) to train pysster models with a small grid search testing models with three convolutional layers, kernels of length 12, 18 or 24, 150 or 300 kernels per layer and default parameters otherwise. Figure 5.1 shows the predictive performance results on the held-out test data for a collection of proteins. auROC values are usually very high (e.g. up to 0.990 for RBFOX2) and auPRE values are comparable, because balanced classes were used.

5.3.2 *RBP Sequence Motif Visualizations*

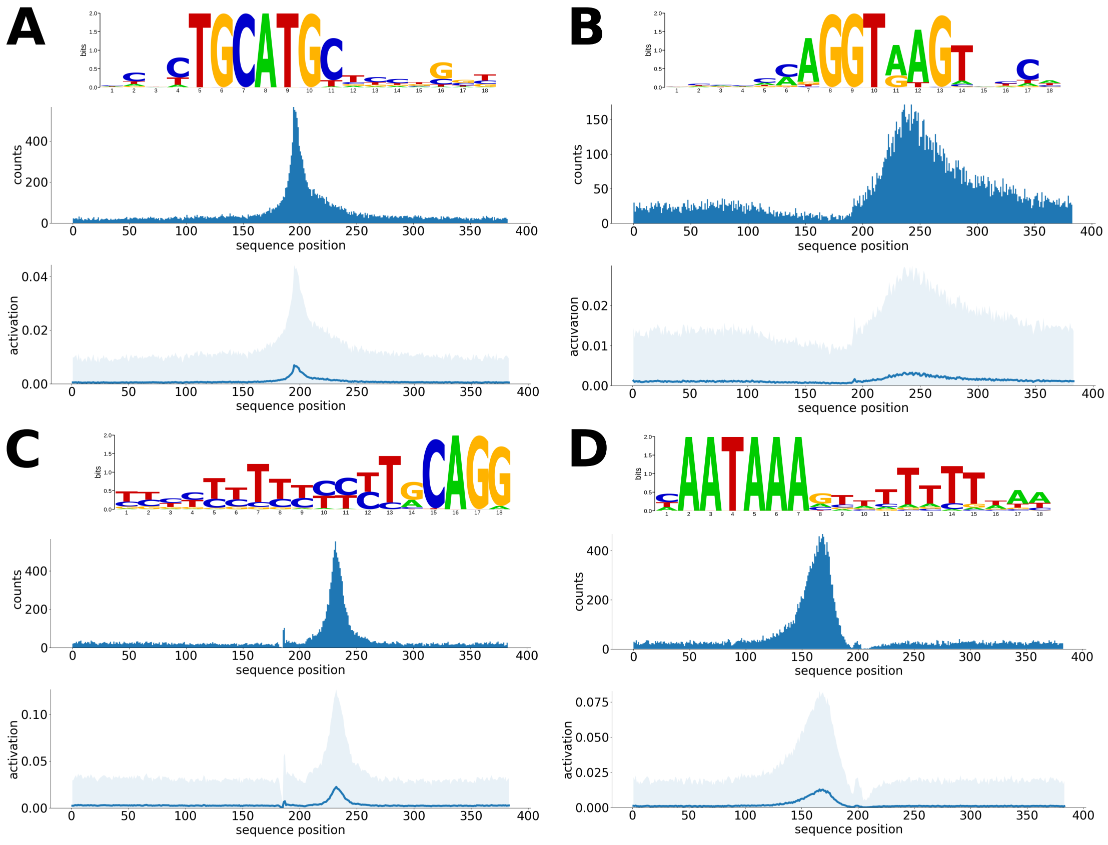


Figure 5.2: **Example Kernel Visualizations.** Many first-layer kernels learned to recognize known motifs that show specific positional enrichment relative to the protein binding site (sequence position 200, enrichment plots taken from the positive class). **(A)** shows the known RBFOX2 binding motif. **(B)** and **(C)** show the canonical 5′ and 3′ splice site motifs from the PPIG and SF3B4 models, respectively. **(D)** displays the polyadenylation signal learned from the CSTF2 model.

To investigate the high predictive performance we subsequently visualized the kernels of the first convolutional layer of all models and Figure 5.2 illustrates some example motifs and their positional enrichment within positive class sequences. Figure 5.2A, for example, shows a kernel that learned to recognize the well-known RBFOX2 binding motif TGCATG [96]. The respective positional enrichment plots show that the motif predominantly starts at sequence position 200, as expected. However, only very few RBP data sets show such unique motifs. For the majority of RBPs our models do not show unique motifs with precise local enrichments, but visually inspecting all kernels rather indicates that many RBPs seem to recognize the general sequence compositions, such as GC-rich or GT-rich sequences, in the broader area around sequence position 200. Moreover, our models learned canonical splice site motifs [97] for many RBPs. These boundary regions between exons and introns (also called 5′ splice site) and introns and exons (3′ splice site) possess specific consensus sequences and many RBPs seem to bind locations relative to splice sites. PPIG, for instance, binds upstream of the 5′ splice site (Figure 5.2B), indicating preferential binding of exons, while SF3B4 binds upstream of the 3′ splice site (Figure 5.2C), indicating preferential binding of introns. Indeed, looking at summary statistics of binding locations affirms that 89% of PPIG binding sites are located in exons and that 67% of SF3B4 binding sites are located in introns (see Figure A.4). Both proteins are known to be part of the splicing machinery [98, 99]. Another general motif that we found for some proteins such as CSTF2 is the polyadenylation signal AATAAA (Figure 5.2D) which marks transcripts to receive a poly(A) tail as a post-transcriptional modification. CSTF2 is a factor known to be involved in the process by binding downstream of the polyadenylation signal [100, 101], which our kernel visualizations reflect as well.

Overall, we found strong motif signals or general sequence patterns that distinguish positive sequences from background sequences for most RBPs. We compiled a list summarizing found motifs for all RBPs in Table A.3. This table also indicates whether we found potential T-rich and G-rich bias motifs which will be discussed in the next section.

5.3.3 *Using Visualizations To Detect Biases*

The motifs highlighted in the previous section are not the only motifs we found during the kernel visualizations. We also consistently found T-rich and G-rich stretches showing a specific signal across many RBPs. For 63 out of 100 proteins our models learned kernels recognizing long T-rich motifs that are located immediately upstream of the protein binding site and that are often slightly depleted on top of and downstream of the binding site (Figure 5.3A). In addition, T-rich motifs do not co-occur with the actual binding motif. For 53 out of 100 proteins we saw kernels learning long G-rich sequences located on top of or downstream of the binding site (Figure 5.3B). The exemplary G-rich motif depicted in Figure 5.3B was found in RBFOX2 models. As shown in the previous section, RBFOX2 exhibits a unique binding motif, however, both unique motif and G-rich motif occupy the same location and do not co-occur.

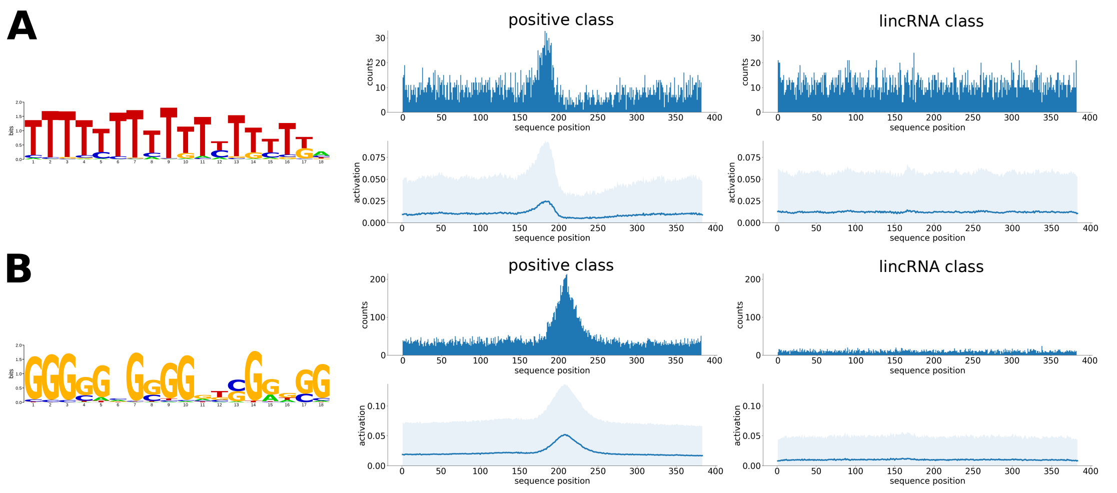


Figure 5.3: **Potential eCLIP Bias Signals.** Many eCLIP data sets show T-rich and G-rich sequence motifs with noticeable similar locations across RBPs relative to the protein binding site. **(A)** depicts an exemplary T-rich kernel from the DDX6 model enriched upstream of the protein binding site at sequence position 200. **(B)** shows a G-rich motif from the RBFOX2 model enriched on top of and downstream of the binding site. Neither T-rich nor G-rich motifs co-occur with the actual binding motif.

Concerning the G-rich motifs, another publication recently reported similar findings and speculates that the motifs either indicate interaction partners of RBFOX2 or crosslinking artifacts in the data [36]. In this publication, the authors also compared in vivo eCLIP data with RNA Bind-N-Seq (RBNS) data, an experimental in vitro protocol that tests the binding of an RBP against a collection of random sequences. While many eCLIP data sets show G-rich motifs, RBNS data for the same RBPs do not. Consequently, the authors labeled such motifs as "eCLIP-only". Whether the motifs indicate real binding of interaction partners in the in vivo eCLIP data or some kind of bias, such as antibody specificity or crosslinking artifacts, is not clear, however.

Concerning the T-rich motifs, it is hypothesized that UV-C light used for protein-RNA crosslinking during eCLIP experiments favors crosslinking to uridines [102, 103]. Corresponding motifs have been named "UV crosslink-associated motifs" (CL-motifs) [103] and described motifs do not necessarily consist of pure stretches of uridines, but are always uridine-rich. Given the literature evidence, it is possible that the motifs learned by our models show said UV-C bias, but the strength of the signal is still surprising, as we found these motifs in highly enriched peaks and not just in the input sample data.

In any case and irrespective of the source of both G-rich and T-rich motifs, it is clear that our CNN models heavily focus on the resulting motif signals which might lead to spurious and false predictions when we actually scan new transcripts for binding sites.

To this end, the next section tries to tackle this problem by introducing an additional class into the models.

5.3.4 *Multiclass Classification Models*

CNNs are supervised machine learning methods and as such the training process focuses on signals in the data that are able to distinguish the given classes. Accordingly, bias signals can distract the model from the actual signals that we would expect the model to focus on. As the bias motifs are not part of the lincRNA background class, this is the case for our models. One way to diminish the impact of bias is to introduce bias into the background class as well. Thereby, the signal that arises from the bias is not able to distinguish the classes anymore and the model should, theoretically, focus on different aspects in the data instead. Here, we decided to introduce a second background class (the "RPB" class from now on) to capture the influence of bias motifs. The RBP class consists of randomly selected binding sites from other proteins that do not overlap with the sequences from the positive class. Analogous to the positive class, sequences from the RBP class are of length 400 and centered at a binding site.

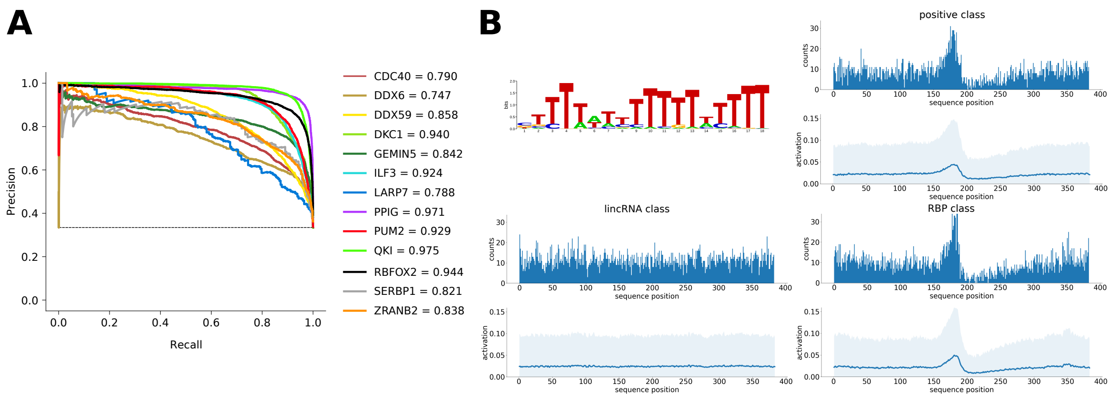


Figure 5.4: **Multiclass Model Evaluation.** Precision-recall curves in **(A)** depict the positive class and were computed in a one-versus-rest fashion. The dotted line indicates the performance of a random classifier. **(B)** shows a kernel from the DDX6 model that learned to recognize T-rich sequences. The kernel shows similar positional enrichment in the positive class and RBP class, highlighting that a collection of randomly selected binding sites contains the potential bias motifs.

We then trained multiclass, single-label CNNs using an equal amount of sequences for all three classes and using the grid search outlined before. Performance results in the form of precision-recall curves on held-out test data are shown in Figure 5.4A. Precision-recall curves are only defined for two classes and the figure therefore shows curves for the positive class computed in a one-versus-rest fashion. Compared to the performance of the binary classification models (average auPRE of 0.97), the multiclass models exhibit a lower performance (average auPRE of 0.87). The visualization of the first-layer kernels indicates that the model is still learning bias motifs (see Figure 5.4B), but both the positive and the RBP class show a comparable positional enrichment for such motifs. The model can, of course, still use the bias motifs to distinguish positive and RBP class sequences from lincRNA class sequences. However, distinguishing positive and RBP class sequences from each other is harder and likely the reason why the predictive performance decreases.

5.3.5 *Measuring Performance In Practice*

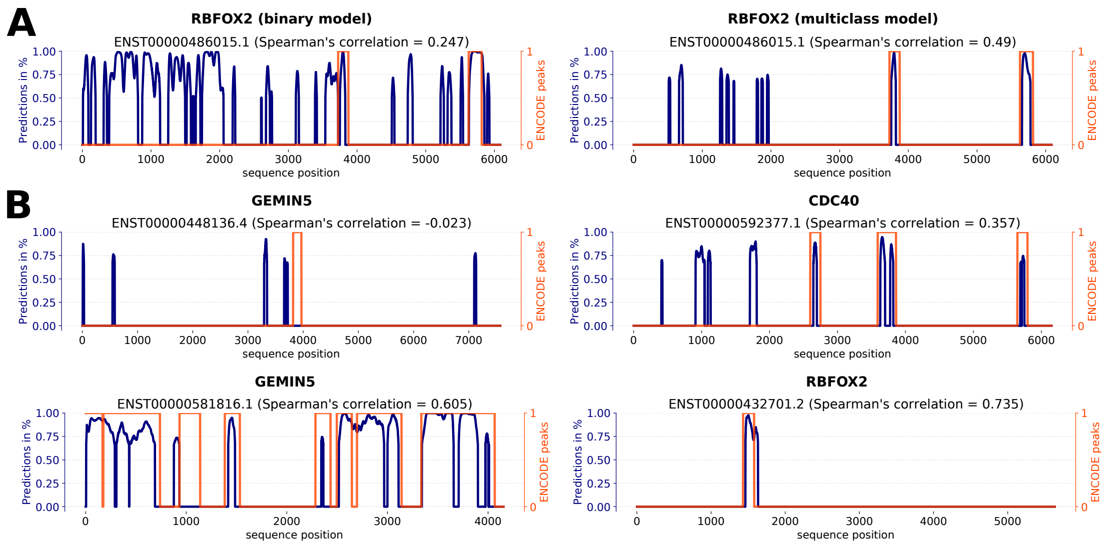


Figure 5.5: **Spearman's Correlation Examples.** When scanning the same transcript with different RBFOX2 models in **(A)**, we see substantially less positive class predictions using the multiclass model. **(B)** shows additional scans to get familiar with different Spearman's correlations between predictions and ENCODE eCLIP signals. To make the blue prediction lines less noisy all values below a threshold have been set to zero (0.5 for the binary models, 0.66 for the three-class multiclass models, see Figure A.5 for details about the effect of thresholds) and Spearman's correlations are computed based on the denoised predictions. The orange ENCODE eCLIP lines do not show the peaks as defined in the bed files, but rather the 5′ ends of the peaks extended by +/- 75 to better reflect the data classification models have been trained on. Only peaks found in both replicates passing the log-fold enrichment over the input threshold as described earlier are used.

To further compare the binary and multiclass models, we subsequently predicted binding for full-length transcripts, as this reflects what we eventually want to do with new transcripts for which no eCLIP data is available. To this end, we "scanned" lincRNA transcripts overlapping with an eCLIP peak of a protein of interest to predict the binding probability for every base using a sliding window approach. Using a window size of 400 and a step size of one, the central base of each sequence was assigned the probability of belonging to the positive class (for positions at the transcript boundaries, flanking regions were used to obtain sequences of length 400).

Figure 5.5A shows the scanning results for a transcript using the binary and multiclass RBFOX2 models. Both models correctly predict the eCLIP peaks that are located within the transcript. However, especially the binary model predicts many additional locations within the transcript to have a high protein binding probability. In fact, this is an observation we made for all binary classification models, as they tend to output a lot of high probability values for the positive class. This is not something we expected given the area under curve metrics of these models. The multiclass models, while having slightly lower area under the curve metrics, produce much less positive class predictions when scanning a full-length transcript. To better quantify this behavior, we measured the similarity between the predictions (represented by a vector containing the predicted probabilities) and the eCLIP signal (represented by an equally-sized vector containing zero/one entries) of a given transcript by computing their Spearman's correlation, that is, we compute the similarity between the blue and orange lines illustrated in Figure 5.5. Spearman's correlation is arguably not explicitly made for such a use case, but it provides a familiar metric (coefficients between -1 and +1 where -1 indicates perfect negative correlation, 0 no correlation and +1 perfect positive correlation) and roughly captures the trend of the relation we would expect when looking at the data by eye. To get a better feeling for the metric, Figure 5.5B shows additional examples for different correlation values.

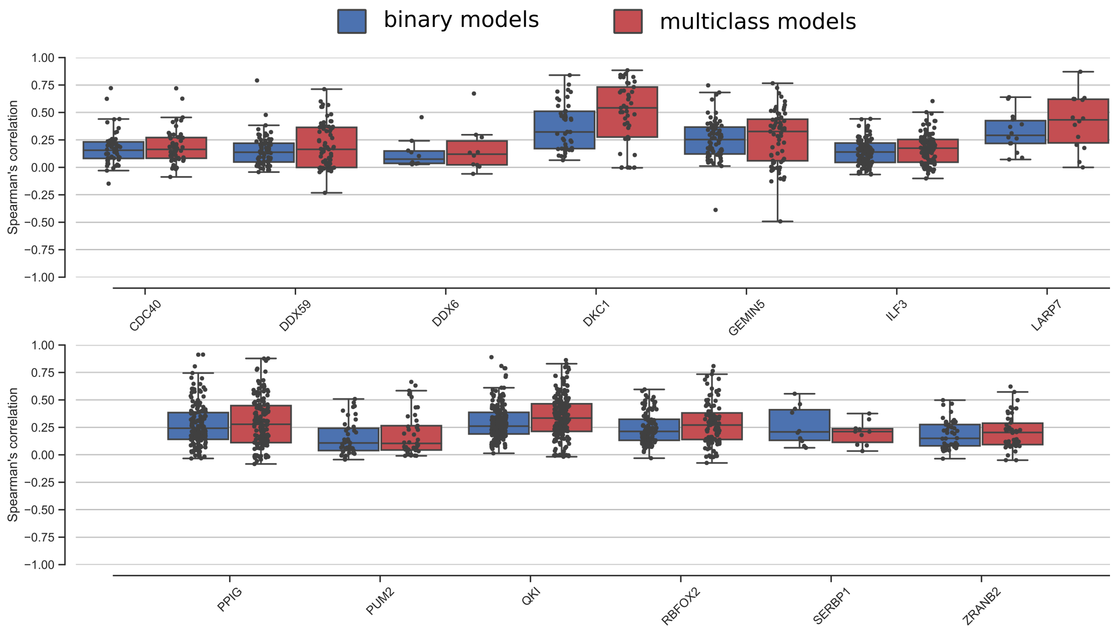


Figure 5.6: **LincRNA Scanning Results For Binary And Multiclass Models.** Each black dot shows the Spearman's correlation for a protein/transcript pair. Correlation values are usually positive and multiclass models tend to provide higher correlations.

Overall, when looking at the general trend for all example proteins and multiple transcripts per protein, the multiclass models outperform the binary classification models in a practical setting (Figure 5.6). They create considerably less predictions for the protein of interest class outside of the desired eCLIP peaks, due to the introduction of the strong bias signals into the background classes of the models. However, predictions are not perfect and probably never should be perfect, as we do not want to predict peaks that contain only bias signal. While multiclass models create less predictions, we still see many predictions outside of eCLIP peaks which we will further improve upon in the next section by training on imbalanced data and by using additional input data beyond pure sequence information.

5.3.6 *Further Improving Predictions in Practice*

So far, models were trained on balanced data, that is, every class consisted of the same amount of sequences. The problem we are trying to solve, however, is not balanced. When scanning a transcript, we do not expect one third (or one half for binary classifications) of the positions to be predicted as belonging to the positive class, because only an average of 3-4% of the bases of a transcript overlap with an eCLIP peak of a given RBP. Yet, machine learning models trained on balanced data tend to deliver "balanced" predictions. When we predict new input with our models, the models of course do not know in advance how many inputs there are going to be, nevertheless, in our experience model predictions show a trend towards the class ratios used during training. To not artificially change the problem at hand, training on class ratios that reflect reality as closely as possible is advisable. We therefore trained models on imbalanced data by increasing the amount of sequences for both background classes by five, e.g. if $50,000$ sequences are available for the positive class $250,000$ sequences are used for both the lincRNA and RBP background class for a total of $550,000$ sequences. Thereby around 9% of the input belongs to the positive class. Class ratios were chosen such that model training for all RBPs could still be completed within a reasonable amount of time (around one week given our single GPU). In the following, models trained that way will be labeled as "imbalanced models".

Furthermore, we explored the usage of input data beyond just sequence information. Binding sites of RBPs are not uniformly distributed across transcripts and many RBPs show strong preferences for binding within either exons or introns (e.g. 92% of FXR2 binding sites are located in exons, 97% of SUGP2 binding sites are located in introns) or for binding close to the transcription start site (TSS) or transcription termination site (TTS) (e.g. NCBP2 predominantly binds close to the TSS, PUM2 binds close to the TTS). Figure A.4 shows a summary of these location information for all RBPs. Supplying these information to the models likely increases their predictive performance.

In the previous chapter, we already showed how pysster models can use additional data on a per-base basis (DNA shape information), but our models are also able to utilize additional data on a per-sequence basis such as the described binding site locations. Additional data on a per-sequence basis is integrated into the models by using additional neurons in the first dense layer (see Figure 5.7). Categorical features (e.g. exon/intron location) are zero/one encoded using an amount of neurons equal to the amount of categories, while continuous features (e.g. distance to the TSS/TTS) use a single neuron. For our models, we use the two-category exon/intron feature (location of sequence position 200) and the distance to the TSS/TTS as additional inputs (distance of sequence position 200, normalized to the transcript length such that zero indicates overlap with the TTS and one overlap with the TSS). In the following, models trained on both imbalanced data and utilizing additional input are referred to as "full models".

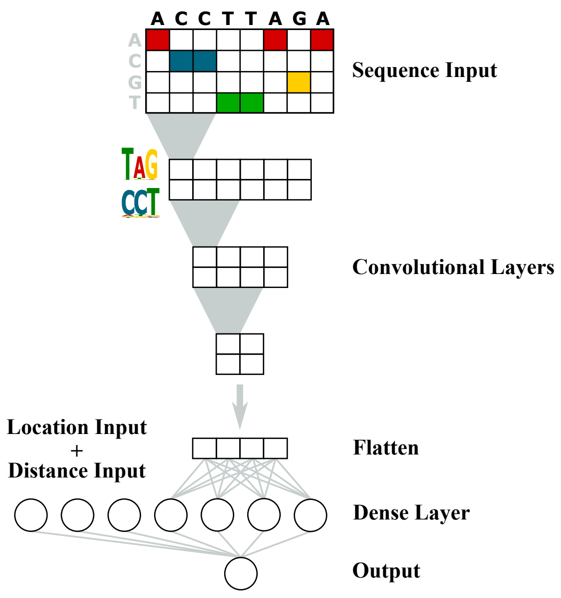


Figure 5.7: **Adding Additional Data On A Per-Sequence Basis.** Additional data, in our case exon/intron location (categorical feature) and distance to the TSS/TTS (continuous feature), are introduced into the model separately from the main sequence input by creating additional neurons in the first dense layer of a network. For categorical features, the number of new neurons is equal to the number of categories and categories are zero/one encoded.

Table 5.1 shows the auPRE results on held-out test data for the imbalanced and full models, as well as for the previously presented binary and multiclass models for comparison. Both imbalanced and full models use three classes and values therefore again represent the auPRE of the positive class computed in a one-versus-rest fashion. With an average of 0.69 the imbalanced models possess lower auPRE than the balanced models, which is not unexpected, as precision-recall curves are sensitive to class imbalance and shift the random baseline performance accordingly. Supplementing the imbalanced models with the additional input features (full models) restores some of the performance and results in an average auPRE of 0.71. The performance results of the full models for all 100 proteins and all three classes can be found in Table A.4.

RBP	BINARY MODELS	MULTICLASS MODELS	IMBALANCED MODELS	FULL MODELS
CDC40	0.953	0.790	0.391	0.497
DDX59	0.951	0.858	0.608	0.645
DDX6	0.955	0.747	0.558	0.518
DKC1	0.984	0.940	0.889	0.878
GEMIN5	0.979	0.842	0.495	0.609
ILF3	0.958	0.924	0.795	0.786
LARP7	0.902	0.788	0.769	0.676
PPIG	0.997	0.971	0.673	0.671
PUM2	0.972	0.929	0.801	0.849
QKI	0.986	0.975	0.915	0.913
RBFOX2	0.989	0.944	0.757	0.752
SERBP1	0.964	0.821	0.713	0.746
ZRANB2	0.969	0.838	0.610	0.657

Table 5.1: **auPRE For All Model Variations.** Held-out test data measurements for binary and multiclass models were previously presented in Figure 5.1 and Figure 5.4, respectively. All values in all columns represent the auPRE of the positive class computed in a one-version-rest fashion.

Interestingly, when looking at the practical transcript scanning results of imbalanced and full models, we observe a much better performance compared to binary and multiclass models (Figure 5.8). Spearman's correlations are usually substantially increased and the full models provide the best match between predictions and eCLIP signal for most proteins. While we do not try to interpret the absolute correlation values, their steady increase for the different model variants suggests to us that the full models indeed deliver the best performance in practice out of all tested variants. Full model scanning results for all 100 proteins are illustrated in Figure A.6.

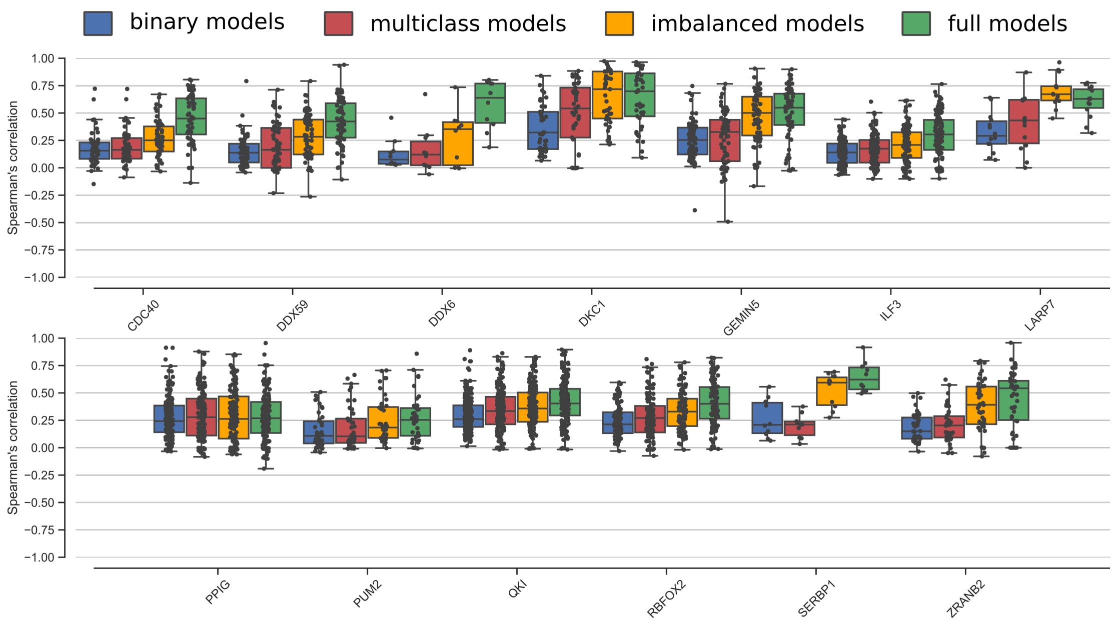


Figure 5.8: **LincRNA Scanning Results For All Model Variations.** Each black dot shows the Spearman's correlation for a protein/transcript pair. Scanning results for binary and multiclass models have previously been shown in Figure 5.6.

5.4 Discussion

In this chapter, we described our process of training CNN models to predict RBP binding sites based on eCLIP data. Starting with simple binary classification models we showed how to continuously improve model performance by adding another class that captured data bias, by adding additional input features beyond sequence information and by making use of the rarity of the signal we try to predict.

Our main conclusion is the observation that models that appear to deliver the best performance according to often-used metrics such as auROC and auPRE do not necessarily deliver the best performance in the practical application. Therefore, one should always conduct tests in a setting that reflects the practical application and appropriately designing the training data to simulate the reality of the problem as close as possible is very important in this regard as well. This includes class balance which is one of the big issues in machine learning. If the amount of minority class samples is not high enough to successfully train a classifier, then artificially balancing data can be taken into consideration. In our case, availability of positive class samples was not a limiting factor and our results show that approximating the class balance of the practical problem leads to the best performing classifiers. We mentioned in Chapter 4 that our pysster models make use of class weighting by balancing out the loss function values of individual inputs. This additionally supports the process by shifting the predicted probability distributions such that the intuitive classification thresholds can be applied (e.g. 0.5 for two classes or 0.66 for three) and makes things easier to interpret for human eyes.

Producing models with high predictive performance was the main objective of the project, but the visualization options we implemented for our pysster CNN library also produce biological insights for individual RBPs as a byproduct. As demonstrated, these visualizations can be used to detect potential bias in the data of which eCLIP data unfortunately appears to have a substantial amount of. If we would have used a different classifier, without comparable visualization options, we might have not been aware of the bias which would have resulted in worse classification models. As mentioned in the beginning, the presented RBP prediction task is part of a bigger project described in more detail in Section 7.1.

6 Summary & Conclusion

Understanding the molecular mechanisms that control the regulation of genes and transcripts has been one of the main goals of biological research. Transcription factors and RNA-binding proteins are key regulatory factors in this regard and while individual genes and transcripts have been studied in great detail for decades, the introduction of genome-wide experimental protocols such as ChIP-seq and CLIP-seq has notably improved the ability to derive broader regulatory rules. The availability of large-scale data measuring protein-DNA and protein-RNA interactions, amongst others, also allows for more effective computational modeling of said measurements and consequently, the field of machine learning has become established in our area of research. Compared to performing experimental protocols, the creation of machine learning models is often cheaper. Models are therefore used to provide predictions for cell lines or cellular conditions for which experimental data is not yet available, to guide experiments towards promising targets or to find patterns in the data that can be used to gain biological insights.

Lately, neural network-based deep learning models have shown promising results, especially together with GPU training, which enabled the analysis of very large data sets within reasonable time frames. By freely combining different layer types and other building blocks, deep learning models can be architectured in a very flexible way and they allow for the integration of many different input data types into a single model. One can essentially customize models to fit specific inputs, while other machine learning methods often require the input to be customized to fit specific models. However, all of this leads to a certain complexity that makes model selection and interpretation hard and deep learning certainly no silver bullet.

To this end, we contributed to the research field by creating a software library that enables researchers to more easily apply deep learning methods to biological sequence data. In Chapter 4 we presented pysster, our Python library that achieves this task by providing CNN models that can be trained and evaluated with only a handful of lines of code. Sequence data are the basic input needed for our library, but they can be supplemented with arbitrary other data provided by the user. We showcased the functionality of our library by modeling sequence and structure data (DNA shape, RNA secondary structure) and we provide users of our library with options to visualize what features the trained network has learned. Sequence motifs, structure motifs and positional patterns in the data received that way can shed light on what exactly the network has learned and combined with domain knowledge of the researcher, these visualizations can aid the understanding of biological mechanisms. Our eCLIP-based models presented in Chapter 5 have shown that visualizations can be used to learn about binding properties of individual RBPs and to detect technical bias in the experimental data, which is a common problem. All of the above visualizations and interpretations can of course be derived using different methods as well. However, having models that provide both high predictive performance and extensive interpretation at the same time is very convenient and we therefore showed that deep learning models do not have to be "black boxes".

When we started working on our library, no easy way of training and interpreting CNNs on one's own data was available, but after pysster other packages such as Selene [104] and Janggu [105] have been published, which are also intended as general deep learning frameworks for bioinformatics. As is often the case with "competing" packages, they overlap to some extent, but they also focus on different layer types or interpretation aspects and thus all contribute to the community such that researchers wanting to simply use pre-configured deep learning models are now able to do so. Concerning our own library, there are many directions for further improvements. In our opinion, the two main points that can positively impact the community are: 1) the implementation of attribution methods (e.g. LRP) that enable the interpretation of individual inputs and make pysster a more complete package and 2) the adaption of our package API for the Kipoi model zoo [106, 107], a repository of pre-trained machine learning models for genomics that tries to unify models from different sources to facilitate their application in a consistent manner.

Finally, while the mentioned flexibility of deep learning models is an advantage over other methods, it is simultaneously its biggest weakness, as a very large number of hyperparameters has to be considered when finding the best network architecture. Extensive parameter searches are very slow even on GPUs and so far no method (grid search, random search, bayesian optimization) appears to consistently stand out, degrading the process essentially into trial and error. Model selection was often a pain point for us and exploring more efficient ways to find the best hyperparameters should have substantial benefits for models that try to integrate different data sources to tackle more complex biological problems in the future.

7 Contributions To Other Projects

During the time of the PhD we were involved in other projects not presented thus far. This chapter briefly describes these projects and our machine learning-related contributions.

7.1 Modeling Chromatin-Associated lincRNAs

In Chapter 5 we described the predictive modeling of eCLIP data for 100 RBPs. As mentioned, the resulting predictions are intended to be used for a currently ongoing project led by Evgenia Ntini. This project aims to dissect the functional landscape of lincRNAs, specifically, it aims to better understand chromatin-associated lincRNAs. Until now, lincRNAs are defined as RNAs that are not translated, that are longer than 200 bases and that do not overlap with annotated protein-coding genes [94]. However, since this definition is very broad, it likely covers groups of functionally different RNAs. Indeed, it has been shown that lincRNAs perform a diverse range of functions both in *cis* and in *trans* [108], where *cis* indicates function close to and dependent of the site of transcription and where *trans* indicates function independent of the site of transcription and often far away from it. In our project, we are interested in chromatin-associated lincRNAs, a subset of cis-acting lincRNAs that stay tethered to their site of transcription. The most well-known example of such lincRNAs are enhancer-like RNAs, i.e. transcripts that arise from enhancer regions which then stay tethered to their site of transcription and interact with promoter regions, the transcription machinery or other proteins within pre-formed chromatin loops to upregulate or repress the expression of genes [108]. In other cases, individual lincRNAs have been shown to influence nearby genes through the mere act of being transcripted or spliced (both in sequence and non-sequence dependent ways). Here, the transcripts likely function by increasing the local concentration of transcription and splicing factors which benefits their neighborhood as well, however, the same principle can also lead to transcriptional interference, that is, the lincRNA competes with neighboring genes for the available resources and triggers a repression of the neighborhood [108].

To better understand chromatin-associated lincRNAs, our objective is a genome-wide classification of lincRNAs into distinct degrees of chromatin association and to understand the underlying features that distinguish chromatin-associated lincRNAs and efficiently dissociating lincRNAs. To this end, Evgenia Ntini performed 4-sU metabolic pulse labeling experiments combined with nuclear fractionation and sequencing in

MCF-7 cells. In brief, cells were labeled with a uridine analog (4-sU) that can be targeted to specifically sequence only transcripts that have incorporated said analog. By separately sequencing both the chromatin-associated and the chromatin-released parts of the transcriptome at multiple time points, we can then track how long individual transcripts stay tethered to the chromatin. Based on the sequencing results, we have clustered lincRNAs into groups of slowly and quickly released transcripts and we now aim to computationally classify these groups based on a collection of features such as RBP binding (our predictions from Chapter 5), splicing efficiency, chromatin marks, R-loops and polymerase pausing. Thereby, we hope to find features that can explain why some lincRNAs stay tethered to their site of transcription. This might also lead to a more general mechanistic insight if a broader pattern can be identified. Overall, our contributions to this project are the prediction of RBP binding sites and the computational classification of lincRNA groups.

7.2 Interpretable Prediction Of Cancer Driver Genes Using Graph Convolutional Networks

Another project is concerned with the prediction of cancer driver genes. This project is led by Roman Schulte-Sasse and has been partially published in [109]. The prediction of new cancer driver genes is continuously attempted by many researchers, but cancer being a very heterogeneous disease makes it hard to receive reliable predictions and to pin-point the influence of individual genes. Due to the increase of publicly available cancer-related sequencing data (e.g. The Cancer Genome Atlas [110]), however, it has at least become easier to integrate different data types into computational models. One promising method are graph convolutional networks (GCNs) [111], a deep learning approach based on CNNs. A GCN expands the general idea of CNNs (scanning kernels over the input) to be applicable to graph structures by learning small patterns in the neighborhood of graph nodes. In this project, the graph is represented by a protein-protein interaction network with the goal to predict a "is cancer-related"/"is not cancer-related" label for every node in the network. One advantage of GCNs is the possibility to add arbitrary feature vectors to every node. For this, mutation rate, DNA promoter methylation and gene expression data for 16 different cell types have been collected from public sources to form 48-dimensional feature vectors for every node. Together with a curated list of cancer driver gene labels and negative labels, the GCN was then applied to predict node labels in a semi-supervised fashion (i.e. labels were not available for all nodes in the network). By combining the protein-protein interaction network with the feature vectors, the GCN was able to outperform previous methods such as network propagation algorithms and mutation-based approaches. Our contribution to this project concerns the interpretation of the GCN predictions. We applied the attribution method LRP (as outlined in Chapter 3) to individual genes to explain why genes were labeled as cancer-related or not. Given a gene, LRP redistributes the GCN prediction over all inputs and assigns a relevance to every input. This allowed us to make statements about which -omics features (mutation, methylation, gene expression) were most important and in which cell types and whether certain neighbors in the network had a strong influence on the prediction of the given gene. Our predictions and interpretations were able to validate known cancer genes, thus, making new, so far unknown predictions promising.

7.3 Modeling microRNA Expression Quantitative Trait Loci

Finally, the beginning of the PhD time (November 2015 - June 2016) was used to finalize a project that had already started previously. This project was led by us, has been published in [112] and investigated the transcriptional regulation of microRNAs.

MicroRNAs are a class of non-coding transcripts. They are single-stranded, roughly 22 bases long and post-transcriptionally repress other transcripts through direct binding of complementary sequence regions. Their regulatory functions have been extensively researched, however, the transcriptional regulation of microRNAs themselves was far less well understood at that point in time. We tried to tackle this problem by exploiting expression quantitative trait loci (eQTL), i.e. genomic locations, often single-nucleotide polymorphisms (SNPs), that correlate with the expression levels of transcripts. By integrating publicly available microRNA-eQTL data with microRNA gene annotations, regulatory annotations and transcription factor binding sites, we built logistic regression models to classify microRNA/SNP pairs into eQTL and non-eQTL. We were able to predict eQTL with 85% accuracy (balanced data set) and found eQTL enrichment for specific transcription factors and related promoter and enhancer annotations. Interestingly, many so-called intragenic microRNAs are embedded within introns of bigger genes, called host genes. These intragenic microRNAs were presumed to share the promoter of their host genes. Nevertheless, there has also been evidence that intragenic microRNAs can use separate promoters to be transcribed independently from their hosts. By using microRNA-specific promoter predictions we could show a significant enrichment of eQTL within these alternative promoters, validating their existence once more. By additionally analyzing eQTL of host genes we found many eQTL to be shared between hosts and microRNAs, however, the majority of shared eQTL affect microRNA and host expression differently, e.g. a SNP repressing the microRNA might upregulate the respective host. Ultimately, since many eQTL overlap with locations found in genome-wide association studies (GWAS), we could also show that predictive models which integrate a variety of genomic and regulatory annotations can be used to aid the interpretation of said GWAS results, as most of them are located in non-coding parts of the genome and therefore comparatively hard to interpret.

A

Appendix

A.1 Pysster: Hyperparameters

PARAMETER	DEFAULT	DESCRIPTION
conv_num	2	number of convolutional/pooling layers
kernel_num	30	number of kernels in each conv layer
kernel_len	25	length of kernels
pool_size	2	size of pooling windows
pool_stride	2	step size of pooling operation
dense_num	1	number of dense layers
neuron_num	100	number of neurons in each dense layer
dropout_input	0.1	dropout portion after input
dropout_conv	0.3	dropout portion after pooling layers
dropout_dense	0.6	dropout portion after dense layers
batch_size	128	batch size during training
learning_rate	0.0005	maximum learning rate of Adam optimizer
patience_lr	5	number of epochs without validation loss improvement before halving learning rate
patience_stopping	15	number of epochs without validation loss improvement before stopping training
kernel_constraint	3	max-norm weight constraint
rnn_type	None	"LSTM" or "GRU" (strings) are possible layers
rnn_num	1	number of RNN layers
rnn_units	32	number of output dimensions of each layer
rnn_bidirectional	True	should layers be bidirectional (bool)
rnn_dropout_input	0.2	dropout portion for input connections
rnn_dropout_recurrent	0.0	dropout portion for recurrent connections

Table A.1: **Default Hyperparameters Of Pysster.**

A.2 Pysster: Hyperparameter Tuning Considerations

The preceding table lists all adjustable hyperparameters of a pysster model together with their respective default values. We found the default values to perform reasonable well for a variety of data, but if a grid search is desired not all parameters are equally important. The *conv_num* (range 1-3), *kernel_num* (range 50-300) and dropout parameters (around 0.1 for the input and 0.2-0.6 otherwise) are usually a good starting point as they hold the strongest influence over the predictive performance. If training time is an issue the *learning_rate*, *patience_lr* and *patience_stopping* parameters can be decreased. However, this will most likely sacrifice some predictive performance.

The *batch_size* is set to 128 by default. In addition to our explanations in Chapter 3 (trade-off between runtime and model generalization) it is also important to choose a large enough *batch_size* such that each random batch likely contains at least one sample from every class. Otherwise, training time might increase because of inconsistent loss values and weight updates. Stratified random sampling is another way to accomplish this, however, it is currently not implemented in pysster. Therefore, a *batch_size* of 128 should be large enough to achieve this goal for most classification tasks. In general, adjusting the *batch_size* parameter can influence the training time, because it controls how often backpropagation will be performed, but it therefore also affects the total number of epochs before early stopping takes effect and might not make a real difference in the end. Generally, a considerable amount of trial and error is needed, if one really wants to find the very best parameters for the problem at hand.

For the realization of the gradient descent optimization scheme we use the Adam algorithm (adaptive moment estimation) [113] as implemented in Keras. Adam performs very well in practice, both in terms of training time and predictive performance, and is currently regarded as the standard optimization algorithm for neural networks [114]. Compared to the basic gradient descent scheme presented in the beginning of the thesis, Adam uses individual learning rates for every weight instead of a single global rate and these individual learning rates change over time based on the gradients of previous batches. Further details can be found in [113]. In pysster models, the *learning_rate* hyperparameter therefore represents an upper bound for the adaptive rates Adam chooses internally.

For more advanced users we offer the option to add recurrent neural network (RNN) layers to the network in between the convolutional and dense blocks (see *rnn_...* hyperparameters). Both long short-term memory (LSTM) and gated recurrent unit (GRU) layers are possible options. Recurrent layers are a popular choice for time series and comparable sequential data, as they take long-term relations among data points into account, and they are often combined with convolutional layers. Nonetheless, in our experience they are not beneficial for the predictive performance of our type of input data. At the same time they substantially increase the training time of networks and therefore no model discussed in this thesis makes use of recurrent layers. The interested reader can learn more about them in [54].

Lastly, while the basic architecture of our models consists of a convolutional block followed by a dense block with an optional recurrent block in between, users can change this and completely remove individual blocks from the network by setting *conv_num* or *dense_num* to zero or *rnn_type* to *None* (this is already the default). Thus, it is possible to train models that only contain dense layers, among other things. However, sequence motif visualization is not possible anymore if the first network layer is not a convolutional layer.

A.3 Pysster: Code Examples

The following code demonstrates a typical pysster worklow and minimal example. The full API documentation and further tutorials can be found at `https://github.com/budach/pysster`. In the below code, we will train a simple binary classification model that distinguishes between binding sites of the RBP PUM2 and background sequences (as has been shown in Chapter 4). Therefore, we first import the necessary pysster components: *Data* objects will handle our input data, *Grid_Search* objects will handle training/model selection and functions from the *utils* module will provide small helper functionality. We also need to create an output directory that we will use to store some model outputs later on.

```python
from pysster.Data import Data
from pysster.Grid_Search import Grid_Search
from pysster import utils
import os

if not os.path.isdir("out/"):
    os.makedirs("out/")
```

Basic input data has to be provided by the user in the form of fasta files. We prepared to fasta files containing sequences for the two classes: "pum2_positives.fasta.gz" contains sequences of length 200 centered at PUM2 binding sites and "pum2_background.fasta.gz" contains an equal amount of sequences of length 200 centered at randomly selected binding sites of other RBPs. Both files can be found at `https://github.com/budach/pysster/tree/master/tutorials/data`. Given these fasta files, we can now use pysster to predict and annotate the RNA secondary structure for all sequences. The *utils* module contains a function predict_structures() to do that and given a fasta file, it outputs another fasta file containing both sequences and structures. Internally, the function uses either the ViennaRNA Python bindings or the RNAfold binary, if available in the PATH environment variable. The function is parallelized to speed up structure predictions and can handle both gzipped and non-compressed files (as can all of pysster).

```python
# entries in the output fasta file contain sequence and
# structure strings on separate lines, e.g.:
#
# > header
# CCCCAUAGGGG
# SSSSHHHSSSS
#
# if the annotate parameter is set to False, the output
# contains dot-bracket strings instead

utils.predict_structures(
    input_file="pum2_positives.fasta.gz",
    output_file="pum2_positives_with_struct.fasta.gz",
    num_processes=20,
    annotate=True)
utils.predict_structures(
    input_file="pum2_background.fasta.gz",
    output_file="pum2_background_with_struct.fasta.gz",
    num_processes=20,
    annotate=True)
```

Next, sequences and structures can be loaded into *Data* objects that automatically handle one-hot encoding, data splitting and communicate with the model later on. By default, creating a *Data* object will split data into 70% training, 15% validation and 15% test. This can be changed manually using the train_val_test_split() method if other ratios or reproducible splits are desired (using the seed parameter).

```
# the number of files given through the class_files parameter
# automatically determines how many classes models are going to have
#
# if only a single file is provided, models will attempt a multi-label
# classification and class memberships of sequences have to be encoded
# into the faster headers (see API documentation)

data = Data(class_files=["pum2_positives_with_struct.fasta.gz",  # class_0 later
                         "pum2_background_with_struct.fasta.gz"],# class_1 later
            alphabet=("ACGU", "HIMS"))
data.train_val_test_split(portion_train=0.6, portion_val=0.2, seed=42)
print(data.get_summary())

# print output:
#
#                class_0    class_1
# all data:         5184       5184
# training:         3142       3078
# validation:       1025       1049
# test:             1017       1057
```

With that, we can start training models on the data. The below code performs a small grid search using models with either 1, 2 or 3 convolution layers and with kernels of length 8 or 16, that is, six different models will be trained (all other hyperparameters have default values). By default, the grid search training returns the model with the highest area under the ROC curve on the validation data. If the pr_auc parameter is set to True, the area under the precision-recall curve will be optimized instead. At the end, the model variable holds a pysster *Model* object, which is a wrapper around a keras model.

```
# if a grid search is not desired, one can directly train a single model:
#
# from pysster.Model import Model
# model = Model({"conv_num": 2, "kernel_len": 8}, data)
# model.train(data)

params = {"conv_num": [1,2,3], "kernel_len": [8, 16]}
searcher = Grid_Search(params)
model, summary = searcher.train(data, pr_auc=True, verbose=True)

# verbose training output (shortened):
#
# Epoch 1/500
# 49/49 [==============================] - 2s 34ms/step - loss: 0.8168 - val_loss: 0.6290
# Epoch 2/500
# 49/49 [==============================] - 0s 9ms/step - loss: 0.6684 - val_loss: 0.4997
# Epoch 3/500
# 49/49 [==============================] - 0s 9ms/step - loss: 0.6076 - val_loss: 0.4752
# Epoch 4/500
# 49/49 [==============================] - 0s 10ms/step - loss: 0.5683 - val_loss: 0.4593
# Epoch 5/500
# 49/49 [==============================] - 0s 9ms/step - loss: 0.5530 - val_loss: 0.4470
# Epoch 6/500
# 49/49 [==============================] - 0s 10ms/step - loss: 0.5249 - val_loss: 0.4243
# Epoch 7/500
# 49/49 [==============================] - 0s 9ms/step - loss: 0.5053 - val_loss: 0.4077
# ...
```

The grid search also returns a training summary as a string, that shows the parameter ranges and the optimized metric on the validation data for all parameter combinations.

```
print(summary)

# print output:
#
# # conv_num: [1, 2, 3]
# # kernel_len: [8, 16]
# conv_num kernel_len pre-auc
#        3          8 0.96434
#        2          8 0.96253
#        2         16 0.95091
#        3         16 0.94821
#        1          8 0.92896
#        1         16 0.92499
```

A summary of the final model architecture can be printed as well. The actual keras model can be accessed through model.model, if desired.

```
model.print_summary()

# print output:
#
# _________________________________________________________________
# Layer (type)                 Output Shape              Param #
# =================================================================
# input_1 (InputLayer)         (None, 200, 16)           0
# _________________________________________________________________
# dropout_1 (Dropout)          (None, 200, 16)           0
# _________________________________________________________________
# conv1d_1 (Conv1D)            (None, 193, 30)           3870
# _________________________________________________________________
# max_pooling1d_1 (MaxPooling1 (None, 96, 30)            0
# _________________________________________________________________
# dropout_2 (Dropout)          (None, 96, 30)            0
# _________________________________________________________________
# conv1d_2 (Conv1D)            (None, 89, 30)            7230
# _________________________________________________________________
# max_pooling1d_2 (MaxPooling1 (None, 44, 30)            0
# _________________________________________________________________
# dropout_3 (Dropout)          (None, 44, 30)            0
# _________________________________________________________________
# conv1d_3 (Conv1D)            (None, 37, 30)            7230
# _________________________________________________________________
# max_pooling1d_3 (MaxPooling1 (None, 18, 30)            0
# _________________________________________________________________
# dropout_4 (Dropout)          (None, 18, 30)            0
# _________________________________________________________________
# flatten_1 (Flatten)          (None, 540)               0
# _________________________________________________________________
# dense_1 (Dense)              (None, 100)               54100
# _________________________________________________________________
# dropout_5 (Dropout)          (None, 100)               0
# _________________________________________________________________
# dense_2 (Dense)              (None, 2)                 202
# =================================================================
# Total params: 72,632
# Trainable params: 72,632
# Non-trainable params: 0
```

Given a trained model, we can now predict new data. In our case, we predict the held-out test data and the model.predict() function returns a numpy array of shape (number of sequences, number of classes) containing predicted probabilities.

```
# the group parameter controls what portion of the data is predicted
# possible values are "train", "val", "test" and "all"

# "all" permits the prediction of all sequences in a non-random order (the
# order in which sequences were loaded from the fasta files) and is useful
# if a completely new Data object not used for model training should be
# predicted

predictions = model.predict(data, group="test")
print(predictions)

# print output:
#
# [[0.9628826  0.03711742]
#  [0.29889268 0.7011074 ]
#  [0.03372031 0.9662796 ]
#  ...
#  [0.40860504 0.59139496]
#  [0.93532455 0.06467547]
#  [0.89703226 0.10296768]]
```

To compare predictions to the ground truth, labels for the held-out test data can be retrieved from the respective *Data* object. The result is again a numpy array of shape (number of sequences, number of classes).

```
labels = data.get_labels(group="test")
print(labels)

# print output:
#
# [[1 0]
#  [0 1]
#  [0 1]
#  ...
#  [0 1]
#  [1 0]
#  [0 1]]
```

Given predictions and labels, we can use a *utils* function to get a predictive performance overview.

```
print(utils.get_performance_report(labels, predictions))

# print output:
#
#               precision    recall  f1-score   roc-auc    pr-auc          n
#      class_0      0.895     0.862     0.878     0.946     0.950  |    1017
#      class_1      0.872     0.903     0.887     0.946     0.950  |    1057
#
# weighted avg      0.883     0.883     0.883     0.946     0.950  |
```

Next, we can visualize the kernels from the first convolutional layer. Therefore, we first compute the maximum activations for each kernel/sequence pair and then visualize kernels as explained in Chapter 3 and Chapter 4 of the thesis. The visualize_all_kernels() function will place all output files into a provided folder (i.e. sequence motif image files, violin plots, local enrichment plots as explained in Chapter 4) and it additionally creates a summary.html file that presents an overview of all kernel visualizations ordered by importance score (30 kernels in our case, as this is the default hyperparameter; see Figure A.1 for a web browser screenshot of summary.html).

```
activations = model.get_max_activations(data, group="test")
logos = model.visualize_all_kernels(activations, data, folder="out/")
```

The visualize_all_kernels() function also returns a list of pysster *Motif* objects that can be utilized to access position frequency matrices or that can be saved in MEME format for further use. Since we trained models on two strings, the function returns a list of tuples where the first tuple member represents the sequence motif and the second member the corresponding structure string.

```
utils.save_as_meme(logos=[logo[0] for logo in logos],
                   file_path="out/motifs_seq.meme")
utils.save_as_meme(logos=[logo[1] for logo in logos],
                   file_path="out/motifs_struct.meme")
```

Finally, we can use the previously computed maximum activations for kernel/sequence pairs to perform a hierarchical clustering of both kernels and sequences (see Chapter 4). For this model, results are "boring", because only the PUM2 motif was learned (see Figure A.2).

```
# the classes parameter can be used to restrict the clustering
# to certain classes; here we cluster both classes at the same time

model.plot_clustering(activations, "out/clustering.png", classes=[0, 1])
```

This concludes the common workflow example and both *Data* and trained *Model* objects can now be saved (pickled) for future use (utils.load_data() and utils.load_model() can be used for loading).

```
utils.save_data(data, "out/data.pkl")
utils.save_model(model, "out/model.pkl")
```

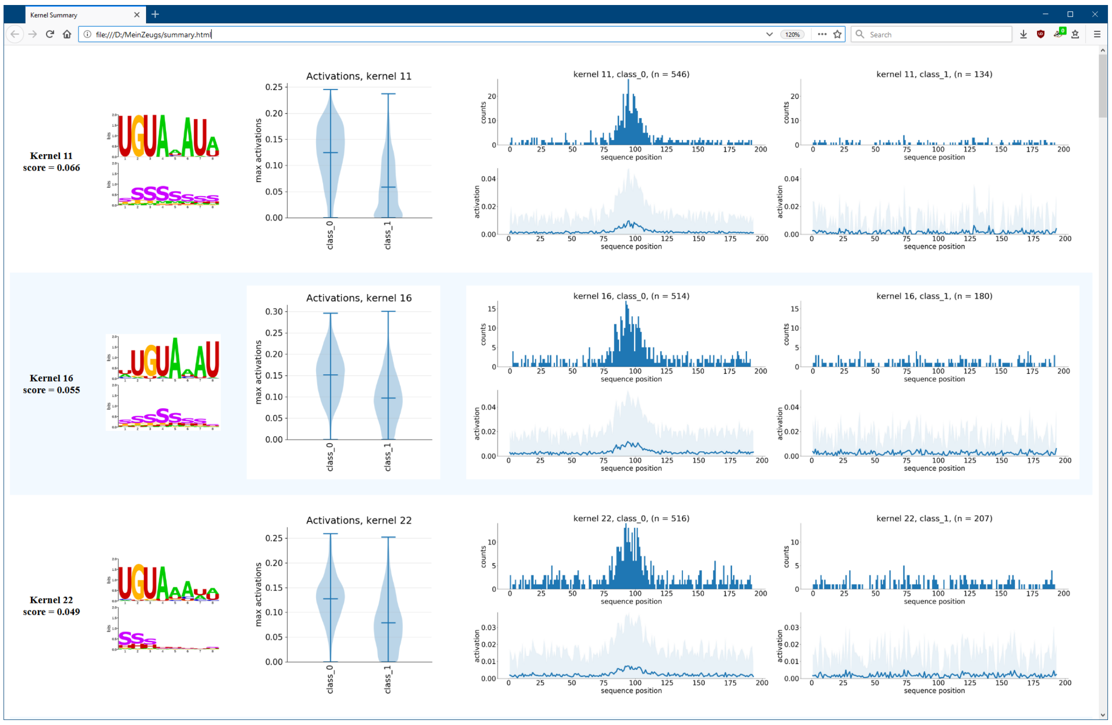


Figure A.1: **Screenshot Of summary.html.** All first-layer kernels are sorted by importance score and visualizations are presented in descending order (a total of 30 kernels for this model).

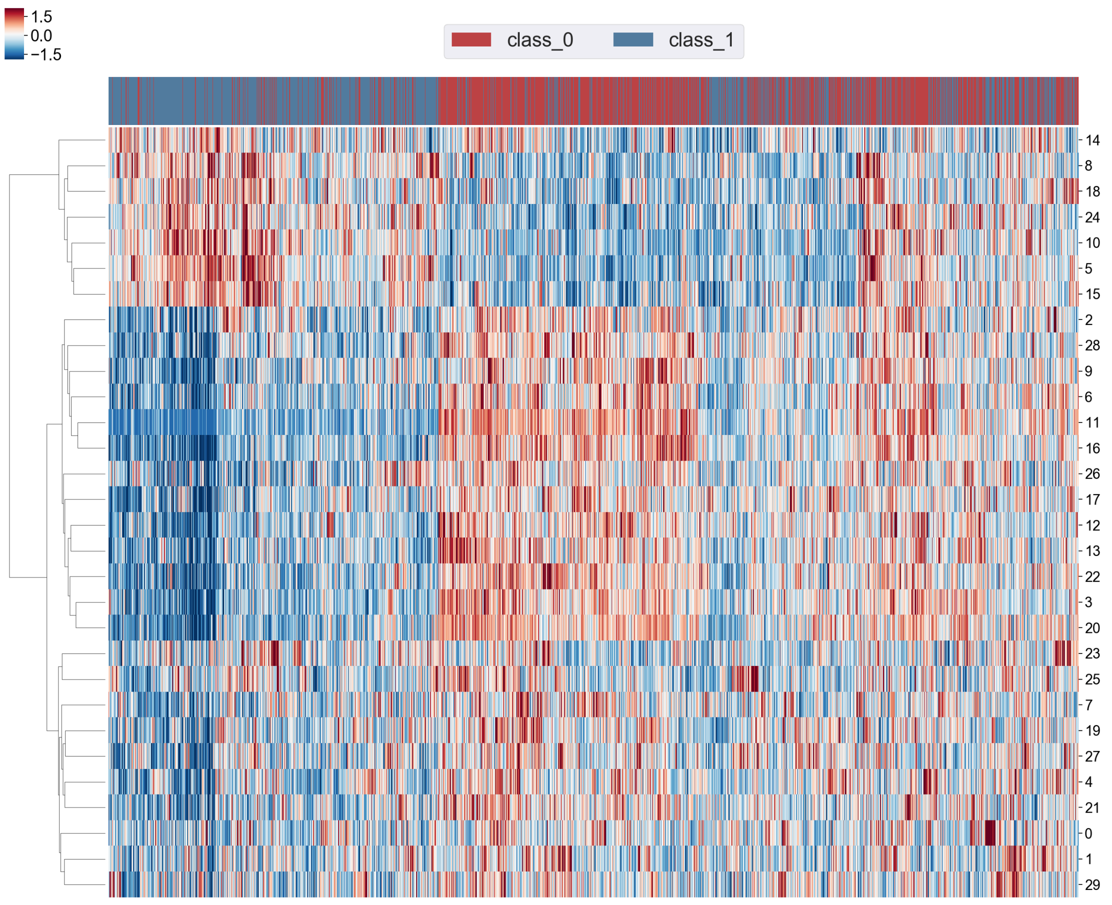


Figure A.2: **Maximum Activation Clustering Of Kernels And Sequences.** Many model kernels learned variations of the PUM2 motif (kernels 2 to 20 in the cluster in the center).

A.4 Further DNA Shape Patterns

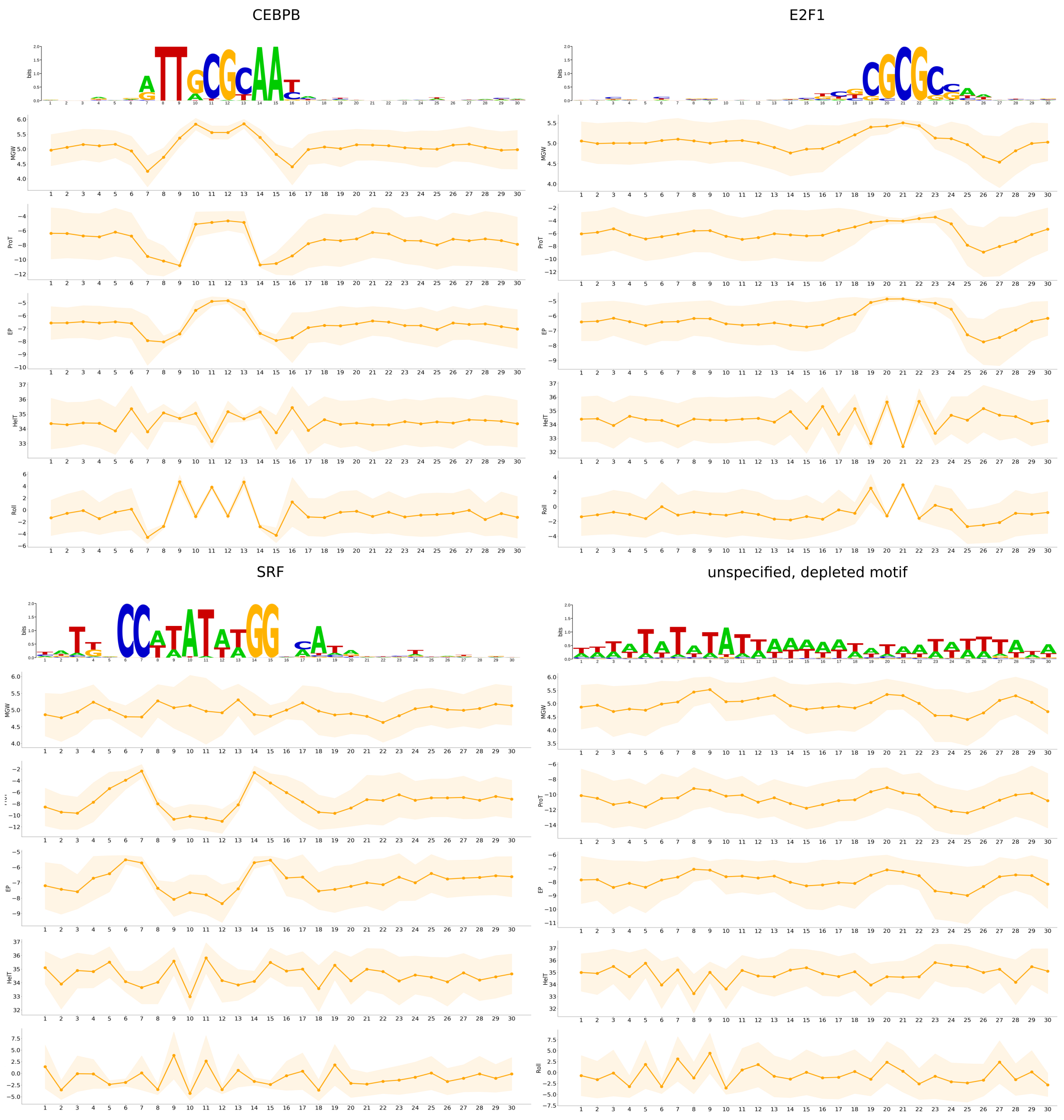


Figure A.3: **DNA Shape Visualizations.** Sequence motif and DNA shape pattern visualizations of kernels that learned to recognize CEBPB, E2F1 and SRF binding sites. Further shape patterns for the AT-rich, depleted motif mentioned in Section 4.3.1 are shown as well.

A.5 RBP Predictions: Data For All 100 Proteins

RBP	CELL LINE	ENCODE ACCESSION	RBP	CELL LINE	ENCODE ACCESSION
AKAP8L	K562	ENCSR206RXT	MTPAP	K562	ENCSR200DKE
BCCIP	HepG2	ENCSR485QCG	NCBP2	HepG2	ENCSR018RVZ
BUD13	HepG2	ENCSR830BSQ	NKRF	HepG2	ENCSR277DEO
CDC40	HepG2	ENCSR815VVI	NONO	K562	ENCSR861PAR
CPSF6	K562	ENCSR532VUB	NPM1	K562	ENCSR867DSZ
CSTF2	HepG2	ENCSR384MWO	PCBP2	HepG2	ENCSR339FUY
CSTF2T	HepG2	ENCSR919HSE	POLR2G	HepG2	ENCSR820WHR
DDX24	K562	ENCSR999WKT	PPIG	HepG2	ENCSR097NEE
DDX3X	HepG2	ENCSR648LAH	PPIL4	K562	ENCSR197INS
DDX42	K562	ENCSR576SHT	PRPF8	HepG2	ENCSR121NVA
DDX55	HepG2	ENCSR845VGB	PTBP1	HepG2	ENCSR384KAN
DDX59	HepG2	ENCSR214BZA	PUM2	K562	ENCSR661ICQ
DDX6	HepG2	ENCSR141OIM	PUS1	K562	ENCSR291XPT
DGCR8	HepG2	ENCSR061SZV	QKI	HepG2	ENCSR570WLM
DHX30	HepG2	ENCSR565DGW	RBFOX2	HepG2	ENCSR987FTF
DKC1	HepG2	ENCSR301TFY	RBM15	HepG2	ENCSR754NDA
DROSHA	K562	ENCSR653HQC	RBM22	HepG2	ENCSR456JJQ
EFTUD2	HepG2	ENCSR527DXF	RBM5	HepG2	ENCSR489ABS
EWSR1	K562	ENCSR887LPK	SAFB2	K562	ENCSR943MHU
EXOSC5	K562	ENCSR013CTQ	SERBP1	K562	ENCSR121GQH
FAM120A	HepG2	ENCSR987NYS	SF3A3	HepG2	ENCSR331MIC
FKBP4	HepG2	ENCSR018ZUE	SF3B1	K562	ENCSR133QEA
FTO	K562	ENCSR989SMC	SF3B4	HepG2	ENCSR279UJF
FUBP3	HepG2	ENCSR486YGP	SFPQ	HepG2	ENCSR965DLL
FXR2	K562	ENCSR224QWC	SLBP	K562	ENCSR483NOP
GEMIN5	K562	ENCSR238CLX	SLTM	HepG2	ENCSR351PVI
GNL3	K562	ENCSR301UQM	SMNDC1	HepG2	ENCSR373ODC
GPKOW	K562	ENCSR647CLF	SRSF1	HepG2	ENCSR989VIY
GRSF1	HepG2	ENCSR668MJX	SRSF7	HepG2	ENCSR513NDD
GRWD1	HepG2	ENCSR893NWB	SRSF9	HepG2	ENCSR773KRC
GTF2F1	HepG2	ENCSR265ZIS	SUB1	HepG2	ENCSR406OOZ
HLTF	K562	ENCSR589YHM	SUGP2	HepG2	ENCSR506UPY
HNRNPA1	HepG2	ENCSR769UEW	SUPV3L1	HepG2	ENCSR580MFX
HNRNPC	HepG2	ENCSR550DVK	TAF15	HepG2	ENCSR841EQA
HNRNPK	HepG2	ENCSR828ZID	TARDBP	K562	ENCSR584TCR
HNRNPL	HepG2	ENCSR724RDN	TBRG4	HepG2	ENCSR916SRV
HNRNPM	HepG2	ENCSR267UCX	TIA1	HepG2	ENCSR623VEQ
HNRNPU	HepG2	ENCSR240MVJ	TRA2A	HepG2	ENCSR314UMJ
HNRNPUL1	HepG2	ENCSR755TJC	TROVE2	HepG2	ENCSR993FMY
IGF2BP1	HepG2	ENCSR744GEU	U2AF1	HepG2	ENCSR328LLU
IGF2BP2	K562	ENCSR062NNB	U2AF2	HepG2	ENCSR202BFN
IGF2BP3	HepG2	ENCSR993OLA	UCHL5	HepG2	ENCSR490IEE
ILF3	HepG2	ENCSR786TSC	UPF1	K562	ENCSR456ASB
KHDRBS1	K562	ENCSR628IDK	XPO5	HepG2	ENCSR921SXC
KHSRP	K562	ENCSR438GZQ	XRCC6	HepG2	ENCSR571ROL
LARP4	HepG2	ENCSR805SRN	XRN2	HepG2	ENCSR655NZA
LARP7	HepG2	ENCSR961OKA	YBX3	K562	ENCSR529FKI
LIN28B	HepG2	ENCSR861GYE	YWHAG	K562	ENCSR867ZVK
LSM11	HepG2	ENCSR135VMS	ZNF622	K562	ENCSR657TZZ
METAP2	K562	ENCSR303OQD	ZRANB2	K562	ENCSR663NRA

Table A.2: **Source Of eCLIP Data.** Bed files for the biological replicates containing called peaks were downloaded on September 6, 2018.

RBP	T	G	OTHER MOTIFS	RBP	T	G	OTHER MOTIFS
AKAP8L	X	X		MTPAP	X		GC-rich
BCCIP		X		NCBP2	X	X	GC-rich
BUD13	X		5′ ss , 3′ ss	NKRF	X	X	GC-rich
CDC40	X	X	CG-rich	NONO	X	X	GC-rich
CPSF6	X	X	GA rich	NPM1	X	X	
CSTF2	X	X	poly(A), ACTAA	PCBP2		X	CT-rich
CSTF2T		X	G- and GC-rich	POLR2G	X	X	ALU motifs
DDX24	X			PPIG			5′ ss, 3′ ss
DDX3X			GC-rich	PPIL4	X	X	
DDX42	X	X		PRPF8		X	5′ ss
DDX55	X		GT-rich	PTBP1	X		CT-rich
DDX59	X		C-rich	PUM2	X		TGTANATA
DDX6	X	X		PUS1	X		
DGCR8	X	X	3′ ss	QKI	X		ACTAA
DHX30	X		G-rich	RBFOX2		X	TGCATG
DKC1	X		GT-rich	RBM15	X		GC-rich
DROSHA	X	X		RBM22	X		5′ ss, 3′ ss, GC-rich
EFTUD2	X	X	5′ ss	RBM5	X	X	
EWSR1	X	X	GC- and GT-rich	SAFB2	X	X	GC-rich
EXOSC5	X	X	GA- and GT-rich	SERBP1			GC-rich
FAM120A		X		SF3A3			3′ ss
FKBP4		X		SF3B1	X	X	3′ ss
FTO	X	X	GC-rich	SF3B4		X	3′ ss
FUBP3	X		TGT	SFPQ		X	CCG
FXR2		X	GC-rich, 5′ ss	SLBP	X		3′ ss
GEMIN5	X		G-rich	SLTM	X	X	GC-rich
GNL3	X			SMNDC1		X	
GPKOW	X	X		SRSF1		X	5′ ss
GRSF1	X	X	GC-rich	SRSF7		X	GT-rich
GRWD1			GC-rich, 5′ ss, 3′ ss	SRSF9		X	GGA
GTF2F1	X	X		SUB1	X		GT-rich
HLTF		X	GA-rich, 5′ ss	SUGP2	X	X	GA-rich
HNRNPA1			AGGGAG, GC-rich	SUPV3L1	X	X	5′ ss
HNRNPC	X	X	ALU motifs	TAF15		X	GA-rich, 5′ ss
HNRNPK			CT-rich, CCC	TARDBP			GT-rich, TGAATG
HNRNPL	X		CA-rich	TBRG4			
HNRNPM			G-rich	TIA1	X	X	5′ ss
HNRNPU	X	X	AGGGAG	TRA2A			GAAGAA
HNRNPUL1	X		G-rich	TROVE2	X	X	
IGF2BP1		X		U2AF1	X	X	3′ ss
IGF2BP2			C-rich	U2AF2	X		3′ ss
IGF2BP3		X	C-rich	UCHL5			3′ ss
ILF3	X		ALU motifs	UPF1	X		G- and C-rich
KHDRBS1	X		poly(A)	XPO5			GC-rich
KHSRP	X		G-rich	XRCC6	X		ALU motifs
LARP4	X			XRN2		X	GC-rich
LARP7			TGA	YBX3			TGTCATC, C-rich
LIN28B			GA-rich	YWHAG	X	X	
LSM11	X			ZNF622			GA-rich, 5′ ss, 3′ ss
METAP2	X		GC-rich	ZRANB2	X	X	AGGTA (5′ ss?)

Table A.3: **Motif Summary For All 100 RBPs.** Crosses in the "T" and "G" columns indicate whether kernels learning T-rich and G-rich eCLIP bias motifs were found. Bias motifs make the detection of similar genuine motifs hard, therefore G-rich motifs in the "Other Motifs" column indicate that G-rich motifs with a positional enrichment different from the one shown in Figure 5.3 were found. "ss" stands for splice site.

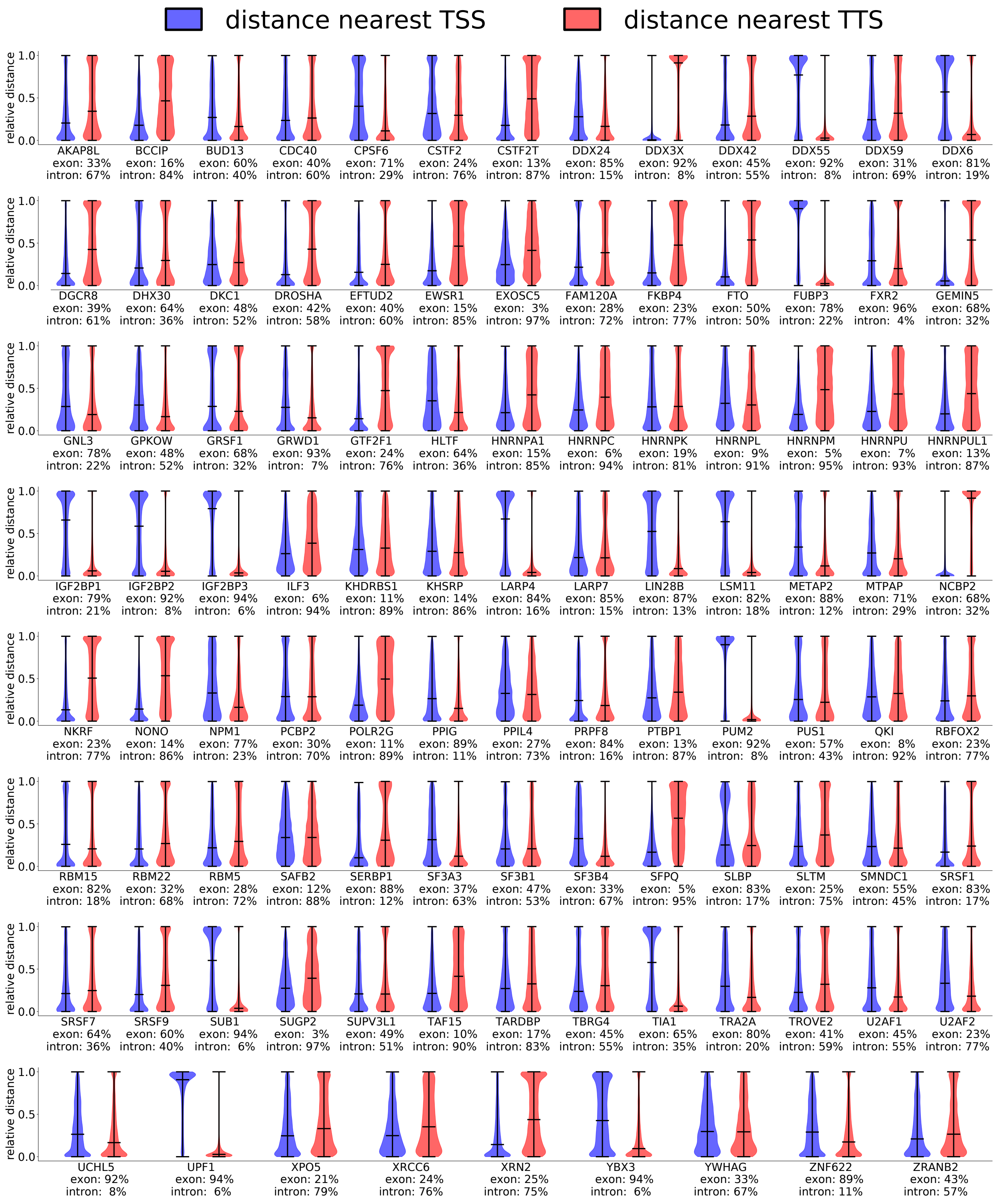


Figure A.4: **RBP Location Information Overview.** Summary statistics based on peaks found in both eCLIP replicates that have a log-fold enrichment over the input of bigger than two. Distances of binding sites to the nearest TSS/TTS are normalized to the transcript length, i.e. a distance of 0 indicates that the binding site is equal to the TSS/TTS.

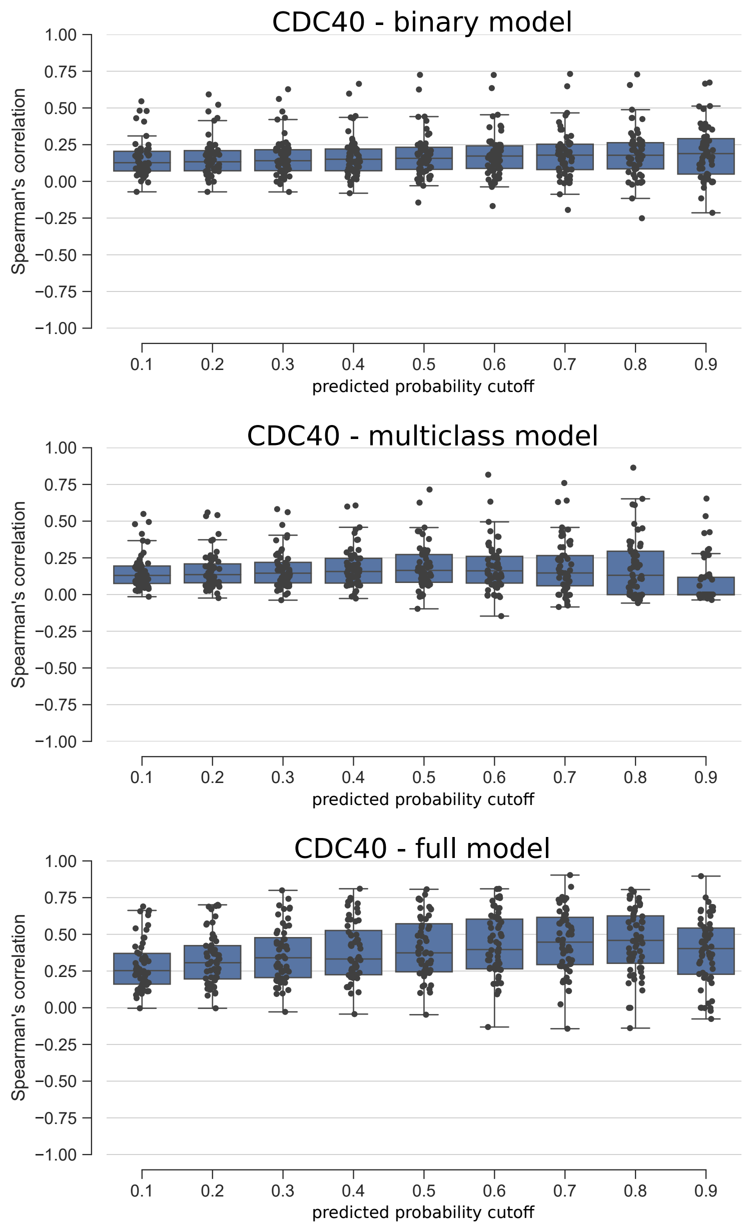


Figure A.5: **Effect Of Probability Cutoffs On Spearman's Correlation.** Each black dot shows the Spearman's correlation for a protein/transcript pair. To denoise the predicted probabilities, we apply a probability cutoff before computing their Spearman's correlation with the eCLIP signal, that is, all values below the cutoff are set to zero. For binary models, the impact of the cutoff is small (possibly due to binary models producing many predictions even for high cutoffs), while it is bigger for multiclass and full models (multiclass + imbalanced data + additional input). 0.5 (binary) and 0.66 (three-class multiclass) are reasonable cutoffs for many proteins, but cutoffs can also be tuned for individual proteins at the risk of overfitting.

RBP	AUPRE	RBP	AUPRE
AKAP8L	0.552 / 0.944 / 0.851	MTPAP	0.447 / 0.978 / 0.868
BCCIP	0.676 / 0.920 / 0.862	NCBP2	0.743 / 0.975 / 0.925
BUD13	0.603 / 0.991 / 0.920	NKRF	0.540 / 0.970 / 0.881
CDC40	0.497 / 0.970 / 0.866	NONO	0.659 / 0.954 / 0.883
CPSF6	0.583 / 0.965 / 0.892	NPM1	0.513 / 0.946 / 0.851
CSTF2	0.807 / 0.936 / 0.915	PCBP2	0.778 / 0.953 / 0.903
CSTF2T	0.629 / 0.980 / 0.929	POLR2G	0.604 / 0.910 / 0.857
DDX24	0.491 / 0.989 / 0.914	PPIG	0.586 / 0.992 / 0.929
DDX3X	0.787 / 0.994 / 0.957	PPIL4	0.537 / 0.892 / 0.857
DDX42	0.589 / 0.937 / 0.854	PRPF8	0.797 / 0.991 / 0.956
DDX55	0.452 / 0.995 / 0.918	PTBP1	0.850 / 0.911 / 0.882
DDX59	0.645 / 0.949 / 0.876	PUM2	0.819 / 0.991 / 0.969
DDX6	0.518 / 0.989 / 0.890	PUS1	0.417 / 0.906 / 0.796
DGCR8	0.574 / 0.964 / 0.845	QKI	0.913 / 0.936 / 0.937
DHX30	0.526 / 0.952 / 0.849	RBFOX2	0.752 / 0.967 / 0.911
DKC1	0.878 / 0.969 / 0.934	RBM15	0.516 / 0.984 / 0.891
DROSHA	0.587 / 0.954 / 0.868	RBM22	0.717 / 0.967 / 0.906
EFTUD2	0.581 / 0.992 / 0.940	RBM5	0.516 / 0.953 / 0.862
EWSR1	0.577 / 0.980 / 0.933	SAFB2	0.642 / 0.912 / 0.906
EXOSC5	0.847 / 0.918 / 0.905	SERBP1	0.746 / 0.994 / 0.954
FAM120A	0.603 / 0.984 / 0.914	SF3A3	0.795 / 0.987 / 0.955
FKBP4	0.645 / 0.985 / 0.928	SF3B1	0.705 / 0.936 / 0.850
FTO	0.619 / 0.956 / 0.876	SF3B4	0.838 / 0.989 / 0.968
FUBP3	0.834 / 0.986 / 0.968	SFPQ	0.648 / 0.949 / 0.906
FXR2	0.541 / 0.991 / 0.955	SLBP	0.841 / 0.934 / 0.885
GEMIN5	0.609 / 0.979 / 0.903	SLTM	0.566 / 0.942 / 0.856
GNL3	0.569 / 0.969 / 0.857	SMNDC1	0.647 / 0.972 / 0.886
GPKOW	0.756 / 0.973 / 0.912	SRSF1	0.614 / 0.987 / 0.927
GRSF1	0.691 / 0.985 / 0.929	SRSF7	0.643 / 0.972 / 0.901
GRWD1	0.502 / 0.994 / 0.933	SRSF9	0.599 / 0.972 / 0.908
GTF2F1	0.604 / 0.962 / 0.893	SUB1	0.530 / 0.993 / 0.939
HLTF	0.590 / 0.963 / 0.905	SUGP2	0.756 / 0.928 / 0.913
HNRNPA1	0.733 / 0.938 / 0.925	SUPV3L1	0.591 / 0.984 / 0.891
HNRNPC	0.855 / 0.940 / 0.932	TAF15	0.652 / 0.929 / 0.899
HNRNPK	0.842 / 0.960 / 0.933	TARDBP	0.907 / 0.936 / 0.919
HNRNPL	0.874 / 0.909 / 0.927	TBRG4	0.708 / 0.941 / 0.867
HNRNPM	0.828 / 0.950 / 0.928	TIA1	0.738 / 0.983 / 0.953
HNRNPU	0.540 / 0.900 / 0.864	TRA2A	0.722 / 0.983 / 0.942
HNRNPUL1	0.479 / 0.888 / 0.833	TROVE2	0.501 / 0.938 / 0.830
IGF2BP1	0.717 / 0.988 / 0.941	U2AF1	0.580 / 0.966 / 0.877
IGF2BP2	0.583 / 0.993 / 0.940	U2AF2	0.809 / 0.955 / 0.928
IGF2BP3	0.574 / 0.997 / 0.955	UCHL5	0.435 / 0.994 / 0.920
ILF3	0.786 / 0.935 / 0.927	UPF1	0.689 / 0.995 / 0.968
KHDRBS1	0.822 / 0.900 / 0.921	XPO5	0.703 / 0.932 / 0.880
KHSRP	0.901 / 0.941 / 0.941	XRCC6	0.581 / 0.896 / 0.861
LARP4	0.518 / 0.987 / 0.908	XRN2	0.532 / 0.967 / 0.888
LARP7	0.676 / 0.982 / 0.913	YBX3	0.624 / 0.994 / 0.952
LIN28B	0.605 / 0.993 / 0.941	YWHAG	0.571 / 0.915 / 0.833
LSM11	0.535 / 0.988 / 0.903	ZNF622	0.500 / 0.989 / 0.918
METAP2	0.409 / 0.989 / 0.878	ZRANB2	0.657 / 0.942 / 0.855

Table A.4: **Class-wise auPRE For All 100 RBPs.** Performance results on held-out test data for the full models (multiclass + imbalanced data + additional input). Numbers indicate the auPRE of the individual classes computed in a one-versus-rest fashion in the following order: positive class / lincRNA class / RBP class.

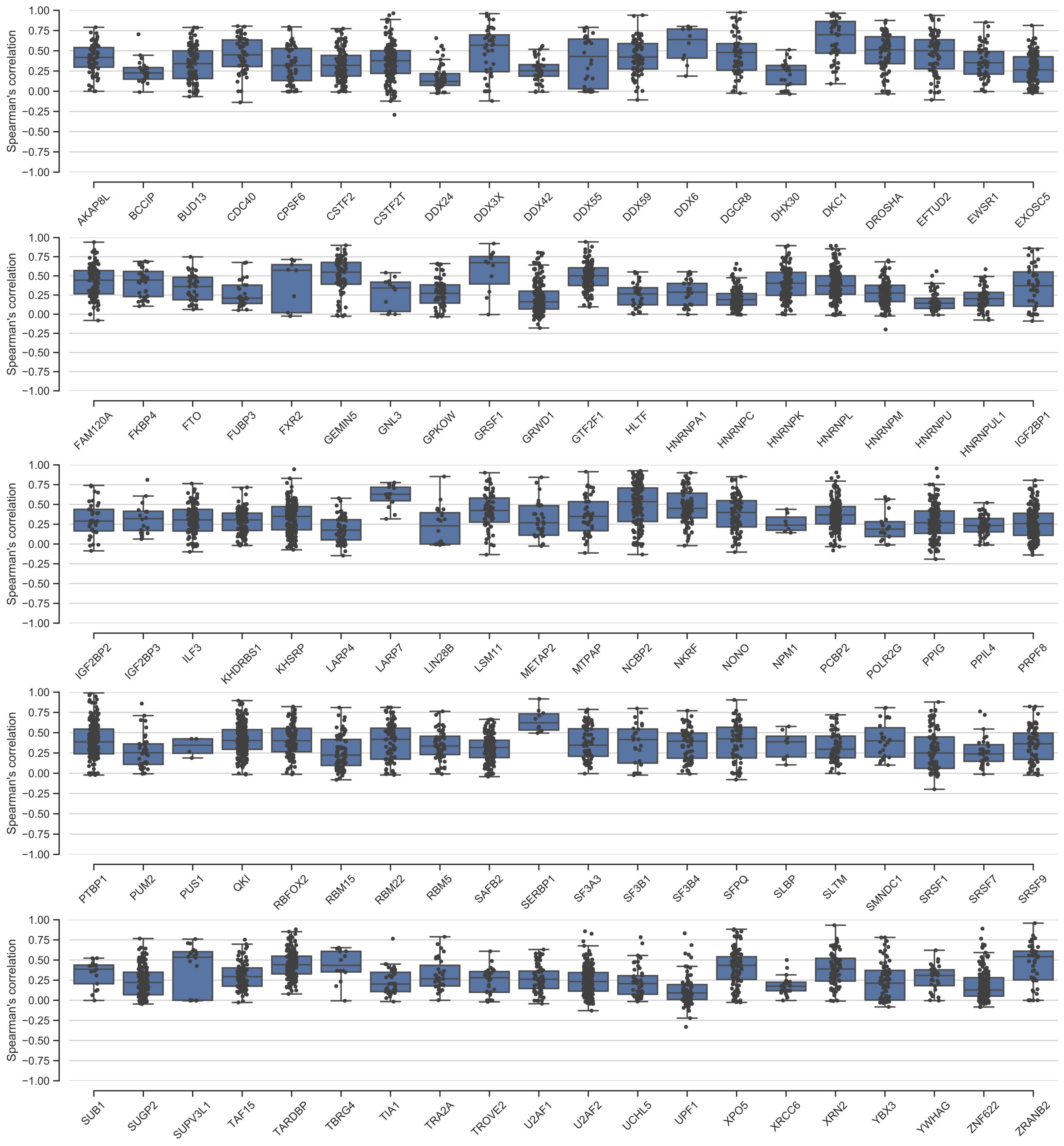


Figure A.6: **LincRNA Scanning Results For Full Models.** Each black dot shows the Spearman's correlation for a protein/transcript pair using the full CNN models (multi-class + imbalanced data + additional input).

Bibliography

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification." In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.

[2] Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. "Methods for interpreting and understanding deep neural networks." In: *Digital Signal Processing* 73 (2018), pp. 1–15.

[3] John R Zech, Marcus A Badgeley, Manway Liu, Anthony B Costa, Joseph J Titano, and Eric Karl Oermann. "Variable generalization performance of a deep learning model to detect pneumonia in chest radiographs: A cross-sectional study." In: *PLoS medicine* 15.11 (2018), e1002683.

[4] Jeffrey Dastin. *Amazon scraps secret AI recruiting tool that showed bias against women*. `https://www.reuters.com/article/us-amazon-com-jobs-automation-insight/amazon-scraps-secret-ai-recruiting-tool-that-showed-bias-against-women-idUSKCN1MK08G`. Accessed: 9-7-2019. 2018.

[5] Kasper D Hansen, Rafael A Irizarry, and Zhijin Wu. "Removing technical variability in RNA-seq data using conditional quantile normalization." In: *Biostatistics* 13.2 (2012), pp. 204–216.

[6] Leonid Teytelman, Deborah M Thurtle, Jasper Rine, and Alexander van Oudenaarden. "Highly expressed loci are vulnerable to misleading ChIP localization of multiple unrelated proteins." In: *Proceedings of the National Academy of Sciences* 110.46 (2013), pp. 18602–18607.

[7] Raga Krishnakumar, Anupama Sinha, Sara W Bird, Harikrishnan Jayamohan, Harrison S Edwards, Joseph S Schoeniger, Kamlesh D Patel, Steven S Branda, and Michael S Bartsch. "Systematic and stochastic influences on the performance of the MinION nanopore sequencer across a range of nucleotide bias." In: *Scientific reports* 8.1 (2018), p. 3159.

[8] Timothy W Nilsen and Brenton R Graveley. "Expansion of the eukaryotic proteome by alternative splicing." In: *Nature* 463.7280 (2010), p. 457.

[9] Jacob O'Brien, Heyam Hayder, Yara Zayed, and Chun Peng. "Overview of microRNA biogenesis, mechanisms of actions, and circulation." In: *Frontiers in endocrinology* 9 (2018), p. 402.

[10] Run-Wen Yao, Yang Wang, and Ling-Ling Chen. "Cellular functions of long noncoding RNAs." In: *Nature cell biology* 21.5 (2019), pp. 542–551.

[11] Alan B Sachs, Peter Sarnow, and Matthias W Hentze. "Starting at the beginning, middle, and end: translation initiation in eukaryotes." In: *Cell* 89.6 (1997), pp. 831–838.

[12] Juan M Vaquerizas, Sarah K Kummerfeld, Sarah A Teichmann, and Nicholas M Luscombe. "A census of human transcription factors: function, expression and evolution." In: *Nature Reviews Genetics* 10.4 (2009), p. 252.

[13] Stephen T Smale and James T Kadonaga. "The RNA polymerase II core promoter." In: *Annual review of biochemistry* 72.1 (2003), pp. 449–479.

[14] Murat Tuğrul, Tiago Paixao, Nicholas H Barton, and Gašper Tkačik. "Dynamics of transcription factor binding site evolution." In: *PLoS genetics* 11.11 (2015), e1005639.

[15] Bing Li, Michael Carey, and Jerry L Workman. "The role of chromatin during transcription." In: *Cell* 128.4 (2007), pp. 707–719.

[16] Moyra Lawrence, Sylvain Daujat, and Robert Schneider. "Lateral thinking: how histone modifications regulate gene expression." In: *Trends in Genetics* 32.1 (2016), pp. 42–56.

[17] Jinsen Li, Jared M Sagendorf, Tsu-Pei Chiu, Marco Pasi, Alberto Perez, and Remo Rohs. "Expanding the repertoire of DNA shape features for genome-scale studies of transcription factor binding." In: *Nucleic acids research* 45.22 (2017), pp. 12877–12887.

[18] Remo Rohs, Sean M West, Alona Sosinsky, Peng Liu, Richard S Mann, and Barry Honig. "The role of DNA shape in protein–DNA recognition." In: *Nature* 461.7268 (2009), p. 1248.

[19] Richard Lavery and Bernard Pullman. "The electrostatic field of DNA: the role of the nucleic acid conformation." In: *Nucleic acids research* 10.14 (1982), pp. 4383–4395.

[20] Tsu-Pei Chiu, Federico Comoglio, Tianyin Zhou, Lin Yang, Renato Paro, and Remo Rohs. "DNAshapeR: an R/Bioconductor package for DNA shape prediction and feature encoding." In: *Bioinformatics* 32.8 (2015), pp. 1211–1213.

[21] David S Johnson, Ali Mortazavi, Richard M Myers, and Barbara Wold. "Genome-wide mapping of in vivo protein-DNA interactions." In: *Science* 316.5830 (2007), pp. 1497–1502.

[22] Stephen G Landt, Georgi K Marinov, Anshul Kundaje, Pouya Kheradpour, Florencia Pauli, Serafim Batzoglou, Bradley E Bernstein, Peter Bickel, James B Brown, Philip Cayting, et al. "ChIP-seq guidelines and practices of the ENCODE and modENCODE consortia." In: *Genome research* 22.9 (2012), pp. 1813–1831.

[23] Tina Glisovic, Jennifer L Bachorik, Jeongsik Yong, and Gideon Dreyfuss. "RNA-binding proteins and post-transcriptional gene regulation." In: *FEBS letters* 582.14 (2008), pp. 1977–1986.

[24] Matthias W Hentze, Alfredo Castello, Thomas Schwarzl, and Thomas Preiss. "A brave new world of RNA-binding proteins." In: *Nature Reviews Molecular Cell Biology* 19.5 (2018), p. 327.

[25] Bradley M Lunde, Claire Moore, and Gabriele Varani. "RNA-binding proteins: modular design for efficient function." In: *Nature reviews Molecular cell biology* 8.6 (2007), p. 479.

[26] Daniel Dominguez, Peter Freese, Maria S Alexis, Amanda Su, Myles Hochman, Tsultrim Palden, Cassandra Bazile, Nicole J Lambert, Eric L Van Nostrand, Gabriel A Pratt, et al. "Sequence, structure, and context preferences of human RNA binding proteins." In: *Molecular cell* 70.5 (2018), pp. 854–867.

[27] James R Williamson. "Induced fit in RNA–protein recognition." In: *Nature Structural & Molecular Biology* 7.10 (2000), p. 834.

[28] Silvi Rouskin, Meghan Zubradt, Stefan Washietl, Manolis Kellis, and Jonathan S Weissman. "Genome-wide probing of RNA structure reveals active unfolding of mRNA structures in vivo." In: *Nature* 505.7485 (2014), p. 701.

[29] Meghan Zubradt, Paromita Gupta, Sitara Persad, Alan M Lambowitz, Jonathan S Weissman, and Silvi Rouskin. "DMS-MaPseq for genome-wide or targeted RNA structure probing in vivo." In: *Nature methods* 14.1 (2017), p. 75.

[30] Ivo L Hofacker and Ronny Lorenz. "Predicting RNA structure: advances and limitations." In: *RNA Folding*. Springer, 2014, pp. 1–19.

[31] Jernej Ule, Kirk B Jensen, Matteo Ruggiu, Aldo Mele, Aljaž Ule, and Robert B Darnell. "CLIP identifies Nova-regulated RNA networks in the brain." In: *Science* 302.5648 (2003), pp. 1212–1215.

[32] Eric L Van Nostrand, Gabriel A Pratt, Alexander A Shishkin, Chelsea Gelboin-Burkhart, Mark Y Fang, Balaji Sundararaman, Steven M Blue, Thai B Nguyen, Christine Surka, Keri Elkins, et al. "Robust transcriptome-wide discovery of RNA-binding protein binding sites with enhanced CLIP (eCLIP)." In: *Nature methods* 13.6 (2016), p. 508.

[33] Yong Zhang, Tao Liu, Clifford A Meyer, Jérôme Eeckhoute, David S Johnson, Bradley E Bernstein, Chad Nusbaum, Richard M Myers, Myles Brown, Wei Li, et al. "Model-based analysis of ChIP-Seq (MACS)." In: *Genome biology* 9.9 (2008), R137.

[34] Sabrina Krakau, Hugues Richard, and Annalisa Marsico. "PureCLIP: capturing target-specific protein–RNA interaction footprints from single-nucleotide CLIP-seq data." In: *Genome biology* 18.1 (2017), p. 240.

[35] ENCODE Project Consortium et al. "An integrated encyclopedia of DNA elements in the human genome." In: *Nature* 489.7414 (2012), p. 57.

[36] Eric L Van Nostrand, Peter Freese, Gabriel A Pratt, Xiaofeng Wang, Xintao Wei, Steven M Blue, Daniel Dominguez, Neal AL Cody, Sara Olson, Balaji Sundararaman, et al. "A large-scale binding and functional map of human RNA binding proteins." In: *bioRxiv* (2018), p. 179648.

[37] Kazuko Nishikura. "A-to-I editing of coding and non-coding RNAs by ADARs." In: *Nature reviews Molecular cell biology* 17.2 (2016), p. 83.

[38] Ernesto Picardi, Anna Maria D'Erchia, Claudio Lo Giudice, and Graziano Pesole. "REDIportal: a comprehensive database of A-to-I RNA editing events in humans." In: *Nucleic acids research* 45.D1 (2016), pp. D750–D757.

[39] Ernesto Picardi, Caterina Manzari, Francesca Mastropasqua, Italia Aiello, Anna Maria D'Erchia, and Graziano Pesole. "Profiling RNA editing in human tissues: towards the inosinome Atlas." In: *Scientific reports* 5 (2015), p. 14941.

[40] Erez Y Levanon and Eli Eisenberg. "Does RNA editing compensate for Alu invasion of the primate genome?" In: *Bioessays* 37.2 (2015), pp. 175–181.

[41] Shengyong Yang, Peng Deng, Zhaowei Zhu, Jianzhong Zhu, Guoliang Wang, Liyong Zhang, Alex F Chen, Tony Wang, Saumendra N Sarkar, Timothy R Billiar, et al. "Adenosine deaminase acting on RNA 1 limits RIG-I RNA detection and suppresses IFN production responding to viral and endogenous RNAs." In: *The Journal of Immunology* 193.7 (2014), pp. 3436–3445.

[42] Thomas D Schneider and R Michael Stephens. "Sequence logos: a new way to display consensus sequences." In: *Nucleic acids research* 18.20 (1990), pp. 6097–6100.

[43] Patrik D'haeseleer. "What are DNA sequence motifs?" In: *Nature biotechnology* 24.4 (2006), p. 423.

[44] Modan K Das and Ho-Kwok Dai. "A survey of DNA motif finding algorithms." In: *BMC bioinformatics*. Vol. 8. 7. BioMed Central. 2007, S21.

[45] Timothy L Bailey, Mikael Boden, Fabian A Buske, Martin Frith, Charles E Grant, Luca Clementi, Jingyuan Ren, Wilfred W Li, and William S Noble. "MEME SUITE: tools for motif discovery and searching." In: *Nucleic acids research* 37 (2009), W202–W208.

[46] Nga Thi Thuy Nguyen, Bruno Contreras-Moreira, Jaime A Castro-Mondragon, Walter Santana-Garcia, Raul Ossio, Carla Daniela Robles-Espinoza, Mathieu Bahin, Samuel Collombet, Pierre Vincens, Denis Thieffry, et al. "RSAT 2018: regulatory sequence analysis tools 20th anniversary." In: *Nucleic acids research* 46.W1 (2018), W209–W214.

[47] Yue Fan, Mark Kon, and Charles DeLisi. "Transcription Factor-DNA Binding Via Machine Learning Ensembles." In: *arXiv preprint arXiv:1805.03771* (2018).

[48] Gunnar Rätsch, Sören Sonnenburg, and Christin Schäfer. "Learning interpretable SVMs for biological sequence classification." In: *BMC bioinformatics*. Vol. 7. 1. BioMed Central. 2006, S9.

[49] Mahmoud Ghandi, Dongwon Lee, Morteza Mohammad-Noori, and Michael A Beer. "Enhanced regulatory sequence prediction using gapped k-mer features." In: *PLoS computational biology* 10.7 (2014), e1003711.

[50] Babak Alipanahi, Andrew Delong, Matthew T Weirauch, and Brendan J Frey. "Predicting the sequence specificities of DNA-and RNA-binding proteins by deep learning." In: *Nature biotechnology* 33.8 (2015), p. 831.

[51] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity." In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[52] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep sparse rectifier neural networks." In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. 2011, pp. 315–323.

[53] Prajit Ramachandran, Barret Zoph, and Quoc V Le. "Searching for activation functions." In: *arXiv preprint arXiv:1710.05941* (2017).

[54] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[55] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. "Learning representations by back-propagating errors." In: *Cognitive modeling* 5.3 (1988), p. 1.

[56] Dominic Masters and Carlo Luschi. "Revisiting small batch training for deep neural networks." In: *arXiv preprint arXiv:1804.07612* (2018).

[57] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." In: *nature* 521.7553 (2015), p. 436.

[58] Kunihiko Fukushima. "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position." In: *Biological cybernetics* 36.4 (1980), pp. 193–202.

[59] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. "Gradient-based learning applied to document recognition." In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[60] Waseem Rawat and Zenghui Wang. "Deep convolutional neural networks for image classification: A comprehensive review." In: *Neural computation* 29.9 (2017), pp. 2352–2449.

[61] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting." In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[62] Lutz Prechelt. “Early stopping-but when?” In: *Neural Networks: Tricks of the trade*. Springer, 1998, pp. 55–69.

[63] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. “Feature Visualization.” In: *Distill* (2017). https://distill.pub/2017/feature-visualization. DOI: 10.23915/distill.00007.

[64] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. “Deep inside convolutional networks: Visualising image classification models and saliency maps.” In: *arXiv preprint arXiv:1312.6034* (2013).

[65] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. “Learning important features through propagating activation differences.” In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 3145–3153.

[66] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. “On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation.” In: *PloS one* 10.7 (2015), e0130140.

[67] Jack Lanchantin, Ritambhara Singh, Beilun Wang, and Yanjun Qi. “Deep motif dashboard: Visualizing and understanding genomic sequences using deep neural networks.” In: *PACIFIC SYMPOSIUM ON BIOCOMPUTING 2017*. World Scientific. 2017, pp. 254–265.

[68] Matthew T Weirauch, Atina Cote, Raquel Norel, Matti Annala, Yue Zhao, Todd R Riley, Julio Saez-Rodriguez, Thomas Cokelaer, Anastasia Vedenko, Shaheynoor Talukder, et al. “Evaluation of methods for modeling transcription factor sequence specificity.” In: *Nature biotechnology* 31.2 (2013), p. 126.

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

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

[71] Christof Angermueller, Heather J Lee, Wolf Reik, and Oliver Stegle. “DeepCpG: accurate prediction of single-cell DNA methylation states using deep learning.” In: *Genome biology* 18.1 (2017), p. 67.

[72] Xiaoyong Pan, Peter Rijnbeek, Junchi Yan, and Hong-Bin Shen. “Prediction of RNA-protein sequence and structure binding preferences using deep convolutional and recurrent neural networks.” In: *BMC genomics* 19.1 (2018), p. 511.

[73] Jesse Davis and Mark Goadrich. “The relationship between Precision-Recall and ROC curves.” In: *Proceedings of the 23rd international conference on Machine learning*. ACM. 2006, pp. 233–240.

[74] Takaya Saito and Marc Rehmsmeier. “The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets.” In: *PloS one* 10.3 (2015), e0118432.

[75] Stefan Budach and Annalisa Marsico. “pysster: classification of biological sequences by learning sequence and structure motifs with convolutional neural networks.” In: *Bioinformatics* 34.17 (2018), pp. 3035–3037.

[76] Yoshua Bengio. “Practical recommendations for gradient-based training of deep architectures.” In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 437–478.

[77] François Chollet et al. *Keras*. `https://keras.io`. 2015.

[78] Ronny Lorenz, Stephan H Bernhart, Christian Höner Zu Siederdissen, Hakim Tafer, Christoph Flamm, Peter F Stadler, and Ivo L Hofacker. “ViennaRNA Package 2.0.” In: *Algorithms for molecular biology* 6.1 (2011), p. 26.

[79] Bernhard C Thiel, Irene K Beckmann, Peter Kerpedjiev, and Ivo L Hofacker. “3D based on 2D: Calculating helix angles and stacking patterns using forgi 2.0, an RNA Python library centered on secondary structure elements.” In: *F1000Research* 8 (2019).

[80] Stephan R Starick, Jonas Ibn-Salem, Marcel Jurk, Céline Hernandez, Michael I Love, Ho-Ryun Chung, Martin Vingron, Morgane Thomas-Chollier, and Sebastiaan H Meijsing. “ChIP-exo signal associated with DNA-binding motifs provides insight into the genomic binding of the glucocorticoid receptor and cooperating transcription factors.” In: *Genome research* 25.6 (2015), pp. 825–835.

[81] Tsu-Pei Chiu, Lin Yang, Tianyin Zhou, Bradley J Main, Stephen CJ Parker, Sergey V Nuzhdin, Thomas D Tullius, and Remo Rohs. “GBshape: a genome browser database for DNA shape annotations.” In: *Nucleic acids research* 43.D1 (2014), pp. D103–D109.

[82] David Heller, Ralf Krestel, Uwe Ohler, Martin Vingron, and Annalisa Marsico. “ssHMM: extracting intuitive sequence-structure motifs from high-throughput RNA-binding protein data.” In: *Nucleic acids research* 45.19 (2017), pp. 11004–11018.

[83] Aziz Khan, Oriol Fornes, Arnaud Stigliani, Marius Gheorghe, Jaime A Castro-Mondragon, Robin van der Lee, Adrien Bessy, Jeanne Cheneby, Shubhada R Kulkarni, Ge Tan, et al. “JASPAR 2018: update of the open-access database of transcription factor binding profiles and its web framework.” In: *Nucleic acids research* 46.D1 (2017), pp. D260–D266.

[84] Taiyi Kuo, Michelle J Lew, Oleg Mayba, Charles A Harris, Terence P Speed, and Jen-Chywan Wang. “Genome-wide analysis of glucocorticoid receptor-binding sites in myotubes identifies gene networks modulating insulin signaling.” In: *Proceedings of the National Academy of Sciences* 109.28 (2012), pp. 11160–11165.

[85] Alekos Athanasiadis, Alexander Rich, and Stefan Maas. "Widespread A-to-I RNA editing of Alu-containing mRNAs in the human transcriptome." In: *PLoS biology* 2.12 (2004), e391.

[86] Michael Hadjiargyrou and Nicholas Delihas. "The intertwining of transposable elements and non-coding RNAs." In: *International journal of molecular sciences* 14.7 (2013), pp. 13307–13328.

[87] Lin Yang, Tianyin Zhou, Iris Dror, Anthony Mathelier, Wyeth W Wasserman, Raluca Gordân, and Remo Rohs. "TFBSshape: a motif database for DNA shape features of transcription factor binding sites." In: *Nucleic acids research* 42.D1 (2013), pp. D148–D155.

[88] Fabien Tencé. *Visualizing Deep Neural Networks Classes and Features*. `http://ankivil.com/visualizing-deep-neural-networks-classes-and-features/`. Accessed: 19-11-2017. 2016.

[89] Daniel Maticzka, Sita J Lange, Fabrizio Costa, and Rolf Backofen. "GraphProt: modeling binding preferences of RNA-binding proteins." In: *Genome biology* 15.1 (2014), R17.

[90] Markus Hafner, Markus Landthaler, Lukas Burger, Mohsen Khorshid, Jean Hausser, Philipp Berninger, Andrea Rothballer, Manuel Ascano Jr, Anna-Carina Jungkamp, Mathias Munschauer, et al. "Transcriptome-wide identification of RNA-binding protein and microRNA target sites by PAR-CLIP." In: *Cell* 141.1 (2010), pp. 129–141.

[91] Jessica I Hoell, Erik Larsson, Simon Runge, Jeffrey D Nusbaum, Sujitha Duggimpudi, Thalia A Farazi, Markus Hafner, Arndt Borkhardt, Chris Sander, and Thomas Tuschl. "RNA targets of wild-type and mutant FET family proteins." In: *Nature structural & molecular biology* 18.12 (2011), p. 1428.

[92] Anthony Mathelier, Beibei Xin, Tsu-Pei Chiu, Lin Yang, Remo Rohs, and Wyeth W Wasserman. "DNA shape features improve transcription factor binding site predictions in vivo." In: *Cell systems* 3.3 (2016), pp. 278–286.

[93] Igor Ulitsky and David P Bartel. "lincRNAs: genomics, evolution, and mechanisms." In: *Cell* 154.1 (2013), pp. 26–46.

[94] Julia D Ransohoff, Yuning Wei, and Paul A Khavari. "The functions and unique features of long intergenic non-coding RNA." In: *Nature reviews Molecular cell biology* 19.3 (2018), p. 143.

[95] Adam Frankish, Mark Diekhans, Anne-Maud Ferreira, Rory Johnson, Irwin Jungreis, Jane Loveland, Jonathan M Mudge, Cristina Sisu, James Wright, Joel Armstrong, et al. "GENCODE reference annotation for the human and mouse genomes." In: *Nucleic acids research* 47.D1 (2018), pp. D766–D773.

[96] Michael T Lovci, Dana Ghanem, Henry Marr, Justin Arnold, Sherry Gee, Marilyn Parra, Tiffany Y Liang, Thomas J Stark, Lauren T Gehman, Shawn Hoon, et al. "Rbfox proteins regulate alternative mRNA splicing through evolutionarily conserved RNA bridges." In: *Nature structural & molecular biology* 20.12 (2013), p. 1434.

[97] Christopher R Sibley, Lorea Blazquez, and Jernej Ule. "Lessons from non-canonical splicing." In: *Nature Reviews Genetics* 17.7 (2016), p. 407.

[98] Annia Mesa, Jason A Somarelli, and Rene J Herrera. "Spliceosomal immunophilins." In: *FEBS letters* 582.16 (2008), pp. 2345–2351.

[99] Patrick Champion-Arnaud and Robin Reed. "The prespliceosome components SAP 49 and SAP 145 interact in a complex implicated in tethering U2 snRNP to the branch site." In: *Genes & development* 8.16 (1994), pp. 1974–1983.

[100] Yoshio Takagaki, Clinton C MacDonald, Thomas Shenk, and James L Manley. "The human 64-kDa polyadenylylation factor contains a ribonucleoprotein-type RNA binding domain and unusual auxiliary motifs." In: *Proceedings of the National Academy of Sciences* 89.4 (1992), pp. 1403–1407.

[101] Emmanuel Beaudoing, Susan Freier, Jacqueline R Wyatt, Jean-Michel Claverie, and Daniel Gautheret. "Patterns of variant polyadenylation signal usage in human genes." In: *Genome research* 10.7 (2000), pp. 1001–1010.

[102] Yoichiro Sugimoto, Julian König, Shobbir Hussain, Blaž Zupan, Tomaž Curk, Michaela Frye, and Jernej Ule. "Analysis of CLIP and iCLIP methods for nucleotide-resolution studies of protein-RNA interactions." In: *Genome biology* 13.8 (2012), R67.

[103] Nejc Haberman, Ina Huppertz, Jan Attig, Julian König, Zhen Wang, Christian Hauer, Matthias W Hentze, Andreas E Kulozik, Hervé Le Hir, Tomaž Curk, et al. "Insights into the design and interpretation of iCLIP experiments." In: *Genome biology* 18.1 (2017), p. 7.

[104] Kathleen M Chen, Evan M Cofer, Jian Zhou, and Olga G Troyanskaya. "Selene: a PyTorch-based deep learning library for sequence data." In: *Nature methods* 16.4 (2019), p. 315.

[105] Wolfgang Kopp, Remo Monti, Annalaura Tamburrini, Uwe Ohler, and Altuna Akalin. "Janggu-Deep learning for genomics." In: *bioRxiv* (2019), p. 700450.

[106] Ziga Avsec, Roman Kreuzhuber, Johnny Israeli, Nancy Xu, Jun Cheng, Avanti Shrikumar, Abhimanyu Banerjee, Daniel S Kim, Lara Urban, Anshul Kundaje, et al. "Kipoi: accelerating the community exchange and reuse of predictive models for genomics." In: *BioRxiv* (2018), p. 375345.

[107] Žiga Avsec, Roman Kreuzhuber, Johnny Israeli, Nancy Xu, Jun Cheng, Avanti Shrikumar, Abhimanyu Banerjee, Daniel S Kim, Thorsten Beier, Lara Urban, et al. "The Kipoi repository accelerates community exchange and reuse of predictive models for genomics." In: *Nature biotechnology* (2019), p. 1.

[108] Florian Kopp and Joshua T Mendell. "Functional classification and experimental dissection of long noncoding RNAs." In: *Cell* 172.3 (2018), pp. 393–407.

[109] Roman Schulte-Sasse, Stefan Budach, Denes Hnisz, and Annalisa Marsico. "Graph Convolutional Networks Improve the Prediction of Cancer Driver Genes." In: *International Conference on Artificial Neural Networks*. Springer. 2019, pp. 658–668.

[110] John N Weinstein, Eric A Collisson, Gordon B Mills, Kenna R Mills Shaw, Brad A Ozenberger, Kyle Ellrott, Ilya Shmulevich, Chris Sander, Joshua M Stuart, Cancer Genome Atlas Research Network, et al. "The cancer genome atlas pan-cancer analysis project." In: *Nature genetics* 45.10 (2013), p. 1113.

[111] Thomas N Kipf and Max Welling. "Semi-supervised classification with graph convolutional networks." In: *arXiv preprint arXiv:1609.02907* (2016).

[112] Stefan Budach, Matthias Heinig, and Annalisa Marsico. "Principles of microRNA regulation revealed through modeling microrna expression quantitative trait loci." In: *Genetics* 203.4 (2016), pp. 1629–1640.

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

[114] Liangchen Luo, Yuanhao Xiong, Yan Liu, and Xu Sun. "Adaptive gradient methods with dynamic bound of learning rate." In: *arXiv preprint arXiv:1902.09843* (2019).

List of Figures

List of Tables

Zusammenfassung

Biologische Sequenzen - DNA, RNA und Proteine - koordinieren das Verhalten aller lebenden Zellen und der Versuch, die Mechanismen zu verstehen, die die Interaktionen zwischen diesen Molekülen steuern und regeln, motiviert die biologische Forschung seit vielen Jahren. Die Einführung experimenteller Protokolle, die solche Interaktionen auf genom- oder transkriptom-weiter Ebene analysieren, hat auch den Einsatz von maschinellem Lernen in unserem Bereich etabliert, um die riesigen Mengen an erzeugten Daten zu verstehen. In jüngster Zeit hat sich gezeigt, dass Deep Learning, ein Zweig des maschinellen Lernens auf der Grundlage künstlicher neuronaler Netze, und insbesondere sogenannte Convolutional Neural Networks (CNNs), vielversprechende Ergebnisse für prädiktive Aufgaben und automatisierte Mustererkennung liefern. Die resultierenden Modelle sind oft sehr komplex und erschweren dadurch Anwendung und Interpretation. Die Möglichkeit, zu interpretieren, welche Muster ein Modell aus den Daten gelernt hat, ist jedoch entscheidend, um neue biologische Mechanismen zu verstehen und zu erklären.

Diese Arbeit stellt daher pysster vor, unsere Open-Source Softwarebibliothek, die es Forschern ermöglicht, CNNs auf biologischen Sequenzdaten einfacher zu trainieren, anzuwenden und zu interpretieren. Wir bewerten und implementieren verschiedene Interpretationsstrategien und zeigen, dass die Flexibilität von CNNs die Integration zusätzlicher Daten über reine Sequenzen hinaus ermöglicht, um die biologische Interpretationsfähigkeit zu verbessern. Wir demonstrieren dies unter anderem durch den Aufbau von prädiktiven Modellen für die Vorhersage von DNA-Protein und RNA-Protein Interaktionen und durch die Erweiterung dieser Modelle mit Strukturinformationen in Form von DNA shape und RNA-Sekundärstruktur. Die von den Modellen erlernten Muster werden dann als Sequenz- und Strukturmotive zusammen mit Informationen über Motivpositionen und Motivkooperationen visualisiert. Durch die weitere Analyse eines künstlichen Datensatzes mit implantierten Motiven veranschaulichen wir auch, wie der hierarchische Musterextraktionsprozess in einem mehrschichtigen neuronalen Netzwerk abläuft.

Zum Abschluss präsentieren wir eine größere biologische Anwendung, indem wir die Bindungsstellen von Proteinen für Transkripte vorhersagen, für die noch keine experimentellen RNA-Protein Interaktionsdaten verfügbar sind. Hier machten uns die umfassenden Interpretationsmöglichkeiten der CNNs auf mögliche technische und systematische Fehler in den eCLIP-Daten aufmerksam, die als Grundlage für unsere Vorhersagen dienen. Dies ermöglichte eine darauffolgende Anpassung der Modelle und Daten, um in der Praxis aussagekräftigere Vorhersagen zu erhalten.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel und Quellen verwendet habe. Ich erkläre weiterhin, dass ich die vorliegende Arbeit oder deren Inhalt nicht in einem früheren Promotionsverfahren eingereicht habe.

Berlin, Dezember 2019

Stefan Budach