The Pennsylvania State University

The Graduate School

CROSS-SPECIES PREDICTION OF TRANSCRIPTION FACTOR BINDING

A Thesis in

Statistics

by

Vandana Agarwala

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

May 2022

The thesis of Vandana Agarwala was reviewed and approved by the following:

Shaun Mahony
Associate Professor of Biochemistry and Molecular Biology
Thesis Co-Adviser

Qunhua Li
Associate Professor of Statistics
Thesis Co-Adviser

Xiang Zhu
Assistant Professor of Statistics

Ephraim Hanks
Associate Professor of Statistics
Chair of Graduate Studies in Statistics

Abstract

Transfer learning, the application of knowledge gained in one machine learning task to a new and related task, represents an attractive approach to studying gene regulation across different species. Here, we apply transfer learning to study the transcription factor (TF) binding motif patterns of four specific transcription factors in up to seven different species. We expect that TF binding preferences should generalize across different species and thus a model trained on one species' genome should roughly be able to predict binding to another species' genome. However, there are some species-specific genomic features, such as repeat elements, which prevent trained models from generalizing perfectly across different species. To account for this, we propose a domain adaptive model architecture which discourages learning of species-specific genomic sequence features. Our results demonstrate that prediction is feasible on species-agnostic genomic features when such an architecture is used to account for domain shifts, i.e. differences in underlying genomic background. Our results also suggest that analysis may be more informative if evolutionary distance is taken into account in prediction.

Table of Contents

List of Figures

List of Tables

Acknowledgements

First, I would like to express my gratitude to those who have supported this thesis work.

I would primarily like to thank Dr. Mahony, my thesis supervisor. I joined the Mahony lab as an undergraduate student in computer science, with little knowledge of or experience in biology; certainly no background in computational biology. Since then, I have learned a great deal of biology, machine learning, and the intersection of the two. I have been able to complete my simultaneous undergraduate degree and master's degree as part of the IUG program. However, gaining experience in Dr. Mahony's lab has offered me much more than academic knowledge. It has exposed me to the world of computational biology research and completely changed the track of my post-graduate aspirations. I am eternally grateful to Dr. Mahony for supporting me throughout my undergraduate years, engaging with me, and patiently teaching me and working with me through this lengthy learning experience. Along with Dr. Mahony, I would like to thank the other members of the Mahony lab, past and present, for always being extremely supportive, helpful, and encouraging.

I would also like to thank Dr. Hannan, Dr. Hanks, Dr. Li, and Dr. Zhu for making this IUG program possible. It is an unusual combination to study Computer Science and Statistics, and proved logistically tough at times: but all of you have made this experience possible for me, for which I am extremely grateful.

Chapter 1

INTRODUCTION

1.1 Background

During early development in eukaryotic organisms, including humans, cells begin to take on specific functions. Although every cell contains the same DNA, each specific cell type expresses different combinations of genes, all of which are encoded in every cell's DNA, to varying degrees. This is conducted by the process of gene regulation, which ensures that genes are appropriately expressed at the proper times, allowing the cell to determine which genes will be active and which will not. There are many possible gene regulation mechanisms, intervening at any step of the gene expression process, beginning with the genomic sequence and ending with the synthesized protein. For example, signaling, transcription, translation, or post-translational modification may all be regulated to different degrees. Gene regulation is also influenced by epigenetic factors, including chromatin organization [2, 40].

One regulation mechanism of particular interest is quite early in the gene expression process,

ahead of transcription itself. We understand that a crucial form of regulation is transcription factor binding to the genome. Transcription factors bind to two different types of genomic regions: promoters (genomic sequence which initiates transcription; must be close to the gene of interest) and enhancers (genomic sequence which enhances transcription; need not be close to the gene of interest). By choosing to bind or not, these transcription factors modulate gene expression by signaling for transcription to begin or not. If transcription occurs, gene expression can proceed, allowing translation and protein formation to take place.

Transcription factor binding depends on many factors, including genomic sequence, chromatin organization, and overall regulatory environment. We largely focus on genomic sequence, seen as one of the biggest determinants of transcription factor binding sites. In computational analysis of transcription factor binding data in particular, we aim to learn from experimentally determined binding data (e.g. ChIP-Seq data) in order to predict transcription factor binding in previously unseen contexts.

1.2 Motivation

Performing ChIP-Seq experiments in the lab under various experimental conditions can quickly become costly and difficult on a larger and larger scale. If we are able to computationally "learn" about transcription factor binding preferences, this develops our insight into the biological mechanisms of gene regulation while reducing the need to perform many costly experiments. This also enables us to make inferences about experimental conditions which may be difficult to test. Machine learning models, and neural networks in particular, have demonstrated increasing success in predicting transcription factor binding sites. They are powerful computational models which have great predictive power based on complex datasets which are challenging to analyze otherwise.

To explore this idea, we propose a cross-species domain-adaptive neural network model which recognizes not only transcription factor binding sites within genomic sequences, but also attempts to adapt across different species. This allows the model to take in data from many different species

and learn about the transcription factor binding signatures which are common and unique across species. We are then able to apply the trained neural networks to previously unseen datasets to test their prediction performance.

1.3 Research Questions

There are several possible downstream questions we can attempt to answer by using transcription factor binding data from several different species to predict in another species.

One is to apply these methods to several closely related species, which may allow us to understand the evolutionary basis for differences among these species. In this case, we train different types of computational models: one which recognizes sequence signatures of regulatory elements across all species of interest, and separate models which recognize regulatory elements in one species at a time. By comparing predictive performance of each of these models, we can better understand how these genomic regulatory regions evolved amongst different species; in other words, how and when they changed in response to differing environmental conditions or behaviors.

Another is simply to apply these same methods towards improving cross-species prediction. Again, we train different types of models, some of which recognize transcription factor binding in one species at a time and one which attempts to recognize transcription factor binding sites but also the species-specific differences. This furthers our understanding of transcription factor binding as a whole, and allows us then to build on this understanding.

For example, even further downstream of this analysis, we can consider applying biological insights to human health and disease. Gaining a better understanding of transcription factor binding and other gene regulation mechanisms also, crucially, helps us understand where and when these mechanisms fail. Gene regulation, in particular, has been implicated in the development of many diseases/syndromes, including cancer, autoimmune diseases, developmental disorders, neurological disorders, diabetes, and others [39]. These conditions can arise due to mutations in regulatory sequences and in the transcription factors, cofactors, and other regulatory elements that

interact with these sequences. Using computational methods to advance our understanding of the biological mechanisms driving disease progression and, by the same token, can lead to potential solutions.

In this work, we focus on the first part of this analysis: cross species prediction of transcription factor binding. Specifically, we use transcription factor binding data from liver cell lines in seven different species: hg38 (Homo sapiens), mm10 (Mus musculus), canFam4 (Canis lupus familiaris), galGal6 (Gallus gallus), rn5 (Rattus norvegicus), rheMac10 (Macaca mulatta), and monDom5 (Monodelphis domestica). We also study four different transcription factors: CEBPA (CCAAT Enhancer Binding Protein Alpha), FoxA1 (Forkhead Box A1), HNF4a (Hepatocyte Nuclear Factor 4 Alpha), and HNF6 (Hepatocyte Nuclear Factor 6). We train neural networks to learn from binding sites on certain genomes and predict on others, and compare performance of different models.

Chapter 2

LITERATURE REVIEW

This chapter serves as a review of prior work done in gene regulation and in machine learning to more deeply understand both fields. More recently, machine learning (specifically deep learning) has begun to be applied to tasks in understanding gene regulation; this is also discussed here. These concepts are crucial for understanding this thesis work.

2.1 Eukaryotic Gene Regulation

We begin by introducing eukaryotic gene expression and regulation: the basic biological concepts which motivate this work, and which we aim to gain a deeper understanding of via computational analysis.

2.1.1 Gene Expression

2.1.1.1 DNA: Genetic Material

Every cell in a eukaryotic organism contains the same genetic material in the form of DNA. The full set of an organism's DNA, or its genome, stores all of the information necessary to define the organism and all of its characteristics and functions. For every organism, DNA is stored in the form of very long, "double-stranded" molecules, or paired polymer chains, found in a double-helical conformation. Each strand is composed of a very long sequence of nucleotide monomers, where each nucleotide can be Adenine (A), Thymine (T), Guanine (G), or Cytosine (C). The nucleotides on opposing strands bind to each other in a predefined way: A binds to T and C binds to G, making the two strands strictly complementary [2, 68, 69, 63, 54]. Due to these strict rules, any one strand of DNA can serve as the template for its opposing strand during DNA replication or, as will be discussed in more detail following, it can serve as the template for RNA molecules, the initial step of gene expression.

Note also that in the processes of mitosis and meiosis, DNA is replicated, which can introduce mutations at any point. Mutations often have no effect, but occasionally they do affect the organism's function; over time, this leads to evolution, giving different species of organisms differing genetic codes and therefore different functions.

First, in order to provide any useful function, this genetic material must be expressed via transcription and then translation. This process, called gene expression, yields synthesized proteins which then catalyze and carry out the functions necessary for life. Each protein is encoded for by one gene, which if expressed, gives rise to that particular protein. However, DNA molecules are very long and encode thousands of proteins, not all of which a given cell needs at all times. Therefore, each cell regulates gene expression in order to produce the proteins that are needed at any given time [2, 68, 69, 54].

This should not be taken to mean that every subsection of a given organism's genome is involved in coding for different proteins, particularly in larger eukaryotes. In fact, often very little of

the genome actually codes for proteins, while much of it is part of what are known as regulatory regions: genomic sequences that specific proteins may bind to in order to control the rate of gene expression. For example, in the human genome in particular, there are a total of about 3.2 billion nucleotide pairs in the full DNA sequence. However, only about 1.5% of the genome actually codes for proteins (encoding about 30,000 genes), while 98.5% of the genome does not code for proteins [2, 54]. Not all of this 98.5% of the genome is useful at all, but some portion of it, though non-coding, plays a crucial region in regulating gene expression.

This regulation is immediately visible when studying, for example, the human body. Every type of cell, from liver cells to brain cells, appears to be vastly different in both structure and function. Still, each contains an identical copy of the genome. It is merely the differences in which sections of this genome are expressed that create the enormous differences in how specific cells behave. This is often controlled by a set of proteins, called transcription factors, which bind to different regions of DNA. In this way, they may either facilitate or inhibit the expression of nearby genes [2, 68, 40, 69, 60, 63, 54]. We explore this further in the following sections.

2.1.1.2 Protein-DNA Interactions

It is important to note that in order to carry out any of the functions described above: DNA replication, transcription, translation, regulation, etc, proteins are required to interact with the DNA. The cell is effectively self-sufficient: these same proteins are coded for by our genetic material and are therefore synthesized by our cells as needed [63]. However, DNA cannot transcribe and translate itself into proteins on its own; rather, every process and mechanism described in the following sections requires a number of proteins to carry out these functions.

2.1.1.3 Transcription

Assuming that a particular segment of DNA is, in fact, a protein-coding gene and necessary for cell function, it is then to be expressed. The first step of gene expression, as mentioned, is transcription of part of one of the DNA strands into its (much shorter) single-stranded analogue,

RNA. Like DNA, RNA is also a polymer composed of nucleotide monomers. In fact, three of the nucleotides (Adenine, Guanine, and Cytosine) are the same as those used in DNA; the key difference is that RNA uses Uracil (U) in place of Thymine. Since RNA is single stranded, it is synthesized by determining the complement to one of the strands of DNA (therefore, it mimics the opposite strand of DNA). The DNA is unwound and opened so that the enzyme RNA polymerase can bind and move along the DNA, base by base, synthesizing the complementary RNA as it goes [2, 68, 69].

The resulting RNA strand then undergoes some post-processing. One of these is modification of each end of the segment, allowing the cell to verify that the full RNA segment was synthesized as intended. It is then "spliced", meaning parts of the middle of the strand, which are not involved in coding for proteins, are removed. The remaining sections can then be recombined in different ways, allowing our cells to "read" the resulting RNA in a variety of ways, producing several distinct proteins from identical original RNA transcripts [2, 68, 69, 63, 54]. The final resulting RNA strand is then sent out of the nucleus into the cell: in particular, to the ribosomes.

2.1.1.4 Translation

The sequence of bases that compose an RNA transcript now needs to be translated into a protein. The nucleotides are now read in sets of three, each set of three nucleotides corresponding to a particular amino acid. There are twenty different amino acids, the building blocks of proteins. One by one, amino acids are added to a growing chain (called a polypeptide chain) as the RNA transcript is being read [2, 68, 69]. In fact, multiple copies of the protein can be synthesized from the same RNA molecule, allowing cells to manufacture much more protein rapidly.

Once the amino acid chain, also called the protein's primary structure, has been synthesized, the protein begins to fold into a more compact and complex formation. It may also be modified by other enzymes or bind with other small molecules, called cofactors. Eventually, the chain begins to interact and bind with itself, folding up into the lowest-possible-energy configuration of itself, forming its secondary and then tertiary structure. Often, particular sections will fold into

particular configurations that are common among other proteins and confer particular structural or functional features to the folded protein [2, 69, 63, 54]. One example of this is the "helix-turn-helix" domain, which as the name suggests, consists of two helices. The first helix, called the "recognition helix", interacts directly with DNA; this is the region that has a higher affinity for a particular DNA sequence, allowing it to bind to the DNA. Meanwhile, the second helix provides additional stability and helps to correctly orient the first helix [63, 54]. The utility of this kind of genomic sequence specificity becomes clear in discussing transcription factor binding in the following sections. In investigating these relationships between protein structure and function, we may categorize different types of proteins into functional categories based on structural features like these.

Proteins may also form complexes along with other folded proteins (which may be multiple copies of the same protein or different proteins), called the quaternary structure. They may also be modified (e.g. cleaved, which is more severe and irreversible, or less extreme modifications, such as phosphorylation) [63]. They may also interact with or even bind to small molecules, called effectors. These molecules, too, affect the protein structure and function. In the particular case of transcription factor binding, for example, they may serve to provide feedback control [2, 69, 68, 63, 54]. All of these interactions give the protein a new shape and expose specific (usually hydrophilic) functional regions outwards to the rest of the cellular material, determining how the protein interacts with other molecules and compounds [2, 69, 63, 54]. Thus, different proteins take on different functions according to their structure.

2.1.1.5 Mutations

Recall that every time a DNA sequence is replicated, whether during cell division (mitosis) or reproduction (meiosis), there is a small chance of mutation as a new strand of DNA is being synthesized. Similarly, there is a chance of mutation during transcription when RNA is being synthesized. While in most cases these mutations have no effect on the resulting protein, they occasionally can. In some cases, a mutation will affect the amino acid sequence of the protein

which is ultimately synthesized, and this may alter the higher-order structure of the protein. When this results in an improperly formed protein, the function of the protein changes. This may be a negative change, in which the organism is unable to survive and the particular mutation dies out. However, it may be a positive change which confers on the organism some advantage compared to the rest of its species. These changes, which improve the chances of survival, remain in the gene pool for that species and accumulate over time: what we call evolution [2, 68]. Thus, studying differences in gene regulation and expression in closely related species and their common ancestors may allow us to understand the evolutionary basis of these changes. Although that is not the focus of this thesis work, the same methodology presented here may be applied instead to a set of closely related species to suggest answers to questions about the evolution of genomic regulatory regions.

2.1.2 Transcription Factor Binding to Promoter Regions

Recall that every cell differentially expresses the many genes encoded for in our DNA. This determines the overall cell type and cell function, but also changes from moment to moment within a given cell, allowing it to perform different functions and adapt as needed. This means that although every cell contains the same genetic material, our bodies are able to perform a vast variety of different functions using seemingly vastly different cells. This process, of expressing different genes at varying levels, is known as gene regulation, and can take place in several different ways at any step of the gene expression pathway described above. The first mechanism, however, is transcriptional regulation, i.e. determining whether or not the DNA is transcribed into RNA at all, and if so, how much.

The primary way that transcription is regulated are by proteins called transcription factors. We know that each transcription factor recognizes a unique, short (5-10 base pair) sequence of DNA, to which it can bind on the genome. Usually, these transcription factors are able to bind to regulatory sequences, generally located close to the gene which they aim to regulate, allowing them to control whether or not that gene is expressed [2, 68, 40, 69, 60]. In fact, they may bind as dimers; this is promoted by the nucleosome structure and helps to increase specificity for particular

genomic sequences (as this increases the length of the matched sequence on the genome, making it statistically less frequent) [2, 54]. Often, too, they bind to the genome but require an additional small molecule to bind as well, often as part of a feedback loop mechanism (i.e. if enough of a particular product has been synthesized, some may bind to the transcription factor to successfully repress expression of that particular gene) [2, 69, 63, 54]. Once a transcription factor monomer or dimer has bound to the DNA, we now turn to how they affect gene expression. In this case, we focus on transcription factor binding to promoter regions: the sections of the genome which initiate transcription. These regions are usually found near the beginning of the gene, so that transcription may begin there.

In some cases, transcription factors may act as activators, which encourage transcription. When activators bind to the sequences they recognize within promoter regions, it is usually because RNA polymerase is less able to bind to the promoter regions on its own. However, when an activator has already bound nearby, RNA polymerase is signalled to bind as well, allowing it to begin transcription [2, 68, 69]. In other cases, transcription factors may act as repressors, which prevent gene expression. When repressors bind to the sequences they recognize within promoter regions, they sometimes serve to block RNA polymerase from binding to the promoter. Recall that RNA polymerase carries out transcription; this therefore prevents transcription. However, repressors can also work via a variety of other indirect mechanisms [2, 68, 69, 60]. For example, they may compete with activators and other transcription factors (rather than competing with RNA polymerase), preventing from the promoter region from being activated and, eventually, transcribed. They may also work by altering the organization of the DNA, either causing it to become more compressed or preventing it from decompressing; in either case, other proteins are unable to access the DNA [2, 40, 54]. (This will be discussed in more depth below when discussing chromatin accessibility.)

In fact, particularly in eukaryotes, genes are often controlled by many transcription factors at once (some of which may be repressors, while others may be activators) to allow for much more nuanced regulation under different cellular conditions [2, 68, 60, 54]. They may bind separately or cooperatively, but this allows the cell to account for current conditions and modulate gene ex-

pression more carefully, including the speed and total amount of gene expression. As eukaryotes are extremely complex organisms, this is very important so that organisms can develop (including developing different types of cells, which maintain their identities, from their origins as pluripotent stem cells) and carry out their varied functions as intended.

2.1.3 Other Regulatory Mechanisms

In this thesis work, we focus only on this one mechanism of gene regulation: the genomic sequences to which transcription factors choose to bind. However, it is also important to note that although each transcription factor recognizes a particular genomic sequence signature, or motif, only a small fraction of the occurrences of any given motif across the full genome are actually bound. This cannot be fully explained merely by nearby sequence variations, so we conclude that there must be other factors which contribute to differential transcription factor binding to the genome.

2.1.3.1 Chromatin Accessibility

In eukaryotic nuclei, DNA does not exist merely as long strands; this would be far too long to be contained within the nucleus. Instead, they are packed tightly into chromosomes (humans have 46 chromosomes in each cell). Each chromosome consists of the long DNA molecule, along with many proteins involved in gene expression and DNA replication; this DNA-protein complex is called chromatin. In fact, this packing of DNA and proteins is highly dynamic, in order to allow particular proteins access to the regions of DNA which need to be transcribed, repaired, etc. The proteins which contribute to this dynamic packing are called histones. The DNA is wrapped around histones, forming what are called nucleosomes, the first level of DNA packing. Nucleosomes are then packed further into chromatin fibers, the next level of DNA packing [2]. Chromatin fibers are highly variable in their "tightness", or how condensed they are. Both of these levels are constantly changing, allowing the DNA to wind and unwind extremely rapidly. This allows the many proteins associated with DNA to access different sections of DNA at all times. Naturally, this also affects

where transcription factors choose to bind, depending on which regions of DNA are available (unwound) at any given time [2, 40, 54]. This feature, which also then affects gene regulation, is called chromatin accessibility.

2.1.3.2 DNA Methylation

Due to the reproductive process, our genetic material contains two copies of each gene, one inherited from the sperm and one inherited from the egg which combine to form the embryo. In the early stages of development, DNA methylation (the addition of a methyl group to the DNA backbone) is used as a marker on the X chromosome. Then as development progresses, DNA methylation patterns change as some locations are demethylated and new locations are methylated. This results in differential gene expression across cell types once early development has concluded [46]. It is more common for methylation to repress genes rather than activate them. This may happen by recruiting proteins involved in gene repression or by inhibiting binding of transcription factors to the relevant regulatory regions [46, 54].

2.1.3.3 Post-Transcriptional Mechanisms

Another regulation mechanism is for transcription to end early or pause after having begun, called transcription attenuation [2]. In this case, the RNA chain which is being synthesized may interact with the RNA polymerase during transcription to pause or end transcription. In some cases, it may be able to resume transcription later if transcription is signalled to re-initiate.

After transcription has occurred and strands of RNA have been synthesized, cells may also control how many of these are translated into protein, how, and how often. For example, RNA may be edited after transcription; this may yield a very slightly different protein with a slightly altered function compared to the original RNA transcript [2, 68, 69, 63]. In fact, recalling RNA splicing discussed previously, RNA can be edited to varying degrees. It may only be edited slightly - changing a few amino acids at a time - or the protein-coding regions may be combined in wholly new ways, creating completely different proteins [2, 68, 69, 63]. Alternatively, RNA strands may

be prevented from leaving the nucleus at all (recall that they must be transported to the ribosome to begin the process of translation) until they are needed. In fact, in mammals it is estimated that only about one-twentieth of all synthesized RNA ever leaves the nucleus [2]. Additionally, there is even a mechanism to regulate translation similar to how we regulate transcription: a repressor may bind to the beginning of the RNA strand, preventing the initiation of translation.

Finally, it is also worth noting that compared to DNA, which is a double-stranded molecule, RNA is only a single-stranded molecule and is therefore less stable. Therefore, while DNA remains in the nucleus for the duration of a cell's life, RNA molecules have a much shorter lifespan. Eventually, they are degraded by the cell and the resulting individual nucleotides are used once again in the process of transcribing a new strand of RNA [2, 68, 69, 63, 54].

2.1.3.4 Interference by Noncoding RNA

Finally, gene expression may also be affected by other molecules of synthesized RNA which do not code for proteins. There are many different types of RNA and although the most common is messenger RNA (mRNA; this type codes for proteins and undergoes translation), there are also several other types which perform different cellular functions. Some of these types of RNA may bind to mRNA and inhibit translation in one of several different ways. In some cases, they merely bind to mRNA and block translation from occurring. In other more extreme cases, they may bind to mRNA and actually slice the mRNA transcript, therefore destroying it and making translation impossible. They may also prevent the synthesis of additional mRNA strands [2, 68].

2.1.4 ChIP-Seq Experiments

Given this biological understanding of gene regulation, and specifically transcription factor binding, we now turn to analysis. Much work in understanding transcription factor binding, this thesis work included, focuses specifically on the sequence specificity of transcription factors. That is, we understand that each transcription factor contains a protein domain which has a particular affinity, or recognition, for specific sequence fragments. We then aim to characterize the sequence

motifs recognized by different transcription factors, leaving aside other impacts (e.g. chromatin accessibility).

In order to perform any analysis, we need experimentally determined genome-wide DNA binding sites for proteins, particularly transcription factors. The main tool used for assaying protein-DNA binding is chromatin immunoprecipitation (ChIP). In ChIP, the DNA-binding protein of interest is first bound to the DNA in vivo. These DNA-protein bonds are then "trapped", or crosslinked, using formaldehyde, which forms a covalent bond between the DNA and the protein [27]. The chromatin is extracted and sonicated into small fragments, usually a few hundred base pairs long. An antibody which selects for the particular protein being assayed is applied in order to immunoprecipitate the desired DNA-protein complexes. The crosslinking is then reversed to release the DNA from the protein, and finally, the DNA sequence is determined [50].

In ChIP-Seq in particular, the DNA sequence is determined by sequencing, using platforms like Illumina [29]. It is thanks, in large part, to the rapid development of next-generation sequencing (NGS) technologies that we are now able to sequence millions of these resulting short DNA fragments in a single run. Previously, this was unimaginable; the predominant experimental method was ChIP-chip, in which a portion of the genome was used and the resulting DNA fragments are hybridized onto a microarray. That is, a DNA microarray is prepared which is spotted with short, single-stranded sequences that covered the section of the genome under analysis. The DNA fragments retrieved from the ChIP experiment are then put over the surface of the DNA microarray. When a ChIP fragment finds a complementary fragment on the microarray, they hybridize into a double-stranded fragment [10]. With this method, however, it proved difficult to then extract the raw data accurately.

However, using NGS technology enables ChIP-Seq, in which the DNA fragments of interest are sequenced directly rather than being hybridized. This allowed for greater accuracy, greater certainty in results, greater specificity, and greater coverage, allowing us to gather precise genome-wide binding data and therefore better identification of the sequence motifs of interest [23].

In fact, as technology rapidly develops, we are now able to perform variants of ChIP-Seq which

confer additional experimental benefits. One is Nano-ChIP-Seq, which allows us to perform experiments using fewer cells (as few as 10,000) due to our increased ability to modify sonication times and antibody concentrations and then amplify the resulting DNA fragments using the polymerase chain reaction (PCR) method. Another is ChIP-exo, which uses a different method (a lambda phage exonuclease) to release the crosslinking of the DNA-protein complex. This results in a shorter DNA fragment, allowing us to identify the protein (specifically, transcription factor) binding site with greater precision. A final example is sequential ChIP, or re-ChIP, which uses antibodies for different proteins to determine genomic binding sites of multiple target proteins. This is useful given that often, DNA-binding proteins work in conjunction with each other and studying multiple proteins at once may provide additional insight [23].

In this work, we use the results from existing ChIP-Seq experiments. The resulting sequence data from ChIP-Seq can then be computationally processed and analyzed; example procedures relevant to this thesis work are explained below.

2.2 Computational Analysis of ChIP-Seq Data

We now present the basic framework of tools needed in computational analyses of ChIP-Seq datasets. This serves as a pipeline processing raw experimental data into a more informative format, suitable for machine learning tasks.

2.2.1 File Formats

In genomic analysis such as the work described in this thesis, we often need to work with sequence data stored in a few different file formats in order to easily perform all computational analyses. Each file format is described below.

2.2.1.1 FASTA and FASTQ

The FASTA format was created to store raw sequence information. For each record, a FASTA file stores two lines. The first line is the FASTA definition line, which consists of a carat character (“>”), followed by a unique sequence identifier. This line also then contains information about the organism from which the sequence was obtained (i.e. at least the scientific name, and optionally more information), and finally it contains a short description of the sequence. The second line then contains the nucleotide sequence itself [21, 51, 52]. In practice, the FASTA format is generally used to store reference sequence data; i.e. data that has been confirmed with certainty and has been retrieved from a curated database.

The FASTQ format is an extension of the FASTA format, allowing the file to store not only the raw sequence, but also a quality score associated with each nucleotide in the sequence [17, 52]. This is because when performing genomic sequencing, the confidence in the identity of each nucleotide varies. Each nucleotide is therefore given a quality score, called the Phred quality score [65, 52]. Given a probability E that a given nucleotide is incorrect, the Phred quality score is calculated as $Q = -10 \log E$, such that a higher score indicates a higher likelihood that that nucleotide read is correct, and a lower score indicates a higher likelihood that that nucleotide is incorrect. The FASTQ file format therefore consists of the following four lines for each sequence read. The first line begins with the “@” character and then stores the same definition line as the FASTA format. The second line, again, stores the sequence itself. The third line begins with the “+” character and then may either be empty or repeat the definition line. The fourth line contains the quality scores. In practice, the FASTQ format is generally used to store short reads from sequencing experiments, such as ChIP-Seq results.

2.2.1.2 SAM and BAM

The SAM format stands for Sequence Alignment/Map. SAM files are generally the result of aligning sequencing reads (e.g. a FASTQ file) to the reference genome (i.e. a FASTA file). The resulting SAM file describes where the particular alignment software was able to match each

sequencing read to the reference genome, if at all. A SAM file must have the following eleven pieces of information about each read [41, 52]:

- The read name; similar to the FASTA definition line discussed previously
- A binary flag which stores some metadata about the read alignment, including whether the read was mappable at all, whether it had alternative mappings, whether it was reversed, etc.
- The name of the reference sequence to which the read was aligned
- The leftmost, or starting, position of the read alignment on the reference genome
- The mapping quality, calculated identically to the Phred score seen previously
- A string describing whether the read was found to be an exact match or some form of mutation compared to the reference genome
- The name of the next read which was aligned
- The position of the next read which was aligned
- The total length spanned by all aligned reads in the file
- The read sequence itself
- The quality scores of the read (identical to quality string in FASTQ format)

There may also be more information if needed, but the above fields are always required. These provide all of the necessary information about sequencing reads which have been mapped to the genome.

Finally, note that BAM files contain identical information to SAM files; they are SAM files which have been compressed into a binary format in order to consume less memory. They are commonly used in practice when performing alignments because this makes them less computationally intensive and time-consuming [41, 52].

2.2.1.3 BED

The BED format stands for Browser Extensible Data. BED files are another way of storing mapped genomic sequences; they store less information than SAM/BAM files but are more versatile. In fact, they need not be used specifically to store mapped, or aligned, genomic sequences, but they can also be used to store the position of any genomic feature of interest. They have three required fields: (1) the chromosome name (e.g. chr4), (2) the starting position of the genomic feature on that chromosome, and (3) the ending position of the genomic feature on that chromosome. There are also additional optional fields, including a feature name, a score (e.g. the prominence of the genomic feature of interest), the strand referred to by the positions (i.e. forward or reverse), etc [57]. In practice, BED files are very widely used for computational analysis of genomic data because they simply store genomic positions; these positions can refer to any feature of interest.

2.2.2 Alignment

An extremely fundamental question in genomics, the alignment problem is of mapping two or more sequences to each other in a way that maximizes their similarity. The goal is to identify the similarities between the sequences, which may help us infer different types of relationships between the sequences (for example, evolutionary or functional similarities). This is generally done by matching sequences as closely as possible and inserting gaps between nucleotides so that identical or similar nucleotides are aligned with one another. There are a few key applications of this problem and methods of solving it.

First, we consider the sub-problem of global alignment, or in other words, aligning two full long sequences with one another and forcing the alignment to span the full sequence length; the first such algorithm was the Needleman-Wunsch algorithm [48, 52]. This was used early on to understand homology (the inheritance of particular genes from a common ancestor), including orthology (sequences which share a common ancestor but have since diverged) and paralogy (genes which are related via duplication in their common ancestor). This is particularly useful when

the whole sequence is expected to be quite similar, as in the case of homologs. This concept was then specialized further to calculate local alignments, which aim to find the most similar subsequence between two longer sequences; the first such algorithm was the Smith-Waterman algorithm [61, 52]. This is more useful for sequences which are expected to be dissimilar, but still contain some regions of similarity which may be of particular interest. There are also hybrid methods, which aim to maximally partially align two sequences. One case of this is of particular interest in this work: mapping short genomic sequences to a full chromosomal sequence or even a full genomic sequence. This allows us to determine the position of the sequenced read within the full genome. In this case, we want the short sequence to be fully (globally) aligned and the long sequence to be partially (locally) aligned.

Early on when applying genomic alignment algorithms, these early algorithms worked well. However, as longer DNA sequences became more readily available, these initial methods quickly became too inefficient and computationally intensive to use in practice. Additionally, these methods considered only simple adjustments to alignment, such as insertions, deletions, and substitutions. Over time, more powerful methods were needed, which were both more efficient and allowed for more rearrangements, such as inversions, transpositions, and duplications. Therefore, computational methods like the Burrows-Wheeler transform [12, 52] were developed to use in methods such as the Burrows-Wheeler Aligner and the Bowtie aligner. Currently, a vast number of alignment algorithms exist, evolving alongside our sequencing capabilities, as we are able to generate more sequence data [42].

2.2.3 Peak Calling

When we have aligned sequencing reads to the genome, we then have what is essentially a histogram of the whole genome. Each bin in the histogram, or short region of the genome, is assigned a read count determined by the number of reads mapped to that region. This gives us a distribution of reads across the full genome. We are now interested in locating the “peaks” of this distribution, or the regions that have a statistically significantly high read count. These are

the regions which contain the majority of the reads which were precipitated out using the specific antibody used in the ChIP-Seq experiment. Our interest is often in identifying these high-signal regions to gain biological insight into the binding of a particular protein of interest to the genome. Many different peak calling algorithms can be used, but the choice of algorithm may affect results due to widely varying strategies used to determine the locations of peaks. Over time, peak calling methods have evolved to become more statistically sophisticated [45], summarized here:

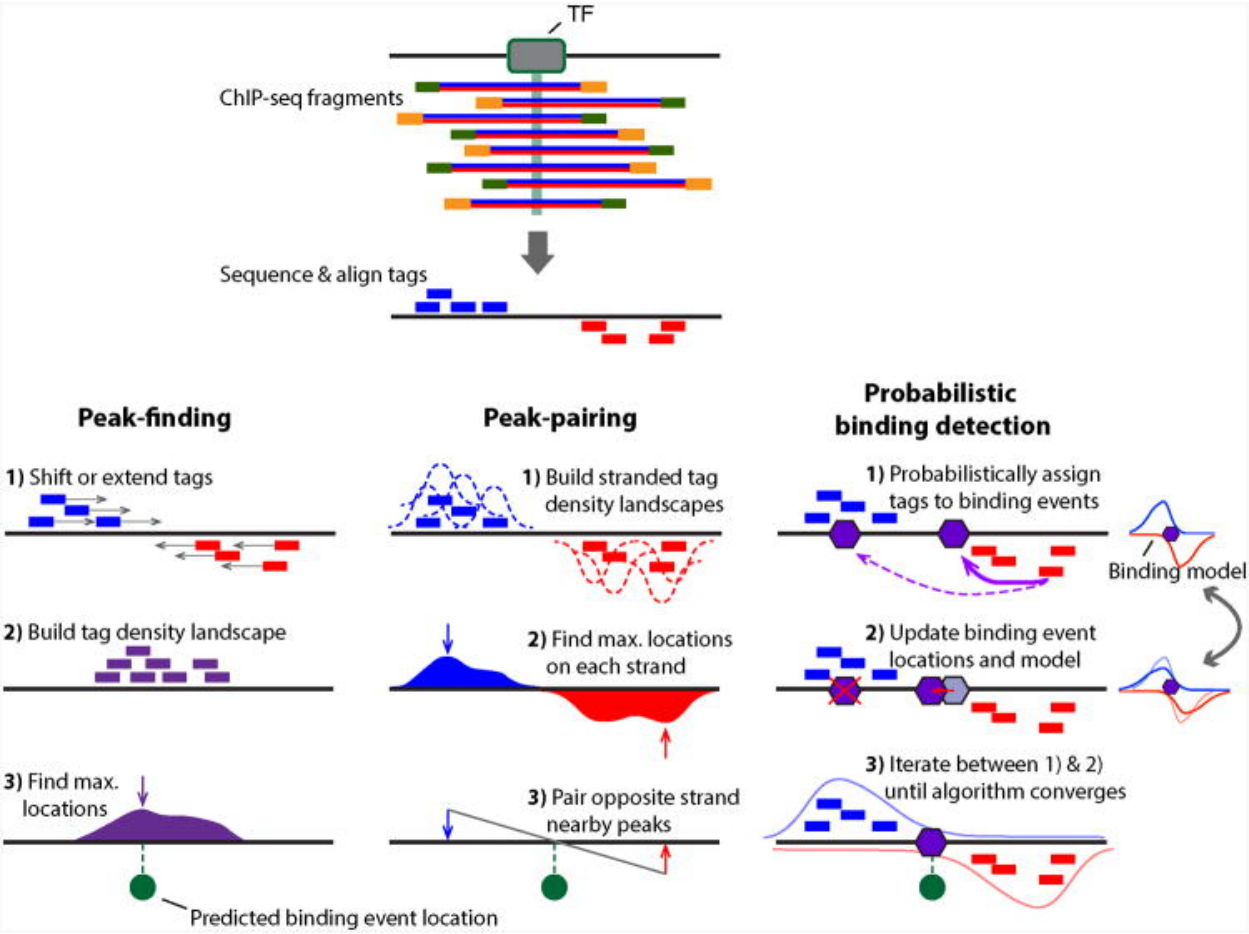


Figure 2.1: Overview of evolution of peak calling algorithms [45]

The earliest peak calling algorithms used a "peak-finding" approach. First, they merged the read alignment results from both strands of the full genome. Reads are sequenced from the 5' end of a strand of DNA, so merging the findings from both strands is important in order to get a clear picture of binding events. Merging in this way essentially extends reads to make longer DNA fragments. This yields a more complete distribution of reads across the whole genome, irrespective

of strand direction. Peak finding methods then simply take the locations with the maximum signal to be the binding event locations. MACS is a prominent peak caller which uses this method [70].

The next generation of peak calling methods took a different approach. Similarly to peak-finding, they considered both strands of DNA to deduce the location of the ChIP-Seq peaks. However, instead of combining the results from both strands to simulate longer immunoprecipitated fragments, they maintained the strand-specificity and considered the distribution on opposite strands. Particularly, this distribution is expected to be bi-modal around the true peak location due to the symmetry of the two strands. Therefore, they detected peaks on the opposing strands and concluded that the true peaks, or binding locations, were at the midpoints between peaks on opposite strands. This method is used by peak callers such as GeneTrack [1]

Finally, probabilistic binding detection methods have since been developed, like GPS [26] and, later, MultiGPS [44]. These take a radically different approach to peak finding, which is applicable in more complex conditions when the locations of peak signals are not so clear. These methods aim, instead of "finding" regions of maximum signal, to estimate the binding locations that would have produced the particular given read distribution. They begin by making a guess as to where the binding event locations may be and probabilistically match the experimental sequenced reads to the nearby estimated binding events with likelihoods based on proximity. Binding location estimates are then updated to achieve a better fit with the sequenced reads, and reads are once again matched with binding event locations. This continues iteratively, called an Expectation-Maximization algorithm. As time goes on, binding events become better matched with sequenced reads and excess estimated binding events (i.e. those with very few reads matched to them) are eliminated. Eventually, the algorithm converges and the binding event locations are optimized in an arrangement which could possibly have given rise to the given read distribution.

2.2.4 Blacklist Regions

Our sequencing capabilities have been transformed and become far more powerful in recent years, but there remain some limitations. One of these limitations are what are now known as

"blacklist" regions [4], locations that have an anomalous or artificially high signal after sequencing, regardless of the cell line or experimental conditions used. This generally is the result of difficulty mapping reads to the genome. This can be because of difficulty in assembling those regions of the reference genome itself; for example, in highly repetitive regions. In this case, it is very difficult to determine how long these regions are, and where in a repetitive region a given read should be aligned. This may lead to incorrect (possibly artificially high) read distribution in these regions and may be considered a peak by the peak calling algorithms described above: an undesirable outcome. These regions should be filtered out; otherwise, analysis results may be incorrect or less biologically informative.

This was found to occur in many genomes; not only the human genome. As part of the ENCODE project, these blacklist regions were also compiled for several different species and genome assemblies commonly analyzed via ChIP-Seq experiments (hg19/hg38, mm9/mm10, ce10/ce11, and dm3/dm6). This was done by detecting which regions maintained a high signal regardless of experiment, including input sequences used as the control set for ChIP-Seq experiments. The ENCODE project also found that oftentimes, lower-quality ChIP-Seq experiments have even more signal enrichment in these blacklisted regions, and therefore use this as a quality control metric to ensure the quality of experiments.

These blacklisted regions represent a small fraction of the genome but are often anomalous in their high signal. Excluding them from further computational processing is crucial in order to obtain accurate and high-quality results.

2.2.5 Non-uniquely Mappable Regions

Another related but not identical concept is that of uniquely (and non-uniquely) mappable regions on the genome [32]. Similarly to blacklist regions, these are genomic regions which can skew results of later computational analyses due to inaccuracies and uncertainties. Similarly, too, these regions can sometimes arise due to inconsistencies in the process of aligning reads to the genome, which can be the result of uncertainties in the reference genome itself.

When reference genomes are assembled from a vast number of reads, regions all along the genome can be given a "mappability" score, which indicates how well it can be uniquely mapped to by experimentally sequenced reads. When sequenced reads can possibly be mapped to multiple locations on the genome (i.e. these locations have low mappability), this indicates that these reads offer less certainty in downstream analysis, as we cannot be exactly sure which location on the genome they are actually from. The ability for reads to map with equal plausibility to multiple locations on the genome diminishes certainty in alignment results, which then invalidates peak calling and all results which follow. Therefore it is crucial to know the mappability of every region of the genome to, again, ensure the quality of computational results. Tools like Umap [32] and GenMap [53] have been developed to efficiently compute mappability.

Umap approaches the problem essentially by brute force. For a given read length k, it generates fake "reads" of all length-k fragments of the genome, or k-mers, and attempts to align them to the genome using the Bowtie aligner. It then stores the location of every "read" that could be uniquely mapped to the genome.

GenMap recognizes that this method may not be sufficiently efficient (due to the process of generating all possible k-mers and aligning them) nor satisfactorily accurate (it may not sufficiently account for mismatches in alignment). Thus, their strategy is to search for occurrences of any given k-mer in the full genome sequence, allowing for an acceptably small number of errors, or mismatches. This process is made more efficient by (1) optimizing the search scheme and (b) recognizing the redundancy of searching for substantially overlapping k-mers; overlapping k-mers are highly similar and therefore searching for them separately unnecesarily duplicates computations. The authors note that these optimizations vastly decrease running time, while still delivering highly accurate results.

Like the blacklisted regions, these non-uniquely mappable regions represent a small fraction of the genome but often present inaccurate read distributions in their vicinities. Again, excluding them from further computational processing is crucial in order to obtain accurate and high-quality results.

2.3 Deep Learning

Finally, we must introduce the concepts of deep learning in order to understand the aspect of this thesis work that utilizes machine learning and, particularly, neural networks. Deep learning is an increasingly common technique in biological research and specifically in work done in understanding transcription factor binding.

2.3.1 Deep Learning Methods

Deep learning is a sub-field of machine learning, a field of computer science in which machines "learn" over time to perform tasks. That is, machines aim to recognize patterns and then attempt to imitate these patterns. Most commonly, given two datasets, a machine learning algorithm attempts to imitate the relationship between the two datasets so that it can use one dataset to predict the other: a truly profound feat [67].

Machine learning is used to perform several different kinds of tasks; a few common ones are as follows. Perhaps the most common is classification, used heavily in this work. In a classification task, the algorithm is asked to categorize each input data point into one of several categories (or, similarly, provide a probability of falling into each category). Another common task is regression, in which the algorithm is to predict some other output value (not necessarily a probability or classification) based on the input. A relatively new task is synthesis, or generation, in which the algorithm attempts to generate new data which is similar to the input examples given. And of course, there are many, many more tasks which machine learning algorithms have been employed to solve [25, 15, 9]. This work will focus primarily on classification tasks, and on deep learning methods for achieving classification.

The concept of deep learning, in fact, is developed using insights from our study of the human brain. Our brains, from a very young age, are capable of developing broad intuitions, making decisions, drawing connections, and solving complex problems. Though deep learning does not precisely model the function of a human brain, it borrows a number of concepts (relating to the way

neurons fire and signal one another) to learn the relationships between large sets of data [11, 25].

The general approach that machine learning models take is that of trial and error. That is, they are shown multiple examples of the data they are trying to predict on, and asked to make a prediction. This prediction is compared to the correct value, or the truth; a correct answer reinforces the model, while an incorrect answer forces the model to reevaluate and modify itself [67, 11, 9]. Over time if it is shown enough examples, the model is expected to become highly accurate: just as we are able to quickly recognize patterns around us.

In more detail: in our brains, neurons are designed to receive information from other neurons, process this information, and send along the result to other cells. Deep learning models attempt to mimic this overall structure by receiving inputs, performing different calculations on the inputs, and transforming them into some output. Also like our brain, a full model cannot be built using only one neuron. Instead, there are several layers of neurons. Data flows from layer to layer until the original raw input is converted into a conceptually useful and interpretable output [67, 11, 25, 15, 9]. This framework is called a neural network.

2.3.1.1 Foundations of Prediction

In a neural network, different types of computations are stacked in various ways to build a model which, given some input dataset, aims to produce a targeted output for each input entry, transforming the input dataset into a new dataset of some kind. This input data tends to be information that we can measure and know with certainty; we transform it into valuable output data which requires analysis, i.e. something we want to know [67, 15]. In order to learn the desired transformation, the model needs to tweak various parameters which determine the many computations it performs in sequence. Over time, the model tries different values of particular parameters and continually adjusts them depending on whether the resulting output was correct or not. Eventually, it is able to settle on the optimal configuration which gives the "right answer" as often as possible [67, 15]. We first examine how machine learning models perform this initial transformation; i.e., the prediction step.

This step may use many different statistical methods for representing data, most notably linear regression, maximum likelihood estimation, Bayesian estimation, and principal components analysis [25, 9]. Each of these techniques allows the model to take in larger quantities of data and produce some kind of estimate as output; exactly how this is done depends on the task of choice.

To this end, there are a few different types of computations that neurons often perform. Some neurons perform linear computations, e.g. given a set of inputs, they merely yield a linear combination of those inputs [11, 25]. This is a simple computation, but not particularly complex or powerful [67]. Therefore, we introduce nonlinear neurons, which transform their inputs more radically. One such example is the sigmoid neuron, which transforms the input such that when the input is very small (very negative) the output is close to 0, when the input is very large (very positive) the output is close to 1, and in between the output takes on an S-shape, going from 0 to 1. These are commonly used to output individual probabilities [67, 25]. Finally, we may also want neurons within a layer to introduce nonlinearity *and* to depend upon each other. A common such example is a softmax layer, which introduces several neurons that *together* output probabilities [11, 67, 25]. For example, if we aim to classify an object into one of five categories, we would have five outputs, each describing the probability of the object falling into that category, all summing to 1. Ideally, in this case, one of the values is very close to 1 (indicating that the object is very likely of that particular category), while the other values are nearly 0. We discuss different types of neural network layers later in this section.

When building a neural network, we define exactly which types of computations we would like the neural network to perform. It is then the responsibility of the computer to tweak the parameters involved in these computations to find the optimal solution. In the prediction step, the model is given a large input dataset. The model takes small chunks of that dataset, one at a time, and feeds these input chunks through the predetermined sequence of computations. This produces an output dataset with values corresponding to each entry in the input dataset. The model now needs to compare these values to the "true" values, or the *desired* output [67].

2.3.1.2 Evaluation and Correction

Once we have built a model which can yield predictions, we must compare these predictions to the true target values and evaluate the performance of the network. There are several different ways to measure this error; different metrics, called loss functions, have advantages and disadvantages in different contexts.

Essentially, loss functions aim to measure "how wrong" the model's predictions were, and in what way they were wrong. Of course, the aim is to minimize the loss value. There are several different common ways to measure loss, or error, and custom methods can also be defined. The specific choice of loss function greatly impacts how the neural network learns [67]. This is because the neural network's only aim is actually to minimize the loss, because this is how its accuracy is really measured, and this is what prompts it to later readjust its predictions. Therefore, the choice of loss function is crucial, as the network will do anything that it can to minimize the loss function [67, 11, 25, 15, 9]. For common prediction tasks like the ones used in this work, the loss functions are fairly standard.

For example, regression problems commonly use mean-squared error, or the square of the difference between the prediction and target. This is a simple metric which checks how close the estimate, or prediction, is to the true value. Meanwhile, binary crossentropy loss is used for two-class classification problems. Crossentropy is a metric used to measure the distance between two probability distributions; it is therefore useful in classification settings because these are models that output probabilities [15]. In this case, crossentropy is used to measure the difference between the true distribution and the model's output predictions. Similarly, categorical crossentropy is used for multi-class classification problems; this is the same idea as binary crossentropy, but it is designed for use when there are more than two categories of classification. In this work, we use binary crossentropy loss (in the binding prediction setting) and categorical crossentropy loss (in the species prediction setting).

Finally, the model readjusts to reduce the error in its predictions. It does this using a process called gradient descent: in its most simplified formulation, gradient descent is the process of iter-

atively trying several different adjustments in order to determine exactly which direction to adjust so that error is minimized [11, 67, 25, 15, 9]. Here, too, we perform statistical estimation (e.g. maximum likelihood estimation) to determine the optimal level of adjustment [25].

So far, we have only discussed computations which propagate forwards through the layers of the neural network. However, now, in adjustment, different computations must now go backwards. The model now attempts to determine how much of a role each neuron played in contributing to the total error, and modifies each neuron's computation from there. When this is applied to a neural network with multiple layers, this process is called backpropagation. That is, the neurons' computations are modified, layer by layer, tracing backwards through the neural network based on the connections between neurons in different layers [11, 67, 25, 15, 9].

2.3.1.3 Convolutional Neural Networks

In this work, we use a particular type of neural network called a convolutional neural network (CNN). Convolutional neural networks originated in the field of computer vision in an effort to mimic the human sense of vision (notably, that we can see our surroundings and quickly recognize and identify specific objects) [11, 25]. This is a difficult task using the basic machine learning framework and instead required more powerful methods. CNNs, therefore, examine input data for patterns, or features, which appear to have predictive power.

They primarily do this using the concept of filters. Each filter is a small "feature detector" which scans through the input data looking for a particular feature. It then creates a "filter map", which indicates the locations within the input where that feature was found. This is called a convolution, in which a filter is applied to an entire input [11, 67, 25]. Convolutional filters are generally applied in sets of multiple filters at once, and then are often layered further, being applied on the results from previous sets of filters. This allows a CNN to effectively learn more complex patterns as the depth increases.

Convolutional neural networks are now widely used outside of image processing as well. Of particular relevance here is the use of CNNs in sequence analysis. The idea here is quite similar to

before; in this case, the network examines subsequences of the given input sequence to determine patterns [11, 67]. For example, when looking at genomic sequences, a particular convolutional filter of size 5 may look for the specific subsequence "ATTCT". Then, as with images, given several sets of convolutional filters, the network is slowly able to build up a deeper understanding of which more complex sequence features are informative in the prediction task at hand.

2.3.1.4 Neural Network Layers

There are several different types of neural network layers (sets of neurons, all of which take in the same inputs, whose outputs depend on one another). We discuss a few commonly used layers and computations, relevant to this work, here:

- Convolutional Layers. Here, many convolutional filters (as described previously) are applied, creating a large feature map with several layers [11, 25]. When applied in sequence, convolutional layers allow a model to decipher more and more complicated features of the input data. Most deep neural networks employ several convolutional layers, each with many filters. Convolutional layers are generally followed by activation functions. These serve to help decide if a particular neuron in the neural network should fire or not; the neuron is "activated" when that particular convolutional filter pattern is detected. The sigmoid and softmax functions, discussed previously, are two very common examples of activation functions. The choice of activation function depends on the exact prediction task at hand but in general, activation functions allow us to interpret the output of neural networks more easily [67].

- Long Short-Term Memory (LSTM) Layers. A key component of LSTM units is the memory cell. This holds information learned over time, allowing the network to maintain a "memory" over many training steps as it iterates through the whole input [11, 25]. This allows the network to not only learn the patterns at a particular point within the input sequence, but also to learn how surrounding subsequences may impact a given subsequence. That is, the neural network is able to learn about the input data as a whole, rather than learning only in discrete windows.

- Pooling Layers. These are used to consolidate and reduce the size of the data. When applying different convolutional filters within a convolutional layer, the volume of data involved can grow quite quickly, as each filter generates its own feature map. Pooling layers condense these feature maps into smaller versions which still retain the crucial pieces of information, but are much more compact [11, 25, 67].

- Dense, or Linear, Layers. These perform linear combination computations, as mentioned previously. They generally serve to synthesize all of the neural network output so far and change the dimensionality of the data so that we may more easily define the relationship between the input and output data.

- Dropout Functions. These serve to prevent overfitting by simply randomly leaving out inputs from the previous layer. This essentially requires the model to train using only random subsections of the neural network [67, 25]. Overfitting will be discussed in more detail later in this section.

In building a neural network, we aim to stack different computations in a strategic manner, hoping to maximize what the neural network can learn from the input dataset. For example, activation functions commonly follow convolutional layers in order to distill what the convolutional layers ”learned”. Pooling layers commonly follow sets of convolutions/activations in order to condense the vast volume of data generated in the convolution stage. Meanwhile, dense layers tend to be found closer to the end as we try to gradually reduce the volume of data to create reasonable final predictions.

2.3.1.5 Variable Encoding

Note that there are several different ways to encode data in a format that is useful for a neural network.

The simplest data is that which is simply numerical by nature. The relationship between different values of a numerical variable is very clear and it is easy for the network to perform any type

of computation on purely numerical data. However, in many cases, data is not numerical; rather, it is what is called categorical.

Categorical data are types of data which may be divided into groups; for example, race, sex, or educational level. These types of variables do not readily lend themselves to a numerical interpretation; however, because machine learning models work by performing complex computations on input data (and then inverting these calculations), they require all input and output data to be numeric. This poses a challenge of data representation, commonly solved in one of a few ways.

One way to handle categorical data is simply to convert it to numeric data. This is reasonable when the data has an ordered relationship: in the case of educational level, for example, we may encode ”none”, ”high school”, ”Bachelor’s”, ”Master’s”, and ”PhD” as 0, 1, 2, 3, and 4, respectively. However, this is often not possible.

A more generally applicable method of representing categorical variables is as a one-hot encoding. In this method, each possible category is represented as a separate variable. If an input falls into a particular category, that entry is 1; otherwise, it is 0. In bioinformatics, this is the most commonly used method for representing genomic sequence data, since different base pairs do not have an inherent ordering and yet, a sequence of letters cannot simply be processed by a machine learning model.

2.3.1.6 Issues to Avoid

One issue with neural networks as described above is that they can be quite complicated, and the combination of a neural network consisting of many neurons along with complex datasets can quickly increase the total number of parameters we are attempting to determine in the network. In fact, by building an overly complicated model, we may easily fit the model to our exact training set simply because the model has so many degrees of freedom. However, a model like this does not generalize well to new data points, while a significantly simpler model would generalize far better. This phenomenon is called over-fitting, a common issue when deep learning models by nature contain many neurons [11, 67]. This is therefore a trade-off which must be handled: models need

to be sufficiently complex to be powerful enough to interpret large datasets, but not so complex that they begin to over-fit to the data. In order to ensure we are evaluating our models effectively and preventing over-fitting, we split our data into separate parts: a training set and a test set. This allows us to evaluate model performance on a new set of data it has not seen during training, which shows us whether or not it is able to generalize well. In fact, we actually repeat this evaluation at the end of every epoch, on another subset of data that we call the validation set. This allows us to "check in" at the end of every epoch to periodically make sure we are not over-fitting. We may even stop model training early if we reach a point where training accuracy continues to increase but validation accuracy does not; this would indicate overfitting [11, 67, 25]. Alternatively, recall dropout functions mentioned previously: these are a common technique for preventing overfitting. This is implemented by randomly selecting some neurons to become inactive during training. The intuition here is that the network is forced to accurately learn, even without some pieces of information, so that it is not too dependent on any particular combination of neurons [11, 67, 25].

Another issue that may arise is that of imbalanced data. In classification problems, as discussed previously, each entry in the input dataset belongs to a particular category, or class. In a binary classification problem, when there are two classes, it may be the case that one class is vastly larger than the other; this is the problem of balanced data. Whether this occurs due to a lack of data, difficulty in collecting data, or simply because of the underlying ground truth, this feature can make the deep learning task quite difficult. Because of the imbalanced prior probability of a sample falling into either class, when given imbalanced data, machine learning models will typically over-classify data samples into the larger category, leading to a high rate of misclassification [30]. In fact, the misclassification rate is higher among the smaller subset of the data, which is often the subset of interest. There are several different techniques used to combat imbalance, each of which have their own advantages and drawbacks. Some are data-level methods, which modify the subset of samples used for training the machine learning model. For example, we may undersample from the larger subset (which involves discarding data) and/or oversample from the smaller subset (which may lead to overfitting). Alternatively, we may use algorithmic methods, which alter the

learning process of the neural network in such a way that emphasizes the importance of the smaller class of data, and de-emphasizes the larger class. Finally, there are methods which alter both the data sampling and the learning process for optimal results [30].

2.3.2 Deep Learning in Transcription Factor Binding

Deep learning methods have become a commonly used and effective tool for understanding transcription factor binding to the genome. Historically, sequence binding patterns of DNA-binding proteins had been characterized by position-weight matrices (PWMs) [62], but in recent years more complex and computationally intensive techniques have grown in power and popularity. Particularly, deep learning methods have been adapted to the task of predicting DNA-binding protein sequence specificities - including transcription factor binding motifs [3].

Over the last few years, these methods have achieved higher and higher levels of accuracy and success [8], using a wide variety of deep learning techniques [13]. Deep learning offers several benefits, including the ability to quickly learn from much larger quantities of data, a greater ability to generalize across datasets, and a higher tolerance for imperfect data [3]. Genomics datasets are extremely large and complex, and such algorithmic tools are needed to gain inferences from these datasets. In fact, an additional benefit of machine learning methods is that they are designed to automatically detect patterns in data, rather than depending on domain expertise and hypotheses [20].

Deep learning on genomic sequence data has therefore been applied not only to determining transcription factor sequence motifs, but also to understanding other protein binding, DNA accessibility, histone modifications, DNA methylation, and splicing patterns [33, 55]. It also offers interesting downstream applications, giving rise to the ability to analyze genetic variants that can affect transcription factor binding and gene expression [3]. A trained model can evaluate the influence of each nucleotide on regulatory function of a given sequence [33]. This gives a very detailed view of sequence-function relationships and the impact of possible variants of interest.

It is important to note that although deep learning models have shown a high degree of accuracy

in predicting transcription factor binding sites, it still remains challenging to interpret these models and ensure that they are being trained effectively: as discussed previously, machine learning models' predictions and feature extraction are highly dependent on the input data and training parameters [20]. High performance on benchmark datasets does not necessarily imply that the model has learned the biological ground truth [36], and thus an important branch of continued research is currently focused on interpretability of deep learning models in bioinformatics.

2.3.3 Domain-Adaptive Models

Transfer learning is a field of machine learning in which a machine learning model is built based upon one problem, the source problem, and then applied to a different, but related, problem, the target problem. For example, we may train a model to detect people visible in daytime (light) images, and then apply it to detecting people in nighttime (dark) images. This idea is most useful in settings where large amounts of data are available for the source machine learning task, but not so for the similar target task. In general, if the two tasks are fairly similar, we can expect the model to perform reasonably well on the target task: at the very least, it should be easier to adapt this model to the new task than to train a brand new model.

Domain-adaptation is an emerging sub-field of transfer learning, in which again, we aim to apply a model trained on one dataset to a different dataset. In the case of domain adaptation, though, the two datasets are fundamentally very similar, having the same "feature space" (e.g. genomic sequence data). However, the distribution of the response we aim to predict is different [64]. In this work, this means that while for every species the input data are genomic sequences, the set of sequences which are found to be bound vs unbound will be different. In many scenarios, we have data from both the source and the target datasets, but only have labels for the source data, while the target data is unlabeled. In this work, this corresponds to using the transcription factor binding labels for some species during training, without using the labels on the held-out target species. Additionally, particularly relevant to this work is the multi-source domain adaptation problem, in which there is data from multiple source domains and only one target domain [64].

Several different types of methods have been developed in recent years to handle this question: including sampling methods [28] and domain transformation methods [49]. One class of approaches towards the domain-adaptation problem is to build a sort of mapping between the source data and the target data, such that a model trained on the source data can be applied to the target data using this mapping. Recent research [37] has demonstrated that an effective approach may be to attempt to learn the features that do not discriminate between the source and target domains. As training progresses, the model learns the features that are highly predictive, but indiscriminate between the source and target domains, i.e. the predictive features which are common among both. Using this strategy, the model may be directly applied to the target domain [6].

This is achieved by creating two models operating jointly on the same data: one predictor which is used during both training and testing, which learns the primary classification task, and one discriminator which is used only during training to differentiate between the source and target domains. Over the course of training, these two models work adversarially to one another, so that while highly predictive features are being learned, domain-specific features are *not* being learned, and the resulting model predicts based on domain-invariant features [24]. This is called a domain-adversarial neural network.

Specifically, one method that has been proposed to accomplish this adversarial method during training is a "gradient reversal" layer. During model training, the gradient reversal layer has no effect on the forward computations of the neural network. During backpropagation, however, it negates, or reverses, the gradient during the gradient descent process mentioned previously [24]. This layer is implemented at the start of the domain-differentiating branch of the neural network, serving to "unlearn" these features which discriminate between the source and target domain. Therefore, while the (unmodified) predictor continues to learn the features which are highly predictive in the original classification task, the model simultaneously is prevented from learning the features which are specific to either domain, eventually settling only on the predictive features present in both domains [24].

Chapter 3

METHODS

In this work, I trained deep neural networks to learn the binding sites of four different transcription factors (CEBPA, FoxA1, HNF4a, HNF6) in seven different species. (Note that data exists for all seven species only for CEBPA; for the other three transcription factors, data exists for five of the seven species.). For all genomes and transcription factors, I built and trained four different types of neural networks: two conventional models and two using a domain-adaptive architecture. I then evaluated these models on each species to predict the regulatory regions in each genome. This was done in order to evaluate how accurately a neural network trained on one species can predict transcription factor binding sites in another species.

This required a few major components:

1. Collecting public data from ChIP-Seq experiments done elsewhere
2. Preprocessing of data to present experimental binding sites in a more digestible format conducive to training neural networks
3. Splitting data into training, validation, and testing datasets

4. Training neural network models using different architectures to maximize accuracy

5. Evaluating trained models on genomic sequences for further analysis

Each of these components is described in further detail below.

Note also that all computational analysis and model training was done on the Pennsylvania State University Institute for Computational and Data Sciences (ICDS) Roar supercomputer, using a GPU allocation assigned to Dr. Shaun Mahony.

3.1 Data Sources

First, experimental ChIP-Seq data was gathered from two similar works: one which aimed to demonstrate the interspecies differences in transcriptional regulation [58] and one which aimed to determine the degree to which other species share regulatory regions with humans [5]. This experimental data was gathered in the form of FASTQ files.

```
@ERR005115.1 30WN1AAXX:4:1:964:112 length=44
GGGTTAGGGTTAGGGTTAGGGTTAGGGTTAGGGTTAGGGTTAGG
+ERR005115.1 30WN1AAXX:4:1:964:112 length=44
>>><>=>>><>;>>><>;>:><>=><:<><7>7<9/;5)5;/,;
@ERR005115.2 30WN1AAXX:4:1:565:766 length=44
GGGTAAATGAAACACAACTTTTACAAAAACAACAATATTCAATA
+ERR005115.2 30WN1AAXX:4:1:565:766 length=44
<<$<6$44:9)<$6%489,%-<.%%%/81'3$)50737,"((3'
@ERR005115.3 30WN1AAXX:4:1:899:494 length=44
GAATTAATCAGTGAAACTAATGCAATAAACATATTTTCTAACAG
```

Figure 3.1: Example section of a downloaded FASTQ file (ERR005115.fastq) containing ChIP-Seq data representing the binding sites of the transcription factor CEBPA to the Gallus gallus genome.

Above, each read in a given file comes from the same ChIP-Seq experimental results, and the fastq format tells us the read sequence, read length, and the quality score of each nucleotide that has been sequenced. In the example above, the reads represent CEBPA binding sites to the Gallus gallus genome; once CEBPA was allowed to bind to the genome and the DNA samples were fragmented, a CEBPA-specific antibody would have been used to imunoprecipitate the DNA-CEBPA complexes. These reads are examples of the results once these fragments were sequenced.

Next, genome reference information was gathered from the publicly available UCSC Genome Browser, a project of the International Human Genome Project consortium which has created assemblies of many different genomes as well as tools for visualizing and analyzing this data [34]. This reference information includes full genome reference sequence files in FASTA format and the lengths of each chromosome.

3.2 Data Preprocessing

3.2.1 Alignment

The experimental ChIP-Seq data that has been gathered represents many short subsequences of the genome where a given transcription factor is found to be bound. (In the case of the control ChIP-Seq data, these are simply subsequences of the genome, irrespective of transcription factor binding.) The next step was then to align these subsequences to the full genome to determine their position within the genome. Alignment was performed using the Bowtie aligner [38]. This produces BAM files, which are used to describe alignment data. That is, for each read, they contain the key information about alignment: read sequence, read quality, and the location to which the sequence was aligned (chromosome number and start position). BAM files are in a format that is not human-readable, but can be visualized computationally, such as in the example below:

```
 951       961       971       981       991       1001
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
 KCKCCCCAAKCCCAATGCCAATCCTCATCCCATCATCCCATCATCCCAATC
 acaccccaatcccaatgccaatcctcatcccatcatcccatcat
      ccaatcccaatgccaatcctcatcccatcatcccatcatcccaa
        aatcccaatgccaatcctcatcccatcatcccatcatcccaatc
```

Figure 3.2: Example section of BAM output view. Reads come from a ChIP-Seq experiment in which the transcription factor CEBPA was bound to the Gallus gallus genome.

Here, there are multiple overlapping reads which have been aligned between positions 950 to 1000.

3.2.2 Peak Calling

Now that all reads have been aligned to the genome, files have been created which describe the location of every read that resulted from the ChIP-Seq. Specifically, there are the following:

- aligned sequences from the input data, which was generated by ChIP-Seq experiments where no transcription factor was used. These simply represent a random sampling of DNA sequences and serve as our prior distribution of sequence reads, uninfluenced by transcription factor binding.

- aligned sequences from the experimental data, which was generated by ChIP-Seq experiments where a transcription factor was indeed used. These represent the DNA sequences where the transcription factor was found to be bound.

Peak calling was then performed using MultiGPS [44], which utilizes the genome reference information (as noted in Section 3.1) and the control data and experimental data described above. This allows us to identify the locations on the genomic sequence where there are experimental "peaks", i.e. where the experimental ChIP-Seq yielded many reads in the experimental dataset compared to the input dataset. This tells us where the transcription factor bound to the genome while eliminating pre-existing genomic sequence distribution biases (i.e. smaller peaks in the input data). The result is a BED file, which stores the coordinates of genomic regions. In this case, the results of MultiGPS show us the locations of peak numbers of reads, i.e. the positions at which transcription factors were determined to be bound. Specifically, for each peak, it provides the chromosome number and the start and end positions within the chromosome of where the peak was found, and the score of the peak (i.e., how prominent it is).

```
track name=multiGPS-CEBPA description=multiGPS events CEBPA
chr23      1464848      1464849      multiGPS      1000
chr7       11805936     11805937     multiGPS      1000
chr3       4463099      4463100      multiGPS      1000
chr5       55122002     55122003     multiGPS      1000
chr9       16179104     16179105     multiGPS      1000
chr12      20064861     20064862     multiGPS      1000
chr1       56657227     56657228     multiGPS      1000
chr3       2554446      2554447      multiGPS      1000
chr10      17524619     17524620     multiGPS      1000
```

Figure 3.3: Example section of BED output. Data represents the transcription factor CEBPA binding to the Gallus gallus genome.

Here, we see a few of the peaks detected by MultiGPS and their chromosomal positions, in the case of CEBPA binding to the Gallus gallus genome. Each of these specific peaks has a score of 1000, meaning these were all very strong peaks.

3.2.3 Genome Windows

The next step is to create windows of all of the genomic sequences for each genome we are working with. This will later allow us to create standardized data; we create genomic windows of 500 base pairs each, located every 50 base pairs along the genome. This means that they have substantial overlap, but also that we will have more robust training data later on. This was done using a Python script which simply generated 500 base-pair windows for each chromosome until the limit of the chromosome size was reached (information which is found in the genome reference information files found on the UCSC Genome Browser [34]).

```
chr1        0          500
chr1        50         550
chr1        100        600
chr1        150        650
chr1        200        700
chr1        250        750
chr1        300        800
chr1        350        850
chr1        400        900
chr1        450        950
```

Figure 3.4: Example section of genome windows. For each window, chromosomal position is stored (chromosome number, start position, end position). Sequence windows come from the Gallus gallus genome.

Next, we aim to remove the genomic windows that are not useful and/or relevant, keeping only valid ones for deep learning model training later on.

3.2.4 Blacklist Regions

First, we determine the "blacklist" regions in the genome. These are problematic regions where there is an "artifactual" signal, or an anomalous/artificially high number of mapped reads in the genome regardless of the experiment that was done. In other words, they are low-quality regions in which a detected peak may not truly represent a real transcription factor binding site, so we filter these out.

For the common experimental species (hg38 and mm10), we were able to find the blacklist regions from ENCODE, a repository of data which aims to understand the purpose of the vast non-gene component of the genome, which we now know plays a role in gene expression [19]. On the other hand, for the less common species (canFam4, galGal6, rn5, rheMac10, monDom5), we create a proxy for these blacklist files by using DomainFinder [43]. This uses the input experimental data to determine the locations where there are a statistically significantly high number of reads. We use $p = 0.01$ as the level of statistical significance to ensure that these blacklist regions are truly significant.

```
chrM        49          16750       REGION      1       +
chr3        2474425     2476080     REGION      1       +
chr14       14002603    14015036    REGION      1       +
chr20       1449409     1456898     REGION      1       +
chr14       14634562    14640271    REGION      1       +
chrW        1839577     1845058     REGION      1       +
chr16       1839853     1841620     REGION      1       +
chr2        52143032    52143738    REGION      1       +
chr8        9102185     9103189     REGION      1       +
chr14       14642882    14644894    REGION      1       +
```

Figure 3.5: Example section of DomainFinder output. Data comes from Gallus gallus ChIP-Seq.

This example shows a few of the artificial peaks which were detected from the input data. We then exclude these blacklist regions from our genomic windows by filtering out any genomic windows which overlap with blacklist regions. This allows us to consider only the genomic windows that do not have any artificially enriched signal. This is done using a Bash script which runs BEDtools, a software suite built to handle and process BED files [56]. This process can be visualized as follows:

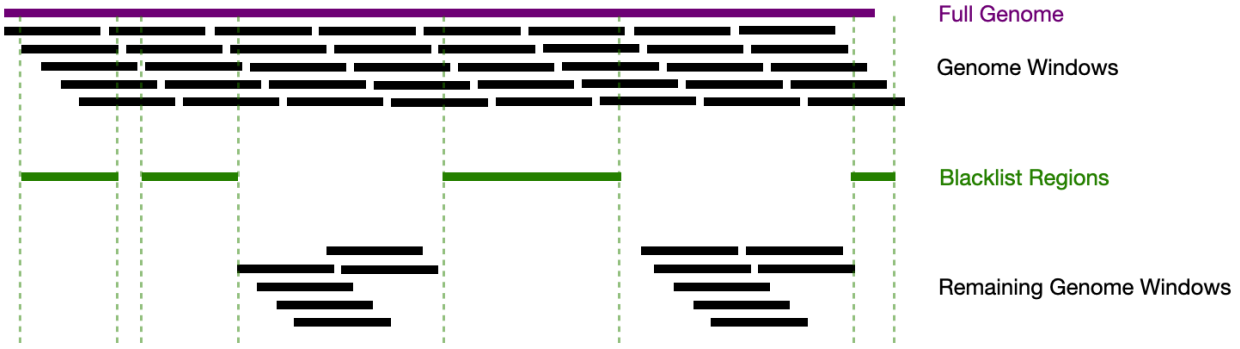


Figure 3.6: Conceptual illustration of filtering out all genomic windows which overlap with blacklist regions.

For Gallus gallus in particular for example, filtering out the blacklist regions yields 20931045 remaining windows out of a total of 20951605. The example output below shows that some windows were removed on chromosome 1 between positions 2550750 and 2553000.

```
chr1        2550500        2551000
chr1        2550550        2551050
chr1        2550600        2551100
chr1        2550650        2551150
chr1        2550700        2551200
chr1        2550750        2551250
chr1        2553000        2553500
chr1        2553050        2553550
chr1        2553100        2553600
chr1        2553150        2553650
```

Figure 3.7: Example section of output after removing blacklist regions. Data comes from the Gallus gallus genome.

3.2.5 Non-uniquely Mappable Regions

Next, we determine the non-uniquely mappable regions regions in the genome. These tend to be particularly repetitive or redundant regions in the genome, where it is difficult to correctly align sequencing reads. This leads to unreliability in the position of a read given by an aligner (such as Bowtie), and so we exclude them.

Once again, for the common experimental species (hg38, mm10), mappability tracks were readily available from the Umap project [32]. For the less common species (canFam4, galGal6, rn5, rheMac10, monDom5), we first tried to use the Umap software, but struggled to integrate it into the existing pipeline. Instead, we attempted to write our own version of the Umap software. To do this, we wrote a Python script to generate 36-base-pair-long genomic sequences using the reference genome for each species. Given these genomic sequences, we aimed to align them back to the genome, once again using Bowtie. Ideally, a given read maps to a unique position on the genome, in which case there is no ambiguity. However, in the case of unmappable regions, the issue is that these regions of the genome are highly repetitive and difficult to align reads to. In this case, there will be multiple possible alignments of a given read; these are the non-uniquely mappable reads. We do not want to include genomic windows which intersect with non-uniquely mappable regions, so we intersect these with our genomic windows, keeping only the windows that do not intersect non-uniquely mappable regions. (This process is exactly identical to excluding the

blacklist regions, as illustrated earlier.) However, this brute force method was too computationally intensive and took too long.

Finally, we used GenMap [53], an efficient algorithm to compute mappability of short genomic sequences. This allowed us to determine the uniquely mappable regions.

```
chr1        0          20
chr1        136        168
chr1        250        313
chr1        352        457
chr1        462        483
chr1        488        495
chr1        510        572
chr1        688        721
chr1        803        884
chr1        931        1010
```

Figure 3.8: Example section of output after generating mappability tracks. These represent the uniquely mappable regions of the genome (chromosome number and start and end positions). Data comes from the Gallus gallus genome.

From here, we generate a list of non-uniquely mappable regions. Note that these regions are exactly complementary to the uniquely mappable regions shown above.

```
chr1        20         45
chr1        45         60
chr1        60         106
chr1        106        112
chr1        112        125
chr1        125        136
chr1        168        250
chr1        313        337
chr1        337        341
chr1        341        347
```

Figure 3.9: Example section of output after generating non-uniquely mappable regions. Data comes from the Gallus gallus genome.

As with the blacklist regions, we then filter out any genomic windows which overlap with the non-uniquely mappable regions. Again, this is done using a Bash script which runs BEDtools. Finally, we have a list of genomic windows that do not overlap with any "problematic" (i.e. blacklist or non-uniquely mappable) regions of the genome. At this point, for Gallus gallus for example, we

have 17378385 remaining genomic windows, compared with the 20931045 remaining windows after removing only blacklist regions and the original total of 20951605.

Now, the data is ready to be transformed into a format which can be used for training, validation, and testing of deep learning models.

3.3 Creating Training, Validation, and Test Datasets

For every combination of species (genome) and transcription factor, we generate labelled data, describing whether a particular site is bound by a transcription factor. To do this, we simply consider our remaining genomic windows (after filtering out blacklist and non-uniquely mappable regions) and the peaks detected by MultiGPS. We then assign a label of 1 (i.e., "bound") to the windows which intersect any peak detected by MultiGPS, and a label of 0 (i.e., "unbound") to the remaining windows. This allows us to create a truly representative sample of bound sequences (as these bound sequences can contain an experimentally determined transcription factor binding region at any position within the window) and a sample as representative as possible of the unbound sequences (all of the remaining sequences in the genome). This is done using a Bash script.

```
chr1        56656100        56656600        1
chr1        56656150        56656650        1
chr1        56656200        56656700        1
chr1        56656250        56656750        1
chr1        56656300        56656800        0
chr1        56656350        56656850        0
chr1        56656400        56656900        0
chr1        56656450        56656950        0
chr1        56656500        56657000        0
chr1        56656550        56657050        0
chr1        56656600        56657100        0
chr1        56656650        56657150        0
chr1        56656700        56657200        0
chr1        56656750        56657250        1
chr1        56656800        56657300        1
chr1        56656850        56657350        1
```

Figure 3.10: Example section of output after labelling windows as bound or unbound. Data comes from the Gallus gallus genome and represents CEBPA binding events.

In the above example, we see that due to binding locations determined at positions 56656256 and 56657227 on chromosome 1, the overlapping windows have been labelled with 1, for "bound", while the others were labelled with 0, for "unbound". In this case (i.e. CEBPA binding to the Gallus gallus genome), we determined that there were 231599 bound windows out of the total of 17378385 windows: a relatively very small minority, or only about 1.3% of the genomic windows.

We now arbitrarily choose chromosome 1 to represent the validation dataset and chromosome 2 to represent the testing dataset for each genome, leaving the remaining chromosomes (chromosome 3 onwards) as training data.

We then split only the training data into two separate files, one containing bound examples and the other containing unbound examples, based on the assigned label. We also randomly shuffle both files so that our training data is in a randomized order; this will allow us to select more representative batches of data in every training step, as needed. This is done using a short Bash script which simply separates the data and writes it out to two different files.

At this point, we also create a smaller set of bound examples, based on the 5000 most prominent ChIP-Seq peaks. Although bound examples were already a very small subset of the total set of genome windows, all of the multi-species models take data from several different species; hence, it is helpful to use a smaller set of data (yet still sizable) in order to speed up model training while maintaining accuracy. However, it is important to choose this subset carefully. In this case, we select this smaller set by limiting to the set of windows bound to the 5000 strongest ChIP-Seq peaks. These sites are all certainly bound by the transcription factor and should therefore serve as a very reliable set of bound windows.

The next issue to tackle is the vast disparity between the number of bound windows and unbound windows. When training a deep neural network, we want the data provided in each batch to be balanced, i.e. the same number of bound and unbound examples. Therefore, we will keep the set of bound windows the same in each training epoch (since these only represent approximately 1% of the total windows), while the set of unbound windows changes in each epoch. In addition, we ensure that no unbound sites are repeated across different epochs. This allows us to

use a balanced amount of data in each training step but also ensure that the neural network sees as many different unbound examples as possible. According to these requirements, we then create a different file of unbound sites for each epoch. This is done using a Bash script which randomizes the unbound sites and extracts separate chunks of the data for each epoch.

```
chr3        26435500        26436000        0
chrZ        53878500        53879000        0
chr3        33594600        33595100        0
chr23       4358800         4359300         0
chr9        21997000        21997500        0
chrZ        52201300        52201800        0
chr8        26588850        26589350        0
chr5        43312450        43312950        0
chr15       12109350        12109850        0
chr10       6481700         6482200         0
```

Figure 3.11: Example section of one batch of unbound training data generated for Epoch 10 in the second run of model training. Data comes from the Gallus gallus genome and represents locations where CEBPA has not bound to the genome.

At this point, we have all of our training, validation, and test data for the part of the neural network which attempts to classify whether a given sequence is bound or unbound.

Next, we need to create training data for the part of the neural network which attempts to determine which species a given sequence has originated from. This process is similar to the process of generating data for the bound vs. unbound classifier, but aims to randomize across species rather than binding status.

First, we check how many bound sites were found in the previous part. This will tell us how large of a training dataset we would like to generate. Then for each species, we randomly select samples (regardless of whether they are bound or unbound); we select the same number of samples from each species so that once again, our training set is balanced. Once again, this is done using a Bash script similar to the one used previously.

3.4 Model Training

Finally, we have all of the training data needed in order to train our deep neural networks.

3.4.1 Encoding Input Data

Currently, our data is in the form of genomic coordinates, which in and of themselves are not extremely useful. Therefore, each time we select a batch of training data, we first convert these windows into the actual genomic sequence found at that location using the SeqDataLoader tool [59]. Note that we perform this conversion on the spot every time we choose a new batch of training data because the sequence data would require much more space to store and more time to read and process. This is unnecessarily computationally expensive and inefficient, so we avoid this by gathering the sequence data only when we need it. Moreover, this sequence data is not simply in the form of letters representing each nucleotide. Instead, deep neural networks require input data to be encoded numerically so that they can perform the mathematical matrix calculations needed to process data between neural network layers. Therefore, the sequence data is transformed into one-hot encoded vectors in which "A" is represented by the vector $[1, 0, 0, 0]$, "C" is represented by the vector $[0, 1, 0, 0]$, "G" is represented by the vector $[0, 0, 1, 0]$, and "T" is represented by the vector $[0, 0, 0, 1]$.

3.4.2 Conventional Model

We begin by training non-domain-adaptive models, which we refer to as "conventional" models. These are simply models that are trained only on data from one species and validated on data from the remaining species, and aim only to predict binding sites of a given transcription factor. We use these models later on to determine how accurately a model trained only on one species can predict transcription factor binding to the genome of a different species. They are also used as a benchmark with which to compare other models; the results on a given species of a model trained only on that species are the "gold standard" to which we compare the performance of other

models, which use training data from other species. We perform the same model training on every possible choice of genome and transcription factor.

3.4.2.1 Training

Each model is trained for 15 epochs, and model training is performed 5 times to ensure that results are consistent and reproducible.

The model architecture is as follows:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
seq (InputLayer)             (None, 500, 4)            0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 500, 240)          19440
_________________________________________________________________
activation_1 (Activation)    (None, 500, 240)          0
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 34, 240)           0
_________________________________________________________________
lstm_1 (LSTM)                (None, 32)                34944
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              33792
_________________________________________________________________
dropout_1 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               524800
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 513
_________________________________________________________________
activation_2 (Activation)    (None, 1)                 0
=================================================================
Total params: 613,489
Trainable params: 613,489
Non-trainable params: 0
_________________________________________________________________
```

Figure 3.12: Summary of conventional model architecture.

Here, we see the following structure:

- Our input is a series of genomic sequences. They are 500 base pairs long (recall that each of

our genomic windows was, in fact, 500 base pairs long), and each nucleotide is represented by a vector of length 4.

- Input sequences go through a Convolutional layer. Here, many convolutional filters are applied, creating a larger feature map. This then goes through a ReLU Activation function; these functions are generally placed after convolutional layers. They serve to help decide if a particular neuron in the neural network should fire or not; the neuron is "activated" when that particular convolutional filter pattern is detected.

- Next is a pooling layer. These are used to consolidate and reduce the size of the data, and they also help us control overfitting.

- We then include an LSTM layer, or Long Short-Term Memory. These are designed for use in sequence prediction problems, as they specifically aim to account for sequential dependencies (i.e. "when nucleotide T follows the sequence AGGCA as opposed to nucleotide C, the transcription factor is more likely to bind.")

- We then have a series of Dense layers, which generally serve to synthesize all of the neural network output so far and change the dimensionality of the data so that we may more easily define the relationship between the input and output data.

- Note that in between the Dense layers, there is a dropout function. These serve to prevent overfitting by randomly leaving out inputs from the previous layer. This helps the model to generalize more effectively.

- Finally, there is another activation function which simply transforms the output into a probability; in this case, the probability that the original sequence represented a bound site.

In each epoch, the model is presented with every bound example and every unbound example as well, in small batches of 400 examples at a time. Each batch of data is processed through the neural network, resulting in some output which represents the calculated probabilities that each of the sequences are bound. This is then compared with the true bound/unbound labels and the loss

value is calculated (in this case, binary crossentropy loss was used). The loss is backpropagated through the network, prompting it to adjust its assigned weights, and it starts again on a new batch of data. After each epoch, the best model thus far is saved.

3.4.3 Multi-Species Model

We build another non-domain-adaptive model, dubbed a "multi-species model", as well. These are models that are trained on data from all but one species and validated on data from the final remaining species, and still aim only to predict binding sites of a given transcription factor. We use these models later on to determine how accurately a model trained on several species can predict transcription factor binding to the genome of a different species. We perform the same model training on every possible choice of genome and transcription factor.

3.4.3.1 Training

Each model is trained for 15 epochs, and model training is performed 5 times to ensure that results are consistent and reproducible. The model architecture is identical to that of the conventional model trained on one species; the only difference comes in the sets of data provided to the model on which to train.

3.4.4 Domain-Adaptive Model

We then train domain-adaptive models. These are two-pronged models: one branch which, like the conventional model, attempts to predict transcription factor binding to a given sequence (termed the "discriminator"), and another branch which attempts to classify the given sequence as having come from a particular species (termed the "classifier").

In each run of domain-adaptive model training, the binding classifier is trained on all but one species, which is left out as the "target" for prediction. The species classifier is exposed to data from all species, attempting to learn the features which differentiate between different species. We use these models later on to determine how accurately a model trained on multiple species can

predict transcription factor binding to the genome of a different species. We perform the same model training on every possible choice of "target" genome and transcription factor.

3.4.4.1 Training

Each model is trained for 15 epochs, and model training is performed 5 times to ensure that results are consistent and reproducible.

The model architecture is as follows:

Layer (type)	Output Shape	Param #	Connected to
seq (InputLayer)	(None,500,4)	0	
conv1d (Conv1D)	(None,500,240)	19440	seq[0][0]
activation_1 (Activation)	(None,500,240)	0	conv1d[0][0]
max_pooling1d (MaxPooling)	(None,34,240)	0	activation_1[0][0]
reshape (Reshape)	(None,8160)	0	max_pooling1d[0][0]
lstm (LSTM)	(None,32)	34944	max_pooling1d[0][0]
gradient_reversal (Gradient	(None,8160)	0	reshape[0][0]
dense_1 (Dense)	(None,1024)	33792	lstm[0][0]
dense_2 (Dense)	(None,1024)	8356864	gradient_reversal[0][0]
activation_2 (Activation)	(None,1024)	0	dense_1[0][0]
activation_3 (Activation)	(None,1024)	0	dense_2[0][0]
dense_3 (Dense)	(None,512)	524800	activation_2[0][0]
dense_4 (Dense)	(None,512)	524800	activation_3[0][0]
classifier (Dense)	(None,1)	513	dense_3[0][0]
discriminator (Dense)	(None,7)	3591	dense_4[0][0]

Total params: 9,498,744
Trainable params: 9,498,744
Non-trainable params: 0

Figure 3.13: Summary of domain-adaptive model architecture.

This is visualized as follows:

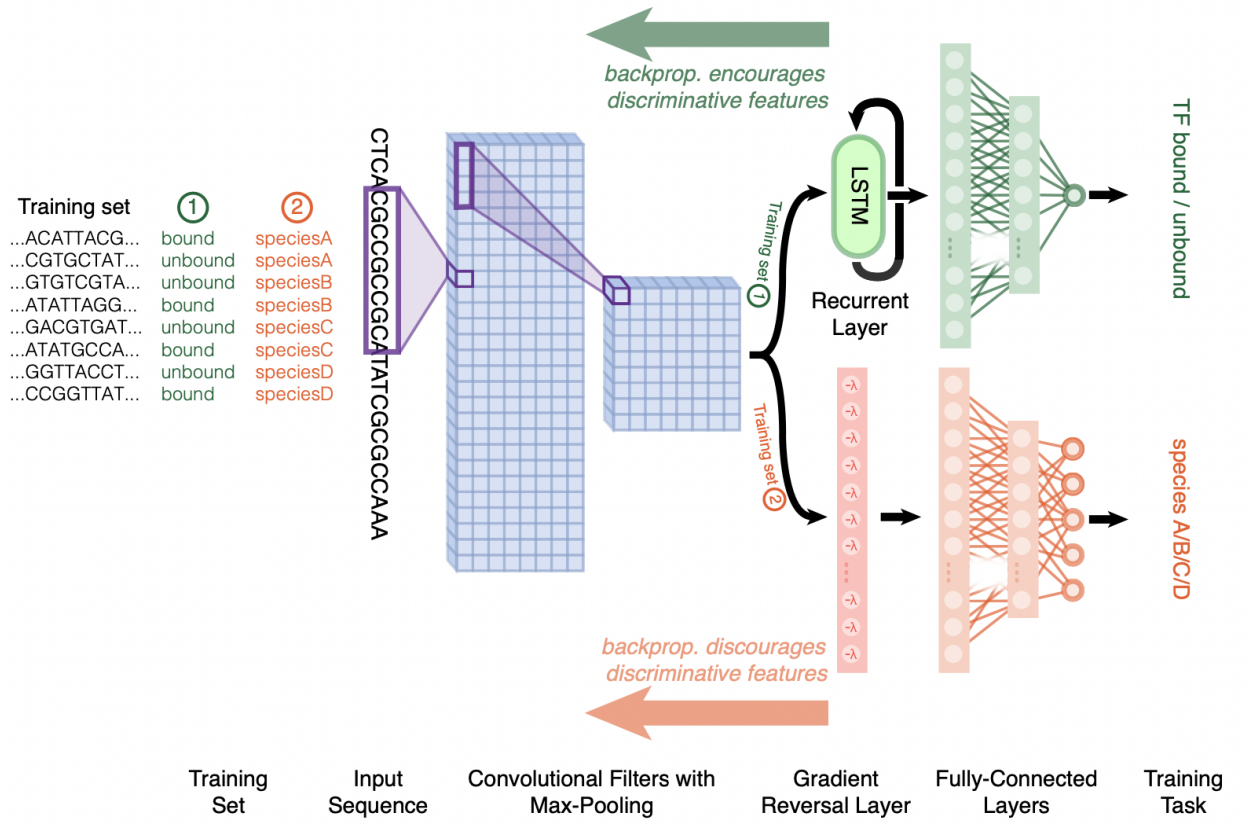


Figure 3.14: View of domain-adaptive model architecture

This domain-adaptive model summary is more difficult to interpret, but it is overall quite similar to the conventional model. Both branches of the model follow a similar structure except for one key difference. Just like the conventional model, the binding classifier has an LSTM neural network layer. The species classifier, in place of the LSTM layer, includes a "Gradient Reversal" layer. Without this layer, the species classifier would learn the features which distinguish between the genomes of different species, and this learning would be backpropagated through the network. Instead, with the Gradient Reversal layer, the model is *discouraged* from learning these species-specific features. The idea is that this will help the model to generalize learning genomic features across multiple species, rather than learning features that are unique to particular species.

Also note that the species classifier uses categorical crossentropy loss rather than binary crossentropy because its output is a categorical variable (i.e. one of several different categories) rather than binary (i.e. one of two possibilities).

3.4.5 Binary Adversarial Model

We build another domain-adaptive model, referred to as a "binary adversarial model", as well. These, too, are two-pronged models that are trained on data from all species. One branch, like the multi-species model, attempts to predict transcription factor binding to a given sequence (termed the "discriminator"), and uses data from all but one species (later used as the test species). The other branch (termed the "classifier"), instead of classifying which species the given sequence came from, simply classifies whether the sequence came from the test species or not. We perform the same model training on every possible choice of genome and transcription factor.

3.4.5.1 Training

Each model is trained for 15 epochs, and model training is performed 5 times to ensure that results are consistent and reproducible. The model architecture is almost identical to that of the domain-adaptive model; the only difference comes in the classification task, which is now binary classification as opposed to multi-class classification.

3.5 Analysis and Visualization

After all of the models described above have been trained, we test them on the datasets that have been left out for testing (data from chromosome 2 on every genome) and use a Jupyter notebook to visualize performance.

We first examine the auPRC, or area under the precision-recall curve, that was calculated during training to determine how accurately conventional models were able to predict transcription factor binding as they learned. Since this metric is commonly used when the data is imbalanced, it is appropriate in this setting, in which we have far fewer bound examples than unbound. We then compare auPRC values across different models, trained on different species.

Next, we examine both bound and unbound sites from human chromosome 2, our testing dataset of interest. We compare how accurately conventional models trained on different species

were able to predict binding on these sites.

Finally, we also compare how the domain-adaptive model targeted towards the human genome (i.e. trained on all other species) performed compared the the conventional model trained on the human genome. Ultimately, this is what addresses our overarching research question, which was aiming to apply a neural network trained on multiple different species to the human genome.

Chapter 4

RESULTS

As previously discussed, in this work, I trained deep neural networks to learn the binding sites of four different transcription factors (CEBPA, FoxA1, HNF4a, HNF6) in seven different species. (Again, note that seven-species data exists only for CEBPA; for the other three transcription factors, data exists for five of the seven species.) For all genome and transcription factor combinations, I built and trained the four different types of neural networks described in the previous section. I then applied these models to predict the regulatory regions in each genome; the results of this evaluation are as follows.

4.1 Validation Analysis

We first examine the conventionally trained models: those which have been trained on the binding data from only one species. These models were trained on each combination of species and transcription factor separately and during training were validated on all species using the held-out validation set (chromosome 1).

Using auPRC as the performance metric (the most useful metric in an imbalanced data setting such as this), validation performance was monitored after each training epoch; results are as follows:

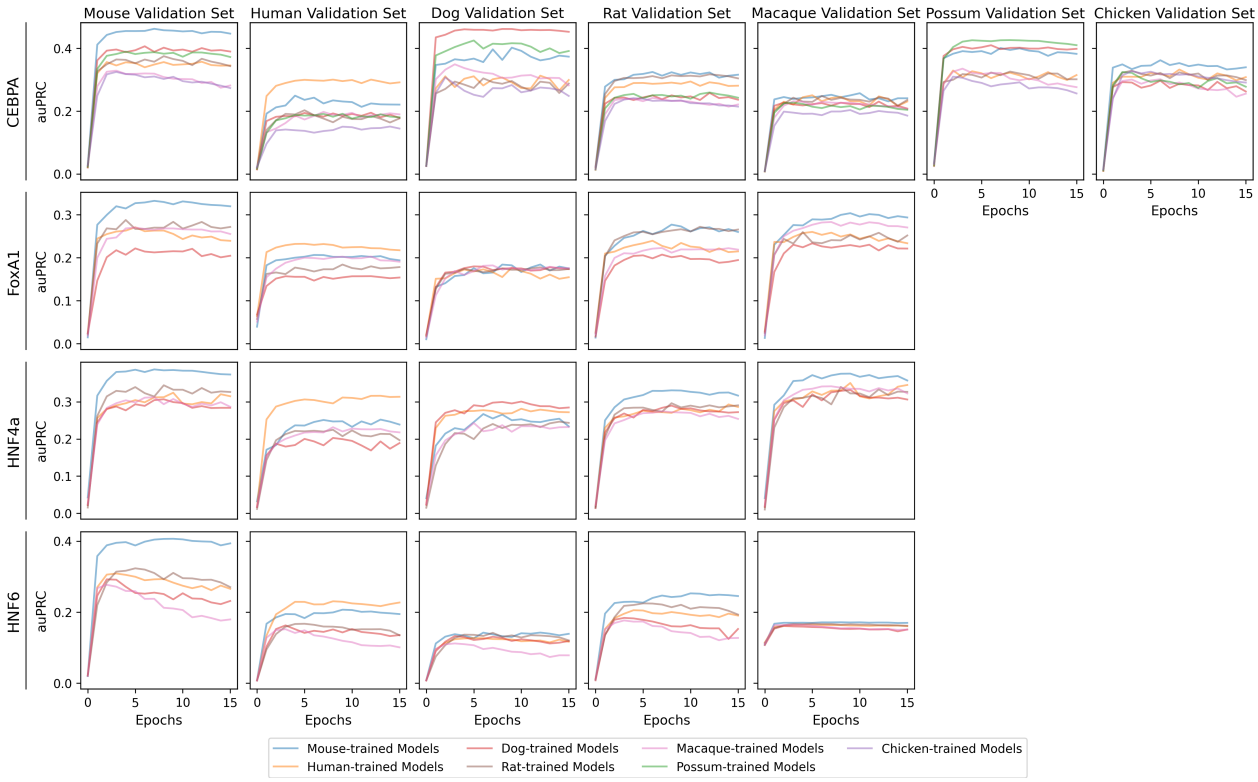


Figure 4.1: Model performance during training, evaluated on validation data.

Note that as expected, for each species, the source species model usually exhibits better accuracy than models trained on the other species. Observe also that in some cases, while validation accuracy may continue to improve on the source species, this may not be the case for the other species. For example, when validating HNF6 models on the mouse (mm10) dataset, the mouse-trained model continues to improve while the others begin to decline. However, in general, this is not the case, and validation accuracy follows the same trend across all species, albeit with different levels of variability in different species and different transcription factors. During training, the version of each model with the highest source-species auPRC was saved for further analysis.

4.2 Conventional Model Test Performance

We next evaluate the performance of different models on human test data, as this is in practice the most valuable testing setting. In particular, we evaluate the model trained on human training data (used as the gold standard with which to compare other models), models trained on each one of the other species, the multi-species model, the domain-adaptive model, and the binary adversarial model. Each of these models is run on the held out test set (chromosome 2) of the human genome, which is a dataset it has not seen before during training.

First, we examine the auPRC of the single-species models, trained on each species.

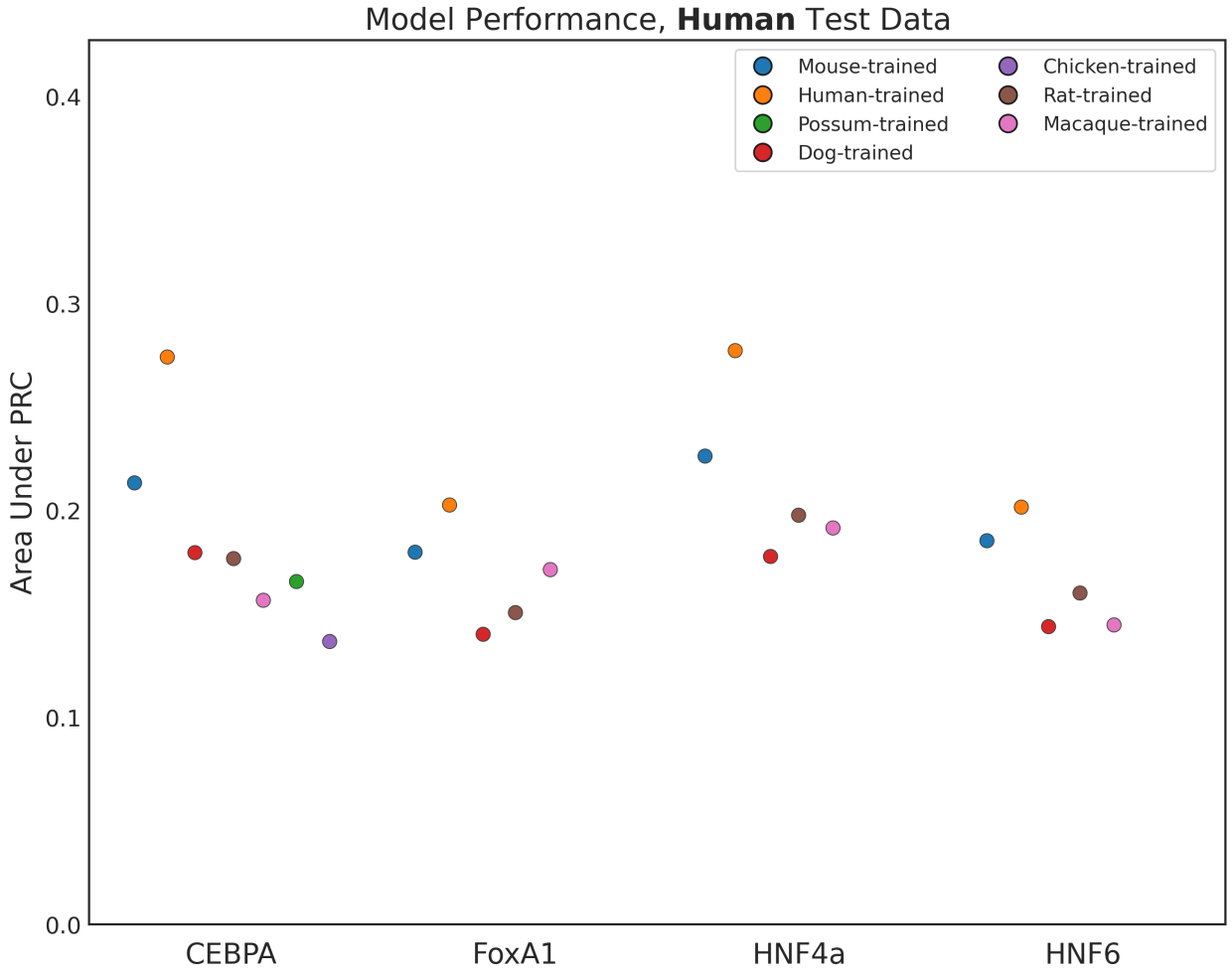


Figure 4.2: Conventional model results for each species, evaluated on test data.

As before with the validation auPRC, we see that there is a performance gap across transcription factors, i.e. for every transcription factor, the human-trained models do outperform the models trained on other species when evaluated on a test set from the human genome. Although the width of this gap varies in different transcription factors, it always remains present.

We therefore examine the source of this performance gap. Particularly, we plot the human-trained model prediction compared to the mouse-trained model prediction (consistently the next best performer) separately on bound and unbound sites in the human genome test dataset. For clarity, only 25% of bound sites and 5% of unbound sites, randomly chosen, are plotted.

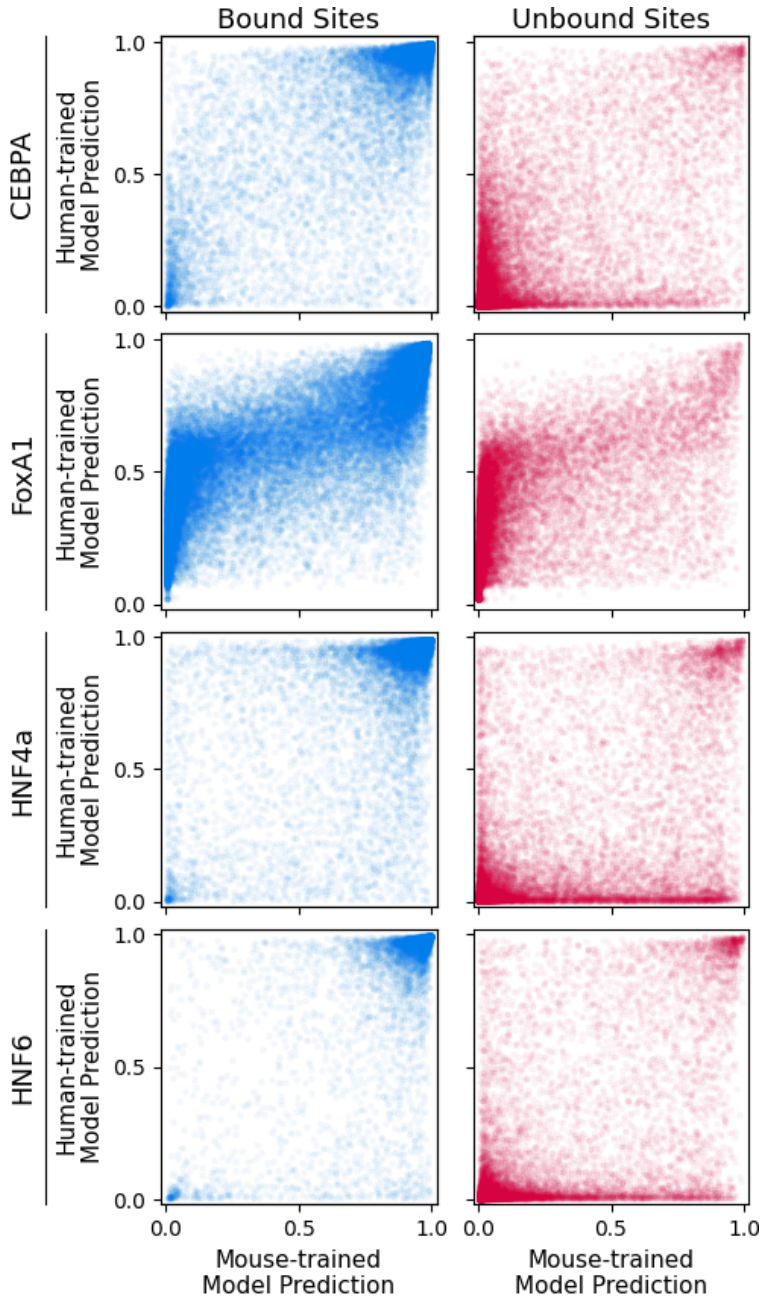


Figure 4.3: Binding prediction by human-trained and mouse-trained models on bound and unbound sites from human genome test data

The goal here is to visualize the difference between human-trained model predictions (again, used as the gold standard) and, in this case, the mouse-trained model predictions. (A similar analysis may be carried out for any other species.) Ideally if there were little to no performance gap, these models' predictions would match well.

For most transcription factors (i.e. CEBPA, HNF4a, HNF6) the bound site prediction was fairly accurate. We see that the points on the scatter plot (representing genomic windows) are largely clustered in the upper right corner (meaning that both models correctly predicted binding), with a few clustered in the bottom left (both models incorrectly predicted lack of binding). These sites do not contribute substantially to the performance gap, as the human-trained model predictions and mouse-trained model predictions largely match.

On the corresponding plots for unbound site prediction, we see that there is a large cluster in the bottom left corner, indicating that both models correctly predicted lack of binding. Again, these sites do not contribute to the performance gap. However, we also see that there is a noticeable cluster along the x-axis. This means that for some subset of unbound sites, the human-trained model correctly predicted a lack of binding, while the mouse-trained model predicted that several of these sites could be bound. These sites display the cross-species performance gap; the mouse-trained model was not always able to correctly predict these sites as unbound.

Meanwhile, note that when predicting on FoxA1 binding sites, there were a substantial number of false negatives from both models (as seen on the plot of bound sites), though this does not contribute to the performance gap; simply a lower overall accuracy. On the plot of unbound sites, in this case, we actually see a cluster of sites in which the mouse-trained model correctly predicted a lack of binding and the human-trained model predicted a possibility of binding (though these scores were still less than 0.5).

Each of these clusters along the axes, particularly along the x-axis, show the origin of the cross-species performance gap. They represent sites where one model was able to correctly predict binding or lack thereof, while the other model was not. Note that in this case the performance gap is primarily due to mis-prediction on unbound genomic windows.

4.3 Multi-Species Model Test Performance

Next, we add to this view the performance of the multi-species model.

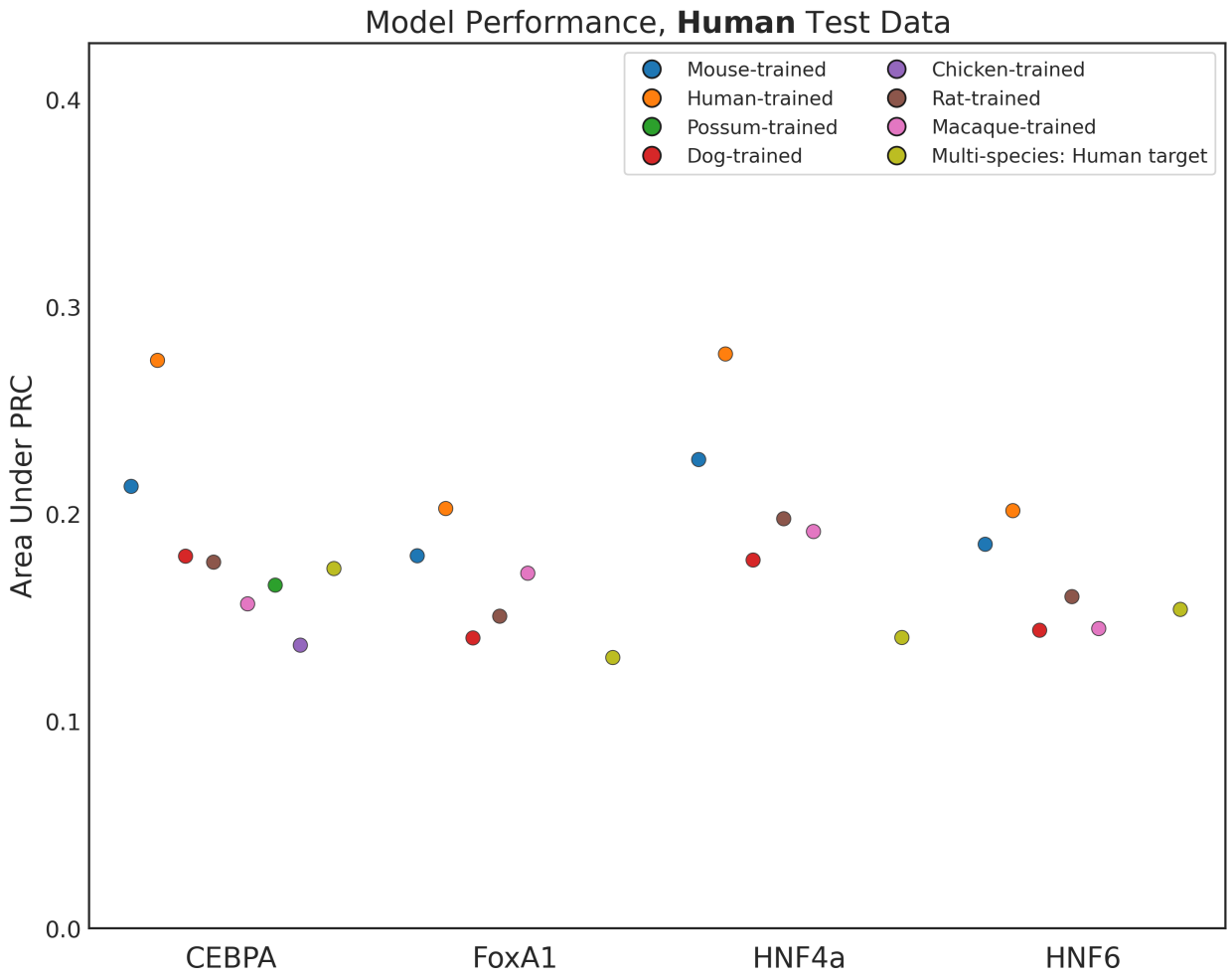


Figure 4.4: Conventional model results for each species and multi-species model results, evaluated on human genome test data.

As one might expect, this model, while still comparably accurate, is generally less effective than models trained on individual species, as it is likely learning the specific features of each species, negatively impacting its prediction on the human genome test set. Some of these species, like mouse and macaque, share many of their genomic features with humans and serve as fairly reasonable proxies for studying humans. However, others of these species, like opossum and chicken, share comparatively few of their genomic features with humans and serve as poor predictors of binding in humans. Note that for CEBPA, for which we had data from these additional species, the multi-species model outperformed these more distantly related species. However, the multi-species model, overall, may struggle to perform at a comparable level to single-species models

trained on classical model species. Again, the width of this performance gap varies across transcription factors, but it is always present and always larger than the performance gap between individual species.

As before, we examine the source of this performance gap as well. Particularly, we plot the human-trained model prediction compared to the multi-species model prediction separately on bound and unbound sites in the human genome test dataset. And again, only 25% of bound sites and 5% of unbound sites are plotted.

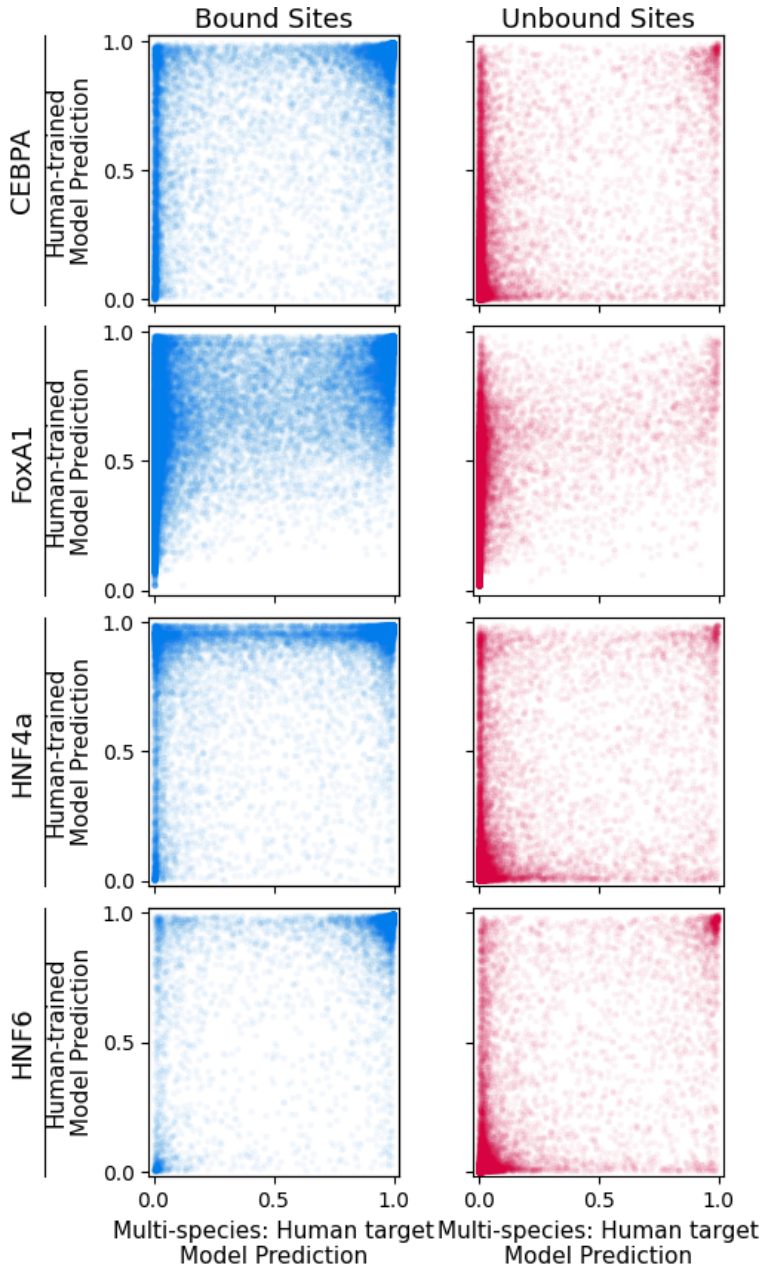


Figure 4.5: Binding prediction by human-trained and multi-species models on bound and unbound sites from human genome test data

First, examine the prediction on bound sites. Again, there is a large cluster in the upper right corner, indicating that both models correctly predicted binding. However, there is also now a cluster along the y-axis; these are sites that the multi-species model incorrectly predicted as unbound, regardless of the human-trained model prediction. There is also a cluster along the top of the plot; these are sites that the human-trained model correctly predicted as bound, while the multi-species

model was inconsistent, contributing to the performance gap.

Next, examine the prediction on unbound sites. There is, as expected, a cluster in the bottom left corner: sites that both models correctly predicted as unbound, which do not contribute to the performance gap. However, there is also again a cluster along the y-axis; these are sites that the multi-species model correctly predicted as unbound, while the human-trained model was inconsistent. These sites actually theoretically reduce the performance gap while contributing to a lower overall auPRC.

In contrast to the mouse-trained and human-trained comparison, this time, the plots for FoxA1 largely resemble those of the other transcription factors, though there seems to be more variability and as we can see from Figure 4.4, overall less accuracy.

Note that there is a pattern here in which predictions cluster along the y-axes of these plots. This indicates that the multi-species tends to over-predict a lack of binding compared to the human-trained model. Meanwhile, the clusters along the x-axis of the unbound site plots have disappeared, indicating that the multi-species model has corrected the false positives seen in the mouse-trained model. Also note that now, in contrast to the single-species model performance gap, the multi-species model performance gap is primarily due to mis-prediction on bound sites (i.e. false negatives) rather than on unbound sites (i.e. false positives).

4.4 Domain-Adaptive and Binary Adversarial Model Test Performance

Finally, we compare these models to the domain-adaptive and binary adversarial models. We first note that hyperparameter tuning, i.e. model optimization, has not yet been performed and therefore the model architectures used here may not be ideal and there is certainly potential for improvement. The immediate next step would be to perform a hyperparameter sweep, i.e. testing several different values of each model input parameter in order to optimize model performance.

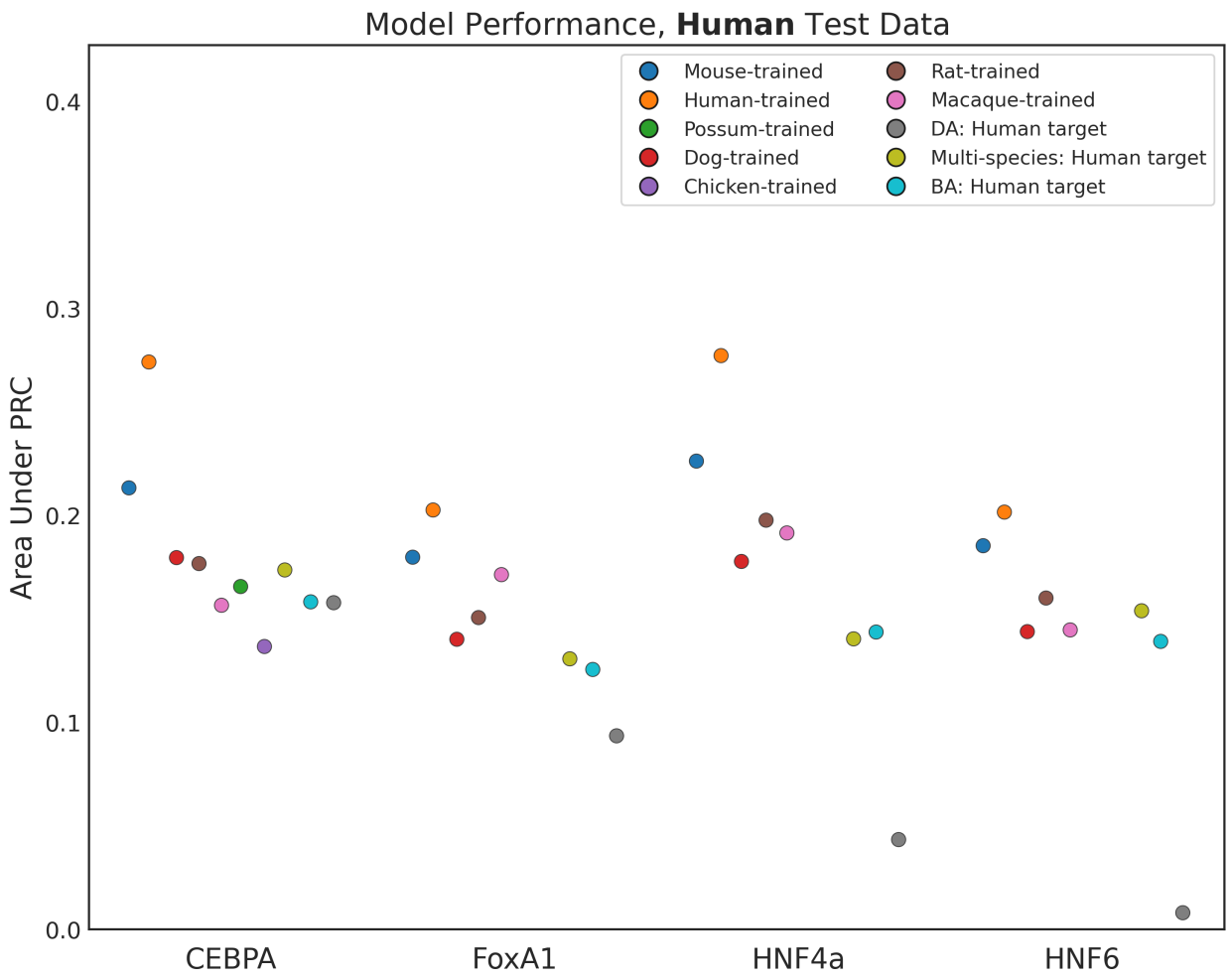


Figure 4.6: Conventional model results for each species, multi-species model results, domain-adaptive model results, and binary adversarial model results, evaluated on human genome test data.

Here, the hope was that the domain-adaptive and/or binary adversarial models would, to some extent, bridge the cross-species prediction performance gap. However, in this case, we notice that the domain-adaptive models perform poorly compared to the other model types, while the binary adversarial model performance is approximately on par with the multi-species model. Only for CEBPA is the domain-adaptive model also comparable to the binary adversarial model and the multi-species model. This is surprising, but once again we look to the source of the performance gap.

First, we plot the human-trained model prediction compared to the domain-adaptive model

prediction separately on bound and unbound sites in the human genome test dataset. And again, only 25% of bound sites and 5% of unbound sites are plotted.

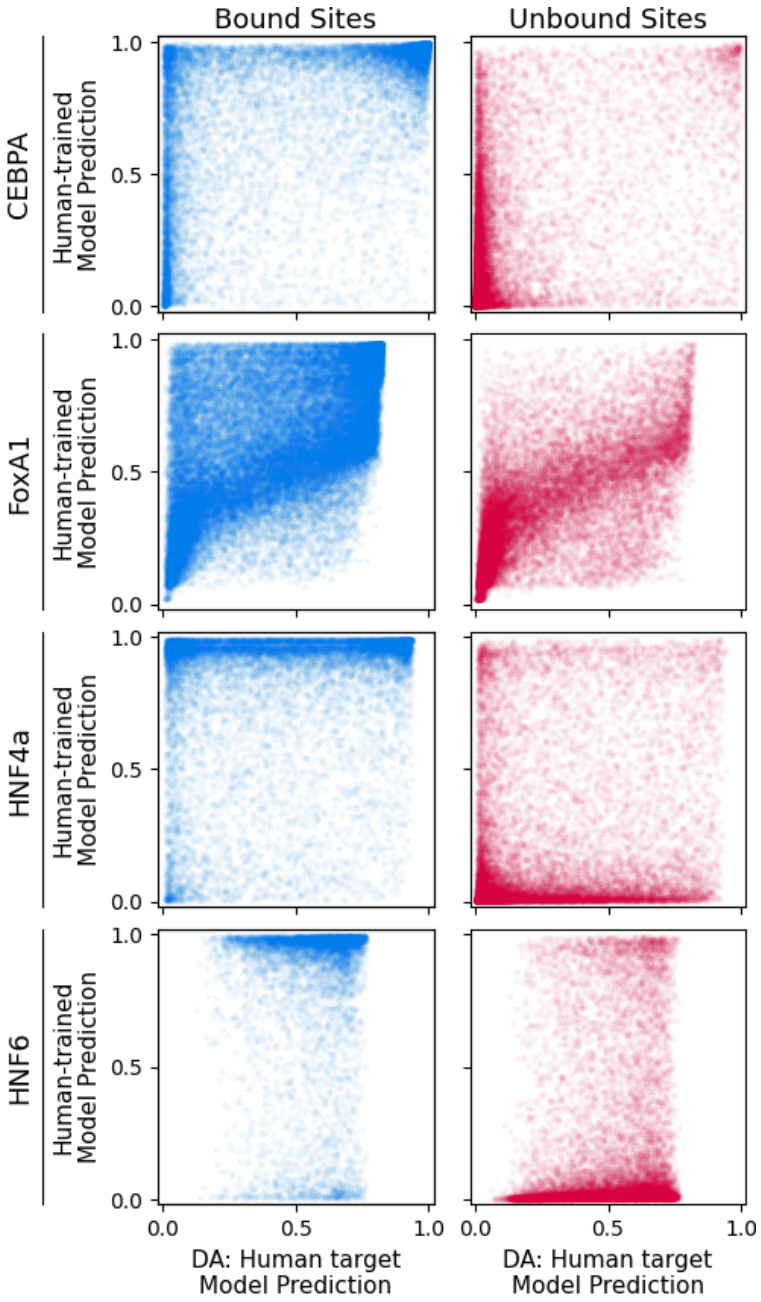


Figure 4.7: Binding prediction by human-trained and domain-adaptive models on bound and unbound sites from human genome test data

Here, we see surprising results almost across the board. The results of domain-adaptive model training on CEBPA binding sites look quite similar to the results from the multi-species model,

confirming that performance was similar in this case.

However, for the other transcription factors, the domain-adaptive model rarely - if ever - predicts definitive binding. This is very surprising, as the final layer in the branch of the neural network focused on binding prediction is a sigmoid activation function, which tends to map input values to output values close to either 0 or 1 rather than to mid-range output values. However, in the case of HNF4a, FoxA1, and especially HNF6 prediction, we see precisely the opposite outcome. This may suggest that the model struggled to detect sufficient commonality between the binding sites of all five species used for these transcription factors. This may also indicate that further model optimization and hyperparameter tuning is necessary. A third possibility is that data sampling methods need modification. As mentioned previously, when data is highly imbalanced, deep learning models tend to over-predict towards the larger class; in this case, unbound samples. This could explain why the domain-adaptive model often fails to predict high binding scores, and could indicate a need for additional methods to combat class imbalance.

In the case of FoxA1, we note that for any given bound genomic window, the binding score given by the domain-adaptive model is almost always lower than or equal to the binding score given by the human-trained model, indicating that the domain-adaptive model is not able to predict binding with an appropriate level of certainty. Meanwhile, for unbound genomic windows, the domain adaptive model's prediction is actually often comparable to the human-trained model's. Therefore in this case, we conclude that the performance gap is usually due to mis-prediction of bound sites.

We also note that for HNF4a and HNF6, there are very prominent clusters along the top axis in bound site prediction and the bottom axis in unbound site prediction. This shows that although the human-trained model accurately predicted binding or lack thereof in these cases, the domain-adaptive model struggled.

In fact, in the case of HNF6, note that the domain-adaptive model rarely - if ever - predicted that sites were definitively bound or unbound; rather, it almost always gave mid-level scores, resulting in the band we see in the above plot.

Finally, we plot the human-trained model prediction compared to the binary adversarial model prediction separately on bound and unbound sites in the human genome test dataset. As usual, only 25% of bound sites and 5% of unbound sites are plotted.

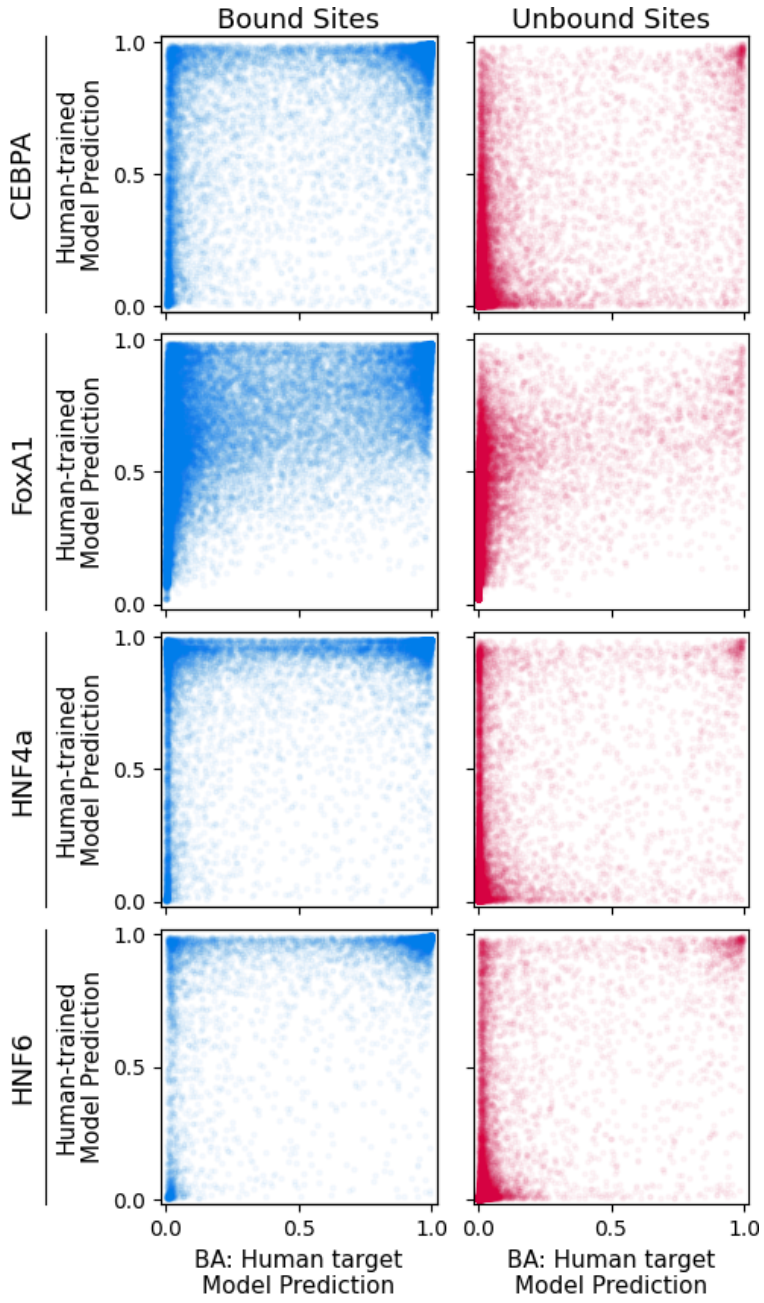


Figure 4.8: Binding prediction by human-trained and binary adversarial models on bound and unbound sites from human genome test data

These plots look almost identical to the comparison between the multi-species model and the

human-trained model, confirming the earlier result that the binary adversarial model and the multi-species model resulted in very similar auPRC values.

As with the multi-species comparison, there is a large cluster in the upper right corner when looking at the bound windows, indicating that both models correctly predicted binding. And again, there is a cluster along the y-axis; these are sites that the binary adversarial model incorrectly predicted as unbound, regardless of the human-trained model prediction. There is also a cluster along the top of the plot; these are sites that the human-trained model correctly predicted as bound, while the binary adversarial model was inconsistent; these windows contribute to the performance gap.

Next, when looking at prediction on unbound sites, there is a cluster in the bottom left corner: sites that both models correctly predicted as unbound, which do not contribute to the performance gap. Once again, there is also a cluster along the y-axis; these are sites that the binary adversarial model correctly predicted as unbound, while the human-trained model was inconsistent. These sites should reduce the performance gap, although they contribute to a lower overall auPRC.

Like with the multi-species and domain-adaptive models, we conclude from this that the binary adversarial model performance gap is primarily due to mis-prediction on bound sites (i.e. false negatives) rather than on unbound sites (i.e. false positives). We also conclude that again, given that the clusters along the x-axis of the unbound site plots have disappeared, it seems that the binary adversarial model has largely corrected the false positives seen in the mouse-trained model.

These results are all consistent with how auPRC is measured. Precision and recall, the two metrics that determine auPRC, do not take into account the proportion of “true negatives”; therefore, they are useful in settings like this where there is a vast majority of negative (unbound) samples and small minority of positive (bound) samples. Since the small positive class is of greater interest, the auPRC should be largely determined by the model’s handling of the positive samples: the better the performance on this subset of data, the higher the auPRC.

4.5 Brief Comparison to Mouse-trained/tested Models

Here, we offer a brief comparison of the human-target model performance compared with the mouse-target model performance. To do this, we recreate Figure 4.6: this time, again, the conventional models for each species are used and tested against the mouse genome. However, the multi-species model, domain-adaptive model, and binary adversarial model depicted are those which were trained using the mouse genome as the target genome, i.e. they were trained on all species except mouse.

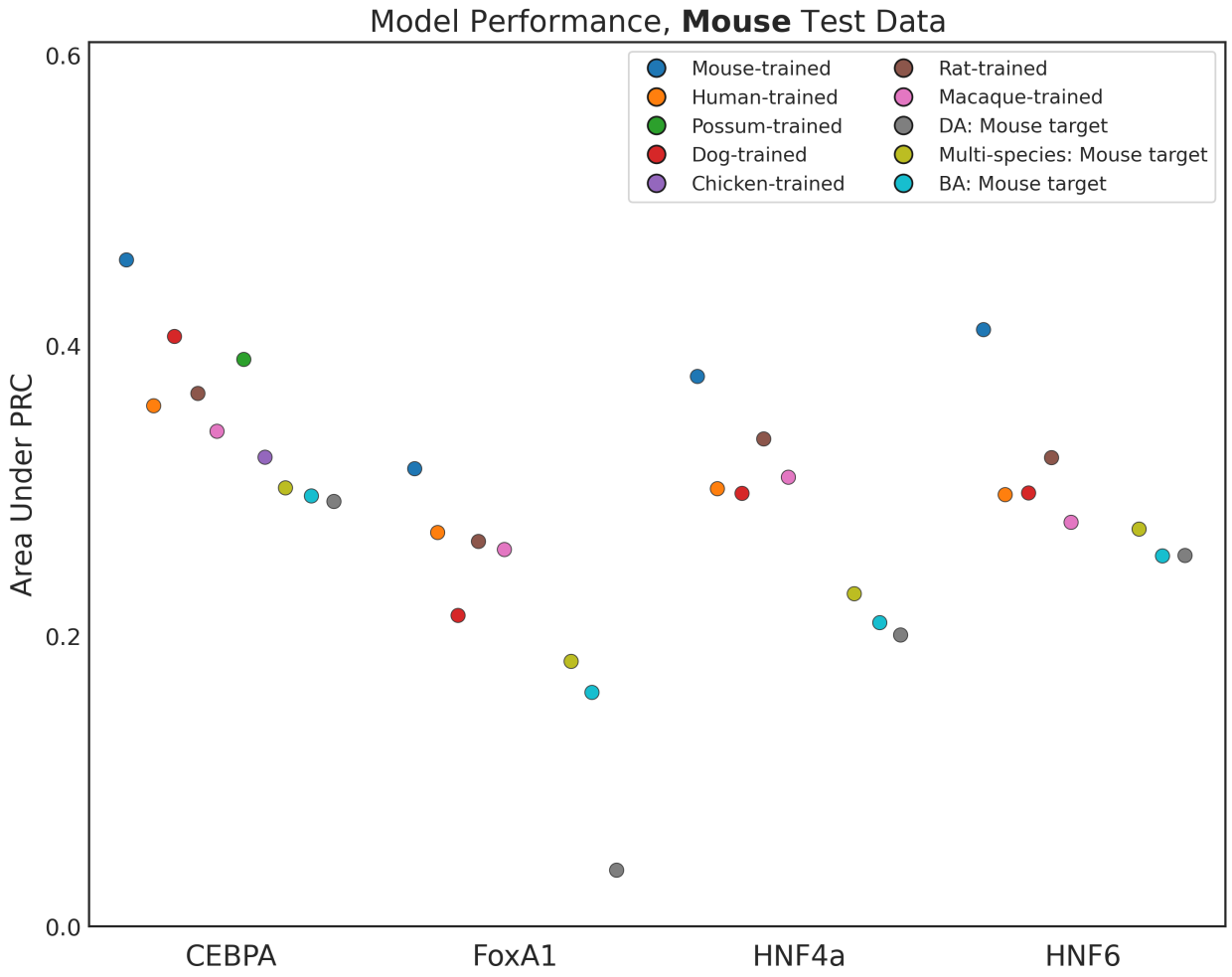


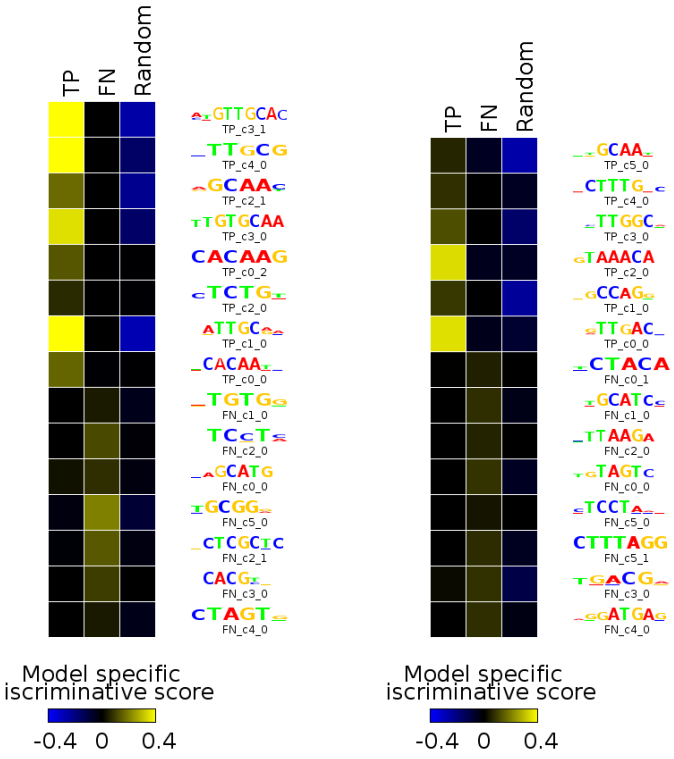
Figure 4.9: Conventional model results for each species, multi-species model results, domain-adaptive model results, and binary adversarial model results, evaluated on mouse genome test data.

Observe that there is one key difference between this and the human-target equivalent. This time, for CEBPA, HNF4a, and HNF6, the performance of the domain-adaptive model is quite comparable to that of the multi-species and binary adversarial models. All three of these models exhibit very similar auPRC scores.

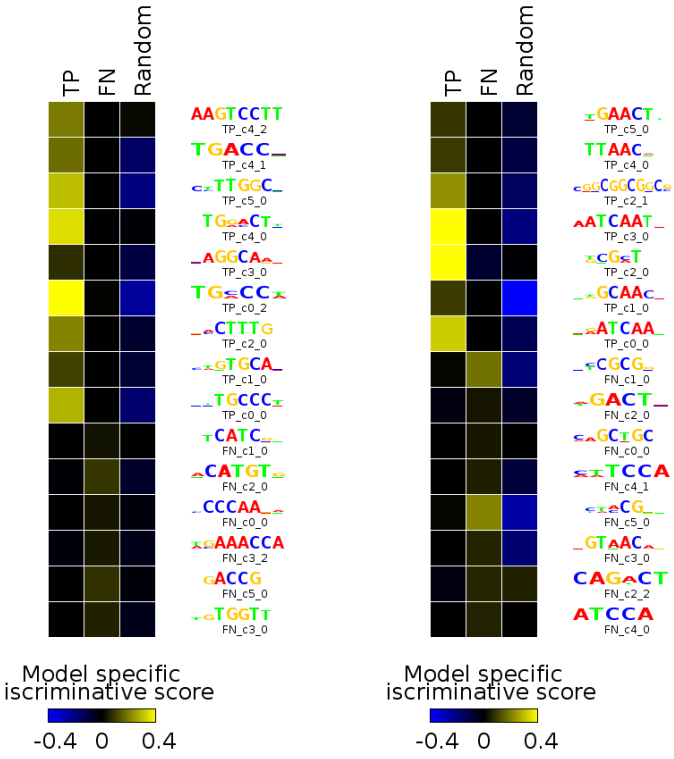
Again, recall that there is quite a wide variety of species used in training, so just as with the human-target models, it seems reasonable that the multi-species model, domain-adaptive model, and binary adversarial model may struggle to compare to the mouse-trained or human-trained single species models. However, we also note that the mouse genome has fewer species-specific transposable elements than the human genome. Alu elements, for example, a major source of the performance gap in human genome binding prediction, are specific to primates and are not found in the mouse genome.

4.6 Discriminative Analysis

As a final step of analysis, we run SeqUnwinder [31], a tool used to determine the discriminative features between two sets of genomic sequences. This extracts motifs which can discriminate between the two sets of sequences. In this case, we use it to compare (a) the set of bound sequences correctly predicted as bound by both the single-species human-trained model and the multi-species model and (b) the set of bound sequences correctly predicted as bound by the single-species human-trained model but incorrectly predicted as unbound by the multi-species model. That is, these are all sites correctly predicted by the single-species human-trained model but for the multi-species model they are true positives (TP) and false negatives (FN), respectively. We hope to uncover motifs associated with both of these categories to better understand the cross-species differences in transcription factor binding. Results are as follows for each transcription factor:



(a) CEBPA (b) FoxA1



(c) HNF4a (d) HNF6

Figure 4.10: Results of discriminative analysis

Digging more deeply into these results, we note that SeqUnwinder was able to detect motifs associated with TP (true positive; both models predicted correctly). In general, these are very strong associations and turn out to be the expected cognate motifs of the particular TF, indicating that these are the most common and conserved binding motifs. This is a reasonable result which serves as confirmation of effective model training.

SeqUnwinder was also able to detect motifs predictive of FN (false negatives; the single-species human-trained model predicted correctly while the multi-species model did not). These are much weaker associations, and are more difficult to interpret; these motifs do not match known motifs for the relevant TFs or others. However, we note that some features that appear weakly predictive of false negatives are shared across datasets, possibly indicating that there is a human- or primate-specific genomic feature that is predictive of binding but is not captured by the multi-species model.

Chapter 5

DISCUSSION

In this section, we discuss the above results in more depth and detail, and also discuss some limitations of the approaches used.

5.1 Evaluation Method

As discussed previously, when training machine learning models, there are a number of crucial choices to be made with respect to data sampling, model architecture, and learning methods. In particular, in this machine learning task there is a vast imbalance between the number of "positive" (bound) samples and the number of "negative" (unbound samples), which we have mentioned can pose challenges when attempting to train a neural network. Specifically in this case, only 1-1.5% of genomic sequence windows are bound for a given transcription factor and genome. This leads to a much larger set of negative samples compared to positive samples. Given all of this data at once, a machine learning model is inclined to predict that samples will be unbound, as the prior distribution indicates that there is a 99% chance this is the case. As has been described in previous

chapters, to combat this, we chose to maintain the same set of bound windows in each training epoch, while randomly under-sampling (without repetition) from the set of unbound windows. This allowed us to keep a balanced training set in each epoch while also maximizing the number of bound and unbound training examples.

However, when evaluating machine learning models' performance, it is also important to choose wisely in terms of which metrics are truly informative and offer representative and useful feedback on the quality of the model. Most metrics are based on a few different values, which are nicely summarized in the confusion matrix:

		Prediction outcome	
		Pos	Neg
Actual Value	Pos	True Positive	False Negative
	Neg	False Positive	True Negative

Table 5.1: Confusion Matrix

The confusion matrix reports the breakdown of prediction outcomes, i.e. four values: the number of positive samples correctly classified as positive (true positive; TP), the number of positive samples incorrectly classified as negative (false negative; FN), the number of negative samples incorrectly classified as positive (false positive; FP), and the number of negative samples correctly classified as negative (true negative; TN) [22]. As usual, in this context, "positive" corresponds to bound while "negative" corresponds to unbound.

The auPRC score is the "area under the precision-recall curve". To calculate this, the recall (i.e. true positive rate) is plotted against the precision (i.e. positive predictive value) and the area under the curve is computed, where recall $= \frac{TP}{TP+FN}$ and precision $= \frac{TP}{TP+FP}$ [22].

Note that computing the auPRC only involves TP, FN, and FP. Because the auPRC does not use

the number of true negatives (TN), it is a more informative metric for a setting like this, where the number of true negatives is, comparatively, overwhelmingly large. Finally, note that the auPRC may range anywhere from, as a lower bound, the fraction of positive samples in the dataset, to 1, as an upper bound; higher scores are preferred [22]. In a setting like this, where the fraction of positive samples is so small, even a mid-range score is considered a substantial improvement. Therefore, in all results we compare model performance using auPRC.

5.2 Conclusions

Our results show that there is a significant prediction performance gap between single-species and cross-species models. That is, when testing on a given target species' genomic sequence, deep learning models trained on multiple different species' binding sites for a given transcription factor consistently under-perform significantly compared to models trained on the target species' binding sites. Moreover, in contrast to previous work on the topic [16], neither the domain-adaptive nor the binary adversarial architecture noticeably bridged this performance gap.

First, we address the single-species vs. multi-species performance gap. One possible explanation for this are the species-specific sequence elements which cannot be captured by the multi-species model. For example, when the target genome is the human genome, the multi-species model is trained on all other species and has never seen training examples specific to the human genome, such as transposable elements like Alu elements [18]. These appear frequently in the human genome, and if they are mis-predicted as "bound", this increases the number of false positives and reduces the auPRC. In contrast, the human-trained single-species model does not have this problem. Also note again that some of these species are, genomically and evolutionarily, more closely related to humans and serve as more reasonable proxies for studying humans. Meanwhile, others of these species serve as poorer predictors of binding in humans. The multi-species model, therefore, may struggle compared to single-species models trained on the usual model species (e.g. mouse).

Next, we address the lack of improvement when using the domain-adaptive architecture compared to the multi-species model. Particularly, we note that neither the more complex domain-adaptive model (with a multi-class species classifier) nor the simpler binary adversarial model (with a binary species classifier) exhibited better performance than the multi-class binding predictor, when measured by auPRC. This could, in fact, be for the same reason discussed above. The aim of the domain-adaptive architecture is to detect the sequence predictors of binding which are common among the different species. Given that several of the species used here are relatively uncommon model species and do not necessarily share their regulatory regions with the human genome, binding events may not necessarily be largely shared.

In fact, when comparing CEBPA and HNF4a binding events in particular, previous studies [58] have shown that binding events are shared approximately 10%-22% of the time between any two mammalian genomes. When compared to the opossum genome (monDom5), this drops to only 6-8%. Finally, when compared to the chicken genome (galGal6), this drops further to only 2%. In fact, the same study showed that only 35 binding events were found to be shared by all five species studied: in other words, less than 0.3% of the total binding sites on the human genome. They note that in mammalian genomes, binding events occurring within species-specific repetitive regions are actually more common than binding events conserved across species [58].

When taking this into consideration, we conclude that in fact, most transcription factor binding is species-specific, and binding events common across species, particularly when including more divergent species such as opossum and chicken, are relatively rare. Indeed, this is the premise of our approach. We understand that due to differences in the underlying genome, the precise locations of transcription factor binding events are poorly conserved across species, but the sequence features that determine transcription factor binding should be conserved across species. Through this lens, our results seem more promising: to some extent, machine learning models are able to predict transcription factor binding across species despite vast differences in binding profiles. However, when using genomic data from this wide variety of species, it may be difficult to gather more than merely the core commonalities which are conserved across species. It may be more

informative and productive to restrict analysis to the usual model species, which are more closely related to and better proxies of one another.

5.3 Limitations

Here, we discuss a few different types of limitations of the models, results, and analysis presented previously. Every design decision impacts model training, often in unforeseen ways, and small choices made throughout the data analysis process can have large impacts later on. A few examples of potential limitations are as follows.

5.3.1 Model Hyperparameter Tuning

The most salient limitation of the above analysis is the lack of hyperparameter tuning performed. The basic model architecture used to train these models is fairly standard. However, in deep learning model training, it is not enough to simply decide on a sequence of neural network layers, each performing a different type of computation. Most algorithms involve several hyperparameters, i.e. "knobs" to be adjusted [7]. That is, these are variables that are set by us before the algorithm is actually applied to any input data. They are not learned by the model during training and do not change during the course of training. Examples include the model's learning rate, batch size, and layer-specific hyperparameters (e.g. number of convolutional filters within a layer). The choice of hyperparameters defines a particular model within a family of similar models, and can have a significant impact on training [7]. It is therefore important to choose the optimal set of hyperparameters; this is usually done via a hyperparameter "sweep", in which different values of each hyperparameter are systematically tested in different combinations until the optimal set of hyperparameter values is reached.

The models presented here have been trained only once with a particular set of hyperparameters. They have not been specifically tuned to optimize performance, and therefore we cannot be sure that the results presented are the best possible outcome. It is possible that with additional

model tuning, results could improve. We would also like to train multiple replicates of each model in order to ensure consistency and reliability of results presented.

5.3.2 Computational Limitation

On a related note, another computational limitation involved here was the tradeoff between the volume of data used in training and the time taken to train models. Above, hyperparameter tuning and replication were not performed because of the time required to train many different versions of each model. Another modification made to reduce training time is subsampling, as described in Chapter 3. That is, only the genomic windows overlapping the 5000 most prominent ChIP-Seq peaks for every transcription factor binding to each species' genome were used, and the unbound windows were undersampled to match to maintain a balance between the classes. It is possible that avoiding subsampling would allow for more accurate results because the model would see more examples of bound and unbound windows during training. However, training models without subsampling would have taken significantly longer.

5.3.3 Class Imbalance

The issue of class imbalance has been described at length in previous chapters. This is a common problem arising in biological and biomedical research, and often the smaller class of data is the class of interest. However, transcription factor binding exhibits extreme class imbalance; recall that only about 1-1.5% of the genome is labelled as bound by a given transcription factor.

It is crucial to determine some evaluation metric by which to track model performance; this allows us to better understand its prediction accuracy and choose methods for model improvement. However, many types of summary metrics are rendered misleading in an imbalanced data setting. This presents challenges when it comes to measuring and comparing model performance in a feasible and productive manner such that models can then be optimized.

Two of the most common evaluation metrics are the auROC and auPRC, or area under the receiver operating characteristic (the false positive rate vs. the true positive rate) and the area

under the precision-recall curve, respectively. However, auROC is not well-suited for imbalanced data, and can be very high simply due to the fact that very little of the data falls into one of the classes [14]. This is because a small number of correct or incorrect predictions can result in a large change in the auROC score. Measuring the auPRC is generally preferred for imbalanced settings, as it focuses on the minority class. However, this, too, may be misleading in targeted validation experiments [14].

Aside from evaluating performance, we also aim to combat class imbalance during training. In this work, only data-level methods were used: specifically, selection of appropriate training datasets. During training, we aimed to provide the model with balanced data in each epoch. To do this, we re-used the same set of "positive" (bound) windows in each epoch and under-sampled from the set of "negative" (unbound) windows so that the number of positive and negative samples were the same in each epoch, but over the course of training the model saw a wide variety of unbound samples.

There are other techniques used for handling class imbalance that we did not explore, which could also improve results [30]. For example, in some cases informed over-sampling (i.e. duplicating samples from the minority group or even producing artificial minority samples) has been shown to be effective. There are also algorithm-level methods which, instead of altering the training data distribution, adjust the learning process to place more importance on the minority examples. This decreases the likelihood that the model will incorrectly classify minority instances, reducing the number of false negatives and raising the auPRC score. Finally, there are hybrid methods which strategically combine the above data sampling adjustments and algorithmic adjustments; these have been shown to often outperform other methods in many cases.

5.3.4 Dataset Design

For the multi-species models, because we subsampled the positive (bound) data before training, we can be a bit more certain that these genomic windows truly are bound by the transcription factor in question. Otherwise, there is certainly some information lost in data processing. In particular,

peak calling is run on the ChIP-Seq data, extracting a number of "bound" peaks from the genome-wide signal. Note that peak calling depends on defining some threshold, above which a peak is denoted a "bound" site and below which a peak is denoted as an "unbound" site. Inevitably, there is a set of regions close to this threshold, some of which will be labelled as "bound" while the others are labelled as "unbound". It is possible that due to this thresholding, some regions that are true peaks may be left out and denoted unbound regions, while some noise regions are taken to be peaks and denoted bound regions [14]. Thus, processing the peak calling results into simply binary labels of "bound" and "unbound" causes some degree of loss of information. This could be avoided by denoting all regions with signal close to the threshold level as "ambiguous" and excluding them from the model training datasets [14]. However, even solutions to potential issues have their own benefits and drawbacks: excluding many regions from the training data, for example, can make the data provided to the neural network less informative and then cause the network to be less confidently able to predict on these types of ambiguous regions.

As discussed above, we can be quite sure that the bound examples used in training are indeed bound, due to our choice of subsampling method. On the other hand, we could also try a different method of subsampling. In this case, we chose to use only the windows which overlapped the 5000 most prominent ChIP-Seq peaks as our bound training set. However, if we were to use a different subsampling criterion (e.g. a random set of bound windows, rather than the "most prominent" bound windows), our results may differ. It could be that the larger ChIP-Seq peaks have different sequence characteristics than the mid-size ChIP-Seq peaks. In that case, the mid-size peaks would also be bound, but they would not have been used during our model training, therefore not allowing the model to learn their features. This may also explain why, compared to the single-species model, each of the other models which use subsampling (multi-species, domain-adaptive, and binary adversarial) overpredicted unboundedness. More experimentation with the training dataset used is required to determine the most informative and representative dataset.

Another crucial feature of the datasets used in this work is that they only include raw sequence data and binding labels. However, we know that binding selectivity is determined also by sev-

eral other factors regarding the regulatory environment at large in the cell, including chromatin accessibility, interactions with other transcription factors and cofactors, DNA methylation, and post-translational modifications [2, 68, 54, 69, 63, 46, 40]. These other regulatory mechanisms certainly play an important role in transcription factor binding preferences, and we may have seen better performance if we took other mechanisms into account, like chromatin accessibility information.

5.3.5 Model Interpretability

Machine learning models have demonstrated great success in many different fields due to their ability to learn complex patterns in data and ultimately make predictions about previously unseen data points [47]. In computational genomics specifically, deep learning models have been able to successfully take genomic sequences as input and predict regulatory function. That is, through repeated training on sequences and their associated regulatory function values, these models are often able to approximate a sequence-function relationship by learning the sequence patterns that are predictive of regulatory function [36]. With a trained deep learning model, then, researchers are able to predict the effects on regulatory function that are associated with (for example) particular sequence variants of interest.

However, machine learning models have often been treated as a black box: input data goes in and the model yields some output, but it is not exactly clear how the model reached its prediction. In recent years, a new area of focus is, in addition to using machine learning models to make predictions, the ability to interpret what a model has learned. After all, the aim is for these models to learn the ground truth regarding the underlying biological processes. This allows us not only to trust the model predictions more fully, but also, if we are able to extract the features the model has learned, to gain a more thorough understanding of the biological mechanisms in question [47].

It has recently been shown that convolutional neural network architecture, specifically filter size and pooling layers, influence the extent to which sequence motifs are learned by convolutional filters. In fact, the design choices made in the model architecture may influence how effectively the

model learns specific binding motifs, and there are a number of methods available for visualizing the sequence motifs learned by the first convolutional filter of a neural network [35]. In this work, we focused only on the prediction task; we did not attempt to interpret the convolutional filters of the network and characterize the particular sequence motifs detected.

Chapter 6

FUTURE WORK

Finally, we discuss ideas for future work based on the above results and analysis. We aim to improve performance of the neural networks built, rectifying the limitations of the current analysis.

6.1 Hyperparameter Tuning

As described previously, the particular choice of hyperparameters used when training a model can have a significant impact on model training: what it is able to learn and therefore the results from testing the model. Training deep learning models involve choosing values for several different hyperparameters. As mentioned, a major limitation of this analysis is that I was not able to test different values of the various hyperparameters, as would have been preferable. The next step in this analysis, therefore, would be to perform a hyperparameter sweep. This is often done in a grid search method [7]. First, each hyperparameter is given several different possible values, first at a relatively coarse resolution. The model is trained with each possible choice of hyperparameters to determine the optimal combination of hyperparameter values amongst those options. Once this set

of values has been determined, we may perform a finer-resolution hyperparameter sweep, in which hyperparameter values are further optimized. This grid search may also be optimized somewhat by gradually pruning combinations which are extremely unlikely to offer better performance. This allows us to eliminate some possible combinations and reduce the time required to perform the sweep.

6.2 Class Imbalance

Another next step would be to employ alternate methods to handle the vast class imbalance found in transcription factor binding data. In this analysis, we have undersampled from the majority class. One alternate (or additional) method would be to oversample from the minority class; however, this presents the risk of overfitting. Instead, my next step would be to combine majority undersampling with algorithmic methods of handling class imbalance. In general, these methods employ different techniques (e.g. cost-sensitive learning [30]) to emphasize the importance of the minority class and de-emphasize the importance of the majority class. Hybrid methods like this have often exhibited the greatest success.

6.3 Other Regulatory Mechanisms

If possible, it may also be helpful to account for other factors within the cell's regulatory environment. I acknowledge that this data may be difficult to find for some of the more unconventional model species, though it is likely readily available for the conventional ones, such as human and mouse. However, if it is at all possible, this may offer additional insight into binding specificity for different transcription factors, as they certainly depend on more than merely the genomic sequence.

6.4 Domain Adaptation

Gradient reversal is only one method of implementing a domain adaptive architecture. There are a number of other ways of detecting and replicating a domain shift in deep learning. Moreover, the ability to create a custom neural network layer such as a gradient reversal layer is quite powerful and opens up a world of alternate possibilities. In this way, we may implement any type of computation as a neural network layer and can experiment with techniques other than gradient reversal.

6.5 Ensemble Models

Ensemble learning refers to machine learning methods which combine multiple models to make an ultimate decision. The premise is that by using multiple models, the prediction error of each will be offset by the others, and the combined prediction performance will be better than any individual [66]. Today, they are a very effective approach and often outperform singular models.

In fact, two particular challenges that ensemble models are used to mitigate are particularly relevant to this work. One is class imbalance: ensemble methods can be used to train different models using different balanced subsamples of the data. Another is concept drift: ensemble approaches have been shown to improve performance in settings in which the distribution of features and labels changes [66]. Given that two particular challenges in this work are class imbalance and domain shift, ensemble models could be particularly helpful.

In this case, we may want to train one domain-adaptive model for each species (as opposed to using all species in domain adaptation, as we have done). The outputs of each of these models can then be fed into a simple classifier which makes a final decision on binding prediction.

6.6 Phylogenetic Weighting

The ensemble model described above would use separate domain adaptive models for each species, and then create a simple predictor on top of that which takes in each model's binding prediction and outputs a final result. Related to this, we may want to specify (or learn by using a very simple model, e.g. regression) a weight given to each species on the prediction task for a particular test species. This would allow us to "emphasize" the results from some of the species which are phylogenetically closer to the test species. For example, when predicting on humans (hg38), we may want to give a higher weight to macaque (rheMac10) and mouse (mm10) because these species are more closely related to humans, compared to chicken (galGal6) or opossum (monDom5). This is essentially a simple form of an ensemble model, with biological and evolutionary basis. It is reasonable to expect that phylogenetically closer species are better predictors for one another.

From an evolutionary standpoint, we may also ask whether the sites that are correctly predicted as bound by both human and multi-species models are particularly significant. For example, if the multi-species model is successful in learning the binding features common between different species, are the sites predicted as bound by both human and multi-species models the ones that are particularly conserved in different species through evolution?

6.7 Conclusions

As more and more experimental data is available to us and we build our computational power and access to a wealth of computational resources, computational methods are becoming more and more dominant in many different fields. We notice that analogous computational methods can be used on different datasets in different fields, as they are very efficient, quick, and adaptable methods. As we have seen with transcription factor binding, computational methods may even help domain experts hypothesize about the underlying ground truth of biological phenomena. Domain

adaptation could potentially be useful for predicting transcription factor binding in settings where data is not available for a given species or cell type. However, we suggest that in cross-species problems such as this, it may be important to (1) account for known species-specific genomic features, like repetitive transposable elements, and/or (2) modify how we use the data and predictions from each species, using domain adaptation on one species at a time and combining the predictions from each species so that they are as informative as possible of binding on the human genome. Overall, more work is certainly necessary to replicate model training and experiment with different model hyperparameters, methods of handling class imbalance, and synthesis of predictions from multiple different models.

Bibliography

[1] I. Albert, S. Wachi, C. Jiang, and B. F. Pugh. Genetrack–a genomic data processing and visualization framework. *Bioinformatics*, 24(10):1305–1306, May 2008.

[2] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter. *Molecular Biology of the Cell*. Garland Science, 6 edition, 2015.

[3] B. Alipanahi, A. Delong, M. T. Weirauch, and B. J. Frey. Predicting the sequence specificities of dna- and rna-binding proteins by deep learning. *Nature Biotechnology*, 33(8):831–838, 2015.

[4] H. M. Amemiya, A. Kundaje, and A. P. Boyle. The encode blacklist: Identification of problematic regions of the genome. *Scientific Reports*, 9(1):9354, 2019.

[5] B. Ballester, A. Medina-Rivera, D. Schmidt, M. Gonzàlez-Porta, M. Carlucci, X. Chen, K. Chessman, A. J. Faure, A. P. W. Funnell, A. Goncalves, C. Kutter, M. Lukk, S. Menon, W. M. McLaren, K. Stefflova, S. Watt, M. T. Weirauch, M. Crossley, J. C. Marioni, D. T. Odom, P. Flicek, and M. D. Wilson. Multi-species, multi-transcription factor binding highlights conserved control of tissue-specific biological pathways. *Elife*, 3:e02626, Oct 2014.

[6] S. Ben-David, J. Blitzer, K. Crammer, A. Kulesza, F. Pereira, and J. W. Vaughan. A theory of learning from different domains. *Machine Learning*, 79(1):151–175, 2010.

[7] Y. Bengio. Practical recommendations for gradient-based training of deep architectures, 2012.

[8] S. Bionetworks. Encode-dream in vivo transcription factor binding site prediction challenge. https://www.synapse.org/#!Synapse:syn6131484/wiki/402026.

[9] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer Science+Business Media, LLC, 2006.

[10] Y. Blat and N. Kleckner. Cohesins bind to preferential sites along yeast chromosome iii, with differential regulation along arms versus the centric region. *Cell*, 98(2):249–259, Jul 1999.

[11] N. Buduma and N. Lacascio. *Fundamentals of Deep Learning: Designing Next-Generation Machine Intelligence Algorithms*. O'Reilly Media, Inc., 2017.

[12] M. Burrows and D. Wheeler. A block-sorting lossless data compression algorithm. Technical report, Systems Research Center, May 1994.

[13] C. Chen, J. Hou, X. Shi, H. Yang, J. A. Birchler, and J. Cheng. Deepgrn: prediction of transcription factor binding site across cell-types using attention-based deep neural networks. *BMC Bioinformatics*, 22(1):38, 2021.

[14] T. Ching, D. S. Himmelstein, B. K. Beaulieu-Jones, A. A. Kalinin, B. T. Do, G. P. Way, E. Ferrero, P.-M. Agapow, M. Zietz, M. M. Hoffman, W. Xie, G. L. Rosen, B. J. Lengerich, J. Israeli, J. Lanchantin, S. Woloszynek, A. E. Carpenter, A. Shrikumar, J. Xu, E. M. Cofer, C. A. Lavender, S. C. Turaga, A. M. Alexandari, Z. Lu, D. J. Harris, D. DeCaprio, Y. Qi, A. Kundaje, Y. Peng, L. K. Wiley, M. H. S. Segler, S. M. Boca, S. J. Swamidass, A. Huang, A. Gitter, and C. S. Greene. Opportunities and obstacles for deep learning in biology and medicine. *Journal of The Royal Society Interface*, 15(141):20170387, 2018.

[15] F. Chollet. *Deep Learning with Python*. Manning Publications Co., 2018.

[16] K. Cochran, D. Srivastava, A. Shrikumar, A. Balsubramani, R. C. Hardison, A. Kundaje, and S. Mahony. Domain-adaptive neural networks improve cross-species prediction of transcription factor binding. *Genome Research*, 2022.

[17] P. J. A. Cock, C. J. Fields, N. Goto, M. L. Heuer, and P. M. Rice. The sanger fastq file format for sequences with quality scores, and the solexa/illumina fastq variants. *Nucleic Acids Res*, 38(6):1767–1771, Apr 2010.

[18] A. G. Diehl, N. Ouyang, and A. P. Boyle. Transposable elements contribute to cell and species-specific chromatin looping and gene regulation in mammalian genomes. *Nature Communications*, 11(1):1796, 2020.

[19] C. ENCODE Project. An integrated encyclopedia of dna elements in the human genome. *Nature*, 489(7414):57–74, Sep 2012.

[20] G. Eraslan, Ž. Avsec, J. Gagneur, and F. J. Theis. Deep learning: new computational modelling techniques for genomics. *Nature Reviews Genetics*, 20(7):389–403, 2019.

[21] N. C. for Biotechnology Information. Fasta format for nucleotide sequences, 2021.

[22] G.-H. Fu, L.-Z. Yi, and J. Pan. Tuning model parameters in class-imbalanced learning with precision-recall curve. *Biometrical Journal*, 61(3):652–664, 2019.

[23] T. S. Furey. Chip–seq and beyond: new and improved methodologies to detect and characterize protein–dna interactions. *Nature Reviews Genetics*, 13(12):840–852, 2012.

[24] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky. Domain-adversarial training of neural networks. *Journal of Machine Learning Research*, 17(59):1–35, 2016.

[25] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. The Massachusetts Institute of Technology Press, 2016.

[26] Y. Guo, G. Papachristoudis, R. C. Altshuler, G. K. Gerber, T. S. Jaakkola, D. K. Gifford, and S. Mahony. Discovering homotypic binding events at high spatial resolution. *Bioinformatics*, 26(24):3028–3034, Dec 2010.

[27] E. A. Hoffman, B. L. Frey, L. M. Smith, and D. T. Auble. Formaldehyde Crosslinking: A Tool for the Study of Chromatin Complexes *. *Journal of Biological Chemistry*, 290(44):26404–26411, Oct. 2015. Publisher: Elsevier.

[28] J. Huang, A. Smola, A. Gretton, K. Borgwardt, and B. Schölkopf. Correcting sample selection bias by unlabeled data. In *Advances in Neural Information Processing Systems 19*, pages 601–608, Cambridge, MA, USA, Sept. 2007. Max-Planck-Gesellschaft, MIT Press.

[29] I. Illumina. Precise analysis of dna–protein binding sequences. `https://www.illumina.com/techniques/sequencing/dna-sequencing/chip-seq.html`, 2022.

[30] J. M. Johnson and T. M. Khoshgoftaar. Survey on deep learning with class imbalance. *Journal of Big Data*, 6(1):27, 2019.

[31] A. Kakumanu, S. Velasco, E. Mazzoni, and S. Mahony. Deconvolving sequence features that discriminate between overlapping regulatory annotations. *PLoS Comput Biol*, 13(10):e1005795, Oct 2017.

[32] M. Karimzadeh, C. Ernst, A. Kundaje, and M. M. Hoffman. Umap and bismap: quantifying genome and methylome mappability. *Nucleic Acids Research*, 46, 2018.

[33] D. R. Kelley, J. Snoek, and J. L. Rinn. Basset: learning the regulatory code of the accessible genome with deep convolutional neural networks. *Genome Research*, 26(7):990–999, 2016.

[34] W. J. Kent, C. W. Sugnet, T. S. Furey, K. M. Roskin, T. H. Pringle, A. M. Zahler, and D. Haussler. The human genome browser at ucsc. *Genome Research*, 12(6):996–1006, 2002.

[35] P. K. Koo and S. R. Eddy. Representation learning of genomic sequence motifs with convolutional neural networks. *PLOS Computational Biology*, 15(12):1–17, 12 2019.

[36] P. K. Koo and M. Ploenzke. Deep learning for inferring transcription factor binding sites. *Curr Opin Syst Biol*, 19:16–23, Feb 2020.

[37] W. M. Kouw and M. Loog. An introduction to domain adaptation and transfer learning, 2019.

[38] B. Langmead, C. Trapnell, M. Pop, and S. L. Salzberg. Ultrafast and memory-efficient alignment of short dna sequences to the human genome. *Genome Biology*, 10(3):R25, 2009.

[39] T. I. Lee and R. A. Young. Transcriptional regulation and its misregulation in disease. *Cell*, 152(6):1237–1251, Mar 2013.

[40] K. M. Lelli, M. Slattery, and R. S. Mann. Disentangling the many layers of eukaryotic transcriptional regulation. *Annual Review of Genetics*, 46(1):43–68, 2012. PMID: 22934649.

[41] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, and R. Durbin. The sequence alignment/map format and samtools. *Bioinformatics*, 25(16):2078–2079, Aug 2009.

[42] H. Li and N. Homer. A survey of sequence alignment algorithms for next-generation sequencing. *Brief Bioinform*, 11(5):473–483, Sep 2010.

[43] S. Mahony. Domainfinder. `https://github.com/seqcode/seqcode-core/tree/master/src/org/seqcode/projects/seed`, 2016.

[44] S. Mahony, M. D. Edwards, E. O. Mazzoni, R. I. Sherwood, A. Kakumanu, C. A. Morrison, H. Wichterle, and D. K. Gifford. An integrated model of multiple-condition chip-seq data reveals predeterminants of cdx2 binding. *PLOS Computational Biology*, 10(3):1–14, 03 2014.

[45] S. Mahony and B. F. Pugh. Protein-dna binding in high-resolution. *Crit Rev Biochem Mol Biol*, 50(4):269–283, 2015.

[46] L. D. Moore, T. Le, and G. Fan. Dna methylation and its basic function. *Neuropsychopharmacology*, 38(1):23–38, 2013.

[47] W. J. Murdoch, C. Singh, K. Kumbier, R. Abbasi-Asl, and B. Yu. Definitions, methods, and applications in interpretable machine learning. *Proceedings of the National Academy of Sciences*, 116(44):22071–22080, 2019.

[48] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J Mol Biol*, 48(3):443–453, Mar 1970.

[49] S. J. Pan, I. W. Tsang, J. T. Kwok, and Q. Yang. Domain adaptation via transfer component analysis. *IEEE Transactions on Neural Networks*, 22(2):199–210, 2011.

[50] P. J. Park. Chip–seq: advantages and challenges of a maturing technology. *Nature Reviews Genetics*, 10(10):669–680, 2009.

[51] W. R. Pearson and D. J. Lipman. Improved tools for biological sequence comparison. *Proc Natl Acad Sci U S A*, 85(8):2444–2448, Apr 1988.

[52] J. Pevsner. *Bioinformatics and Functional Genomics*. Wiley, 3 edition, 2015.

[53] C. Pockrandt, M. Alzamel, C. S. Iliopoulos, and K. Reinert. GenMap: ultra-fast computation of genome mappability. *Bioinformatics*, 36(12):3687–3692, 04 2020.

[54] M. Ptashne and A. Gann. *Genes and Signals*. Cold Spring Harbor Laboratory Press, 2002.

[55] D. Quang and X. Xie. Factornet: A deep learning framework for predicting cell type specific transcription factor binding from nucleotide-resolution sequential data. *Methods*, 166:40–47, 2019. Deep Learning in Bioinformatics.

[56] A. R. Quinlan and I. M. Hall. BEDTools: a flexible suite of utilities for comparing genomic features. *Bioinformatics*, 26(6):841–842, 01 2010.

[57] A. R. Quinlan and I. M. Hall. Bed file format - definition and supported options, December 2021.

[58] D. Schmidt, M. D. Wilson, B. Ballester, P. C. Schwalie, G. D. Brown, A. Marshall, C. Kutter, S. Watt, C. P. Martinez-Jimenez, S. Mackay, I. Talianidis, P. Flicek, and D. T. Odom. Five-vertebrate chip-seq reveals the evolutionary dynamics of transcription factor binding. *Science*, 328(5981):1036–1040, May 2010.

[59] A. Shcherbina, A. Shrikumar, S. Kundu, and A. Kundaje. Seqdataloader. `https://github.com/kundajelab/seqdataloader`, 2019.

[60] T. Siggers and R. Gordan. Protein-dna binding: complexities and multi-protein codes. *Nucleic Acids Research*, 42, Feb 2014.

[61] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *J Mol Biol*, 147(1):195–197, Mar 1981.

[62] G. D. Stormo. DNA binding sites: representation and discovery . *Bioinformatics*, 16(1):16–23, 01 2000.

[63] G. D. Stormo. *Introduction to Protein-DNA Interactions*. Cold Spring Harbor Laboratory Press, 2013.

[64] S. Sun, H. Shi, and Y. Wu. A survey of multi-source domain adaptation. *Information Fusion*, 24:84–92, 2015.

[65] G. A. T. Team. Phred-scaled quality scores, 2021.

[66] C. Tebaldi and R. Knutti. The use of the multi-model ensemble in probabilistic climate projections. *Philosophical transactions of the royal society A: mathematical, physical and engineering sciences*, 365(1857):2053–2075, 2007.

[67] A. W. Trask. *Grokking Deep Learning*. Manning Publications Co., 2019.

[68] L. A. Urry, M. L. Cain, S. A. Wasserman, P. V. Minorsky, J. B. Reece, N. A. Campbell, and R. B. Jackson. *Biology*. Pearson, 12 edition, 2014.

[69] D. Voet, J. G. Voet, and C. W. Pratt. *Fundamentals of Biochemistry: Life at the Molecular Level*. Wiley, 5 edition, 2016.

[70] Y. Zhang, T. Liu, C. A. Meyer, J. Eeckhoute, D. S. Johnson, B. E. Bernstein, C. Nusbaum, R. M. Myers, M. Brown, W. Li, and X. S. Liu. Model-based analysis of chip-seq (macs). *Genome Biol*, 9(9):R137, 2008.