Deep Learning for Promoter Recognition: A Robust Testing Methodology

by

Raul Ivan Perez Martell
B.Sc., Monterrey Institute of Technology and Higher Education, 2016

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science

in the Department of Computer Science

Deep Learning for Promoter Recognition: A Robust Testing Methodology

by

Raul Ivan Perez Martell
B.Sc., Monterrey Institute of Technology and Higher Education, 2016

Supervisory Committee

Dr. Ulrike Stege, Supervisor
(Department of Computer Science)

Dr. Hosna Jabbari, Departmental Member
(Department of Computer Science)

Supervisory Committee

Dr. Ulrike Stege, Supervisor
(Department of Computer Science)

Dr. Hosna Jabbari, Departmental Member
(Department of Computer Science)

ABSTRACT

Understanding DNA sequences has been an ongoing endeavour within bioinformatics research. Recognizing the functionality of DNA sequences is a non-trivial and complex task that can bring insights into understanding DNA. In this thesis, we study deep learning models for recognizing gene regulating regions of DNA, more specifically promoters. We first consider DNA modelling as a language by training natural language processing models to recognize promoters. Afterwards, we delve into current models from the literature to learn how they achieve their results. Previous works have focused on limited curated datasets to both train and evaluate their models using cross-validation, obtaining high-performing results across a variety of metrics. We implement and compare three models from the literature against each other, using their datasets interchangeably throughout the comparison tests. This highlights shortcomings within the training and testing datasets for these models, prompting us to create a robust promoter recognition testing dataset and developing a testing methodology, that creates a wide variety of testing datasets for promoter recognition. We then, test the models from the literature with the newly created datasets and highlight considerations to take in choosing a training dataset. To help others avoid such issues in the future, we open-source our findings and testing methodology.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to thank:

Dr. Ulrike Stege, my supervisor, for her support, motivation, encouragement, patience, and mentoring throughout my Master's program. I personally thank her for giving me the opportunity to work with her. I am grateful for her continuous support and feedback that has helped me grow as a person and kept me going in those rough times.

Dr. Kwang Moo Yi, for his support in my endeavour to learn deep learning, and his great mentorship and guidance in my AI research.

Alison, for being my biology mentor throughout my degree, and for increasing my passion for biology with all her incredible knowledge.

My family, friends, all Rigi and Pita group members, and Nicole for supporting me in my research and creating great moments throughout my degree.

University of Victoria, for funding me with a scholarship.

Chapter 1

Introduction

Since the discovery of DNA, the goal of genomic research has been to derive useful information from DNA sequences. To this end, many researchers have been focusing on determining the function of genes and the elements that regulate them by finding variations among DNA sequences. Thanks to these efforts, we can begin to determine the significance of specific regions of DNA as well as begin to explain the complex interactions inside cells. This thesis focuses on a specific type of non-coding regulatory DNA region. We provide an overview on the methods and techniques used for the analysis of a regulatory region known as the *promoter*. We specifically focus on how machine learning and, most recently, deep learning has been and can be used in this area of research. This chapter introduces our motivation behind this research, the problem definition and objectives, our approach to address these objectives, and our research contributions.

1.1 Motivation

Computational or *in silico* methods are becoming more prevalent as more data is generated from laboratories all over the world, outpacing the researchers' capabilities to analyse the data. As the biological revolution took off in the mid-20^{th} century with the discovery of DNA, it highlighted the importance of genomics and started a new era of *High-Throughput Sequencing* that has dramatically impacted many areas of biological scientific research [104]. Now, biologists are struggling to keep up with the copious amounts of new data. Consequently, this area of research has seen significant growth regarding both published papers and the amount of data that needs to be

analyzed. Analyzing this data has become a central part of biology as it shifts from a hypothesis-driven approach to a more discovery or data-driven approach [53].

There are many uses for regulatory regions, such as promoters, in the biomedical field. In the case of plasmid vectors for prokaryotic organisms, promoters are studied for their use in transcription initiation of transgenes for research and therapeutic purposes. Gene therapy is a form of medicine used to cure illnesses by modifying functional DNA or genes. Promoters are essential in controlling the expression of these therapeutic genes to minimize the possibility of negative effects occurring during treatment [47]. Knowing how to control gene expression mechanisms can also give us clues about human diseases and possible related animal models on which to test these therapeutics. Regulatory DNA regions can also serve as data for *in silico* methods, which encompass disease analysis and control using computational models [34]. In silico models reduce the risk of a tested therapeutic to produce unexpected results when used *in vivo*.

Within an organism a promoter's DNA sequence for a given gene is the same regardless of the cell type. However, other factors between cell types, developmental and health/disease states can impact the functionality of that promoter. Promoter sequences between genes can exhibit extensive variability within an organism. Biologists are able to use techniques ranging from simple molecular biology methods to modern high-throughput omics screening techniques to identify suitable promoters [145]. These screening techniques take great time and effort in order to produce high-quality results. Therefore, an effective and efficient computational tool to detect promoters might in turn enable us to detect genes in silico, which will drastically cut costs and time consumption of biological assays.

Only about one percent of human DNA is made up of protein-coding genes; the other 99 percent is non-coding [99]. Non-coding DNA does not directly provide instructions for making proteins. However, these DNA regions are integral to the function of cells. Non-coding DNA can contain regulatory elements, structural elements of chromosomes, and instructions for specialized RNA molecules such as transfer RNA, ribosomal RNA, microRNA, and long non-coding RNA. Promoters are part of these regulatory elements that were thought to be less important than coding regions [138], but more and more studies demonstrate the crucial importance of non-coding DNA. Recent studies [138] emphasize this in areas such as evolution, where evolutionary changes can be driven by the complex mechanisms of gene regulation and not solely by polymorphism in coding DNA regions, as previously assumed.

Promoter recognition can be of use in the treatment of rare diseases, when aberrant promoters take place inside a genome. Approximately 80 percent of rare diseases are not acquired; they are inherited [61]. These diseases are caused by mutations or defects in genes, with children representing the majority of those afflicted with rare diseases[1]. Although genetic diseases might seem rare on a case-to-case basis, they affect a large number of people. Genetic diseases affect one in ten Americans (i.e., between 25 and 30 million people in the United States), and over 350 million people globally.[1] Rare diseases are also known as orphan diseases because drug companies were uninterested in developing treatments for these genetic conditions that are not widespread among the human population[2]. The better we understand DNA, the easier and cheaper it will be to cure rare genetic diseases that arise[2]. Aside from treating rare diseases, we anticipate a direct avenue into personal medicine. The better we understand these rare diseases, the better we will understand human genetics as a whole. All of these findings feed into medicine and treatment to end suffering caused by other diseases as well.

1.2 Problem Definition and Objectives

Proteins are complex molecules that, in conjunction with RNA, provide all the necessary elements for a cell's well-being including structure and regulation. These functional components need to be created regularly by the cell, as the components deteriorate over time. This creation process starts by reading the cell's DNA. Using DNA as a template, the cell can create all its functional components for use throughout its lifetime. According to the Encyclopedia of DNA Elements [36] (ENCODE), it is believed that about one percent of human DNA is ascribed to protein-coding exons. In other words, about one percent of DNA in the human genome ends up being used as the template in the production of protein. Previously, many researchers believed that the remaining percentage of DNA, also called *non-coding DNA* because of their lack of usage in protein coding, was 'junk' or had very little meaning or function. Now, with the help of projects such as ENCODE, 'junk' DNA is being studied and understood by researchers for its importance to the function and well-being of each cell. These projects, along with ENCODE, discovered that many of the genes are found in proximity to regulatory elements. These elements were found within multi-

[1] `https://globalgenes.org/2009/02/27/rare-disease-facts-and-figures/`

[2] `https://rarediseases.info.nih.gov/diseases/pages/31/faqs-about-rare-diseases`

ple regulatory DNA regions, including promoters, during gene regulation events such as transcription, a process of creating RNA from DNA.

Being able to classify and analyse the different DNA sequences has been an ongoing area of study in genomic research. As such, promoters are also in the process of being characterized since they vary in many different ways. Regulatory variation is present within organisms on a tissue level and across developmental stages [121]. Variation is also much more apparent within the biological domains. Bacterial or prokaryotic promoters are better understood because of their simplicity compared to eukaryotic promoters. This large variation within promoter sequences makes the computational identification of promoters no trivial task. There are still many questions to be answered in this decades old research-intensive subject area of biology [2]. In this thesis, we present a look into the journey of promoter recognition and highlight computational methods for identifying promoter regions *ab initio* that employ genomic data available to the public.

The computational analysis method for promoter recognition that we focus on is deep learning, motivated by its recent success in many other research areas. In addition to deep learning models' success in computer vision and language modeling, advances in deep learning have provided results within recent work in the field of bioinformatics. There are a myriad of deep learning studies suggesting improved performance in biological analyses, including microarray data analysis [94], gene expression inteference [20], single nucleotide polymorphism analysis [55], sequence-based protein interactions [127], protein structure prediction [37].

An essential component for the initiation of eukaryotic transcription of protein-coding genes is RNA polymerase II (RNAP II). This component is comprised of a complex of multiple proteins that work together to 'read' the DNA. While 'reading' the DNA, RNAP II also processes the DNA to make it suitable for the creation of the DNA's functional components counterparts. This process is explained in more detail in section 2.1.2. Specifically, the problem that we are analysing comprises of RNAP II promoter recognition in eukaryotes using deep learning models. Our objectives consist of reproducing the latest deep learning methods on promoter recognition, allowing us to compare them to obtain a vision of the current progress made on this problem. We also investigate how the promoter recognition problem can be approached via methods from natural language understanding. Our final objective is to improve the current methods and advance the knowledge in this field of research.

1.3 Approach

We first delve into the scientific and computational background knowledge that will enable us to reproduce and analyse the current promoter recognition approaches. To do this, we explore the literature for previous approaches to solving the promoter recognition problem in both prokaryotes and eukaryotes. These approaches are relatively recent. The problem was stated in the late 20^{th} century, when the first complete DNA sequencing of genomes changed the field of genomics. In order to analyse and evaluate the deep learning methods, we studied binary classification evaluation metrics for machine learning, as well as used an *attention mechanism* [4, 170] to interpret how the deep learning models were classifying promoters. Classification in machine learning follows the recognition problem. This is because recognizing a promoter when given a DNA sequence means that one can identify the parts of the sequence that are promoters. Promoter classification takes a DNA sequence and splits it into subsequences to then classify for each sub-sequence. Therefore, promoter classification can arrive to the same result as promoter recognition.

Deep learning has been used as a tool for natural language processing (NLP) tasks [4]. NLP is a field that relates linguistics and computer science to analyse human language. DNA sequences can be viewed as a type of unstructured language where different sections of the sequence mean different things according to their functionality within the cell. Named-entity recognition is a task in NLP that seeks to locate and classify entities in an unstructured text. The named entities for DNA sequences could be the different types of sequences depending on their transcribed function, making promoter recognition similar to the named-entity recognition task. With this knowledge, we approach the promoter recognition problem in a natural language perspective.

Our approach can be summarized in the following manner: we select relevant recent literature, review their methods, and reproduce them to then compare these existing methods. For the latter, we devise a methodology to run the test in a realistic environment. This gives researchers a way to test their algorithms and models not only in certain special scenarios, but in generalized situations as well. We give a detailed explanation on the steps to use and reproduce our testing methodology for use in promoter recognition. Finally, we discuss how promoter recognition can be addressed using our findings to advance a solution to this problem.

1.4 Contributions

The main contributions presented in this thesis are:

Contribution 1 Identification of shortcomings in current deep-learning-based promoter recognition methods by cross-testing models to determine most suitable promoter classifiers in the literature.

Contribution 2 Creation of a testing methodology for ab initio promoter recognition machine learning methods as a reference for future research to test models in a unified and comparable manner.

Contribution 3 An overview of the field with insights for improving the performance of current models.

Contribution 4 Release of our source code for our experiments and testing methodologies, along with current deep learning models from the literature. This code also includes an analysis suite of binary classification metrics useful for literature comparisons.

1.5 Thesis Outline

Chapter 2 describes the key concepts needed for this thesis in both the biological and computational realms. It expands upon the different types of promoter recognition methods and the data that is useful for such a task. It also discusses the different deep learning architectures and the frameworks needed to implement the theory.

Chapter 3 presents the related work in the area of promoter recognition in both eukaryotes and prokaryotes, as well as a brief history of the field, ending on the current state of the art using deep learning.

Chapter 4 introduces the evaluation metrics used to compare and validate binary classification models analogous to approaches executed by promoter recognition methods. We also describe advantages and disadvantages of the different evaluation metrics, and provide suggestions for evaluating models.

Chapter 5 describes how we dealt with DNA sequences as a natural language and tested the success of NLP techniques for the promoter recognition task. This gives us insights into how such models are characterizing promoter and non-promoter sequences.

Chapter 6 describes our implementations of approaches from the recent literature approaches and how we compared them, yielding the conclusion that there is a need for an improved evaluation methodology.

Chapter 7 describes in detail our methodology for evaluating promoter recognition approaches, which creates a new dataset for unified testing purposes.

Chapter 8 discusses the findings in chapters 5-7 and offers insights into how the process of promoter recognition can be improved. It also explains how this work can be expanded to contribute to the goal of understanding DNA thoroughly.

Chapter 2

Background

This section introduces background and terminology used in following chapters. It describes the biological background behind promoter recognition and the computational methods that we will be analysing throughout the thesis.

2.1 Genomics

Organisms are made out of cells. We are only starting to understand the instructions by which cells undergo their processes. These instructions are encoded by DNA and are located inside an organelle called the nucleus for eukaryotic cells and in a region called nucleoid for prokaryotic cells. Prokaryotes are characterized by being unicellular organisms in the bacteria and archea domain with no membrane-bound organelles and circular DNA molecules. Prokaryotic organism's largest and main instructions are encoded in a single circular DNA molecule, although many prokaryotes carry additional circular DNA called plasmids. These allow horizontal gene transfer between bacteria, even across species, and are a big part of antibiotic resistance spread. In contrast, eukaryotes can be unicellular or multicellular organisms in the eukarya domain represented by five kingdoms: Plants, Animals, Fungi, Protists, and Chromists. Eukaryotes have membrane-bound organelles and a nucleus with linear DNA in the form of chromatin. Eukaryotes also have mitochondrial DNA (*mtDNA*) within the mitochondria. It encodes genes required by the mitochondria and have a regulatory and expression pattern more similar to bacteria. We focus our work on eukaryotes without taking into consideration its mtDNA. Primary focus is given to the human species (homo sapiens).

A genome is the complete set of genetic material of an organism. Researchers first encountered the complete human DNA sequence in the form of chromosomes while studying cell division under a microscope, depicted in Figure 2.1. During cell division, DNA condense into a very densely packed form, where DNA gets coiled around proteins called *histones*. Packs of coiled DNA can in turn form *nucleosomes* which create the *supercoils* that are visible under a microscope. The mass of genetic material packed into supercoils and bound by other proteins and tethered is known as *chromatin*. When a cell is not dividing, the chromosomes get encapsulated by the cell's nucleus and assume a more relaxed state of packaging known as *euchromatin*. Depending on the cell type, the information that is needed will be unpacked from its coiled form in order for the cell to make use of the stored information contained within to start generating functional components. DNA that is kept firmly packed and thus suppressing gene activity is known as *heterochromatin*.

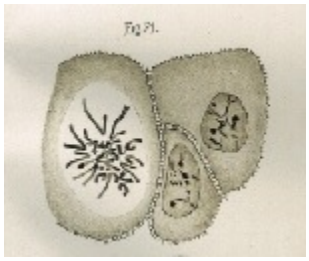


Figure 2.1: Partial drawing of chromosomes by Walther Flemming [7]

2.1.1 Gene regulation

DNA by itself serves as an information storage system for biological organisms for functional and hereditary purposes [1]. Some of the information or DNA sequences, known as *genes*, specify the sequence of functional components inside each of the organism's cells. This process is known as the *central dogma* and can be described in two steps. The first step is *transcription*, where the nucleotides in DNA are read and used for the creation of a second type of nucleotide chain, known as RNA. The second step reads the RNA created in the previous step and forms a chain of polypeptides or a *protein* through a process called *translation*. For some functional components, the central dogma can stop at the RNA level of transcription, as RNA can function as a catalyst by itself.

Using the central dogma, we can find the relation or similarity between DNA,

RNA and protein sequences. This creates an opportunity for *sequence homology* or shared ancestry between multiple organisms by finding organisms with similar genes or protein. There are three ways homology can occur between two DNA sequences. *Orthologs* are similar genes where the DNA sequences contain a common ancestor, both having evolved to distinct species from each a common population. This means that both organisms share this functionality or gene. *Paralogs* result from a duplication event where a gene gets repeated inside the organism and they separately evolve to have different functionality. *Xenologs* arise from horizontal gene transfer, or the transmission of genetic material between organisms that are not familial or related [95, 122].

DNA can be seen as a language of four letters that leads to the components which make us who we are as a species and as individuals [62]. These four letters represent nucleotides that form the commonly known double helical structure of DNA discovered by Franklin and Gosling [41], Wilkins et al. [151], Watson and Crick [146]. The four letters are 'A' for Adenine, 'C' for Cytosine, 'G' for Guanine, and 'T' for Thymine. They are categorized as such because of the different types of molecules that comprise them.

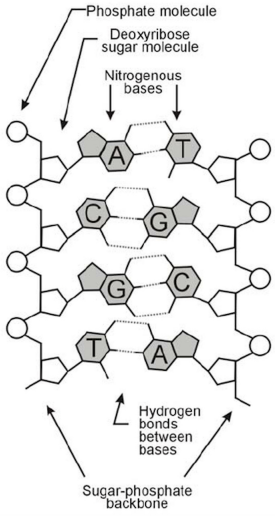


Figure 2.2: DNA structure depiction by Derek Stein [124]

The reading process required in transcription involves the structure of the chained DNA nucleotides. Each nucleotide consists of three parts. It has a sugar molecule known as Deoxyribose, a phosphate group and a nitrogenous base or nucleobase. This structure is depicted in Figure 2.2. The nucleobase differentiates the nucleotides

from each other. In the case of DNA, recall we have four types: 'A', 'C', 'G', 'T'. To form the DNA's double helical structure, two strands of DNA bind together between nucleobases via a chemical process known as hydrogen bonding. Nucleobases that have paired together are called *base pairs*. Each nucleobase has an affinity to bind to another specific nucleobase to make a base pair. Adenine binds with Thymine and Guanine binds with Cytosine. For RNA, we have the same number of types as DNA with the exception of Thymine being exchanged for Uracil. Deoxyribose contains five carbons in its molecular structure, and chemists have created a system for identifying these carbons using a numbering scheme. DNA chains are formed when nucleotides connect to each other through their phosphate group, so a DNA sequence will look like Figure 2.3. When reading DNA, the process always follows the same directionality. This direction is from $5'$ to $3'$ and originates from the numbering of the carbons previously mentioned.

When referring to gene regulation, the level of expression is important. DNA is not the sole way a cell can alter gene expression [54]. Cells can also exhibit heritable changes that do not involve alterations in the DNA sequence. Researchers found that molecules can bind to the DNA or chromatin proteins and alter the way the cell reads its genetic code. This becomes apparent when all cells in an organism share the same DNA, but they can have very different functions depending on the cell type. There are multiple epigenetic systems that have been identified. These include methylation, acetylation, phosphorylation, ubiquitylation, and sumoylation; other mechanisms are likely to be discovered as research continues. RNA-associated modifications are other significant epigenetic processes [35, 147].

An important process is DNA methylation. It involves adding a methyl group (CH_3) to DNA. This process is very specific and only happens in DNA regions where cytosine and guanine nucleotides are contiguous in a DNA chain. These regions are called CpG sites, or CpG islands when there is a high frequency of CpG sites in a designated region. DNA methylation changes the structure of DNA, and modifying the gene's interaction with the molecules needed for transcription such as RNAP. *Imprinting* is a condition where DNA methylation is used in some genes to differentiate the inherited paternal and maternal gene copies. Such differentiation can cause one of the copies to be silenced or inactivated. The counterpart of DNA methylation is DNA demethylation where a methyl group is removed [105, 35].

Acetylation involves adding an acetyl group (CH_3CO) to proteins such as histones. The counterpart of acetylation is deacetylation where an acetyl group is removed. The

Lysine residues of histones undergo regular acetylation and deacetylation, regulating the binding of histones to DNA [78]. In large quantities, this can change chromatin structure from euchromatin to heterochromatin or vice versa, creating a gene regulation mechanism.

Epigenetic changes are also targets for *post-translational modifications* (PTMs) such as phosphorylation, ubiquitylation, and sumoylation. These PTMs are known to bind to histones and other proteins that affect the structure of chromatin and affect gene regulation after translation has occurred [132]. This implies a method of gene expression alteration that can improve the stability of complex signaling pathways through a wide range of regulatory mechanisms [51]. Some RNAs such as small interfering RNA and microRNA in the form of antisense transcripts and noncoding RNAs can also modify chromatin structure by triggering histone modifications and DNA methylation, thus affecting gene expression [35].

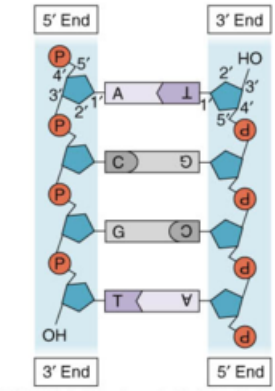


Figure 2.3: DNA directionality depiction by Ben Himme

In this thesis, when referring to DNA, we assume its normal form that contains both strands of nucleotide chains as a pair, as promoters include both strands of DNA. A strand is named depending on its relative directionality to the contextual sequence. The *sense strand* goes from $5'$ to $3'$. The *antisense strand* goes from $3'$ to $5'$. Both are shown in Figure 2.4.

Although genes are templates for the cell's functional components, genes do not control an organism's actions. Genes interact and respond to the organism's environment by providing the cell with the tools necessary for the wide array of circumstances at the appropriate time. *Constitutive genes* might be needed on demand and thus have to be produced at all times. Genes that are transcribed when in need are called *responsive genes*. The final type of gene is known as a *housekeeping gene*. It is a

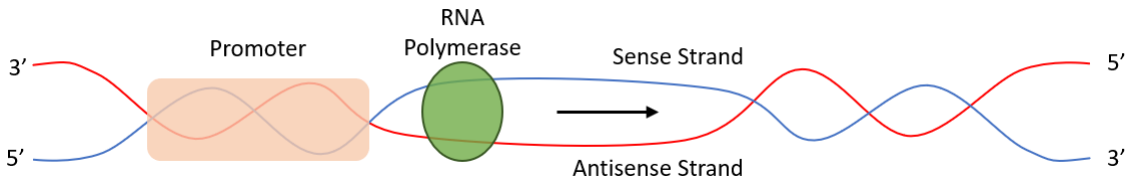


Figure 2.4: Depiction of DNA strands. DNA shown as blue and red lines. Black arrow showing the gene's relative directionality

constitutive gene that is transcribed at a relatively constant level and used mostly for the cell's maintenance and survival [46].

There are multiple ways of obtaining the data set for complete genomes of organisms. The University of California in Santa Cruz (UCSC) offers a web tool called Genome browser [101] that contains multiple organisms' genomes, as well as their phylogenetic relatedness. Apart from the vast array of genetic sequence data, it offers annotations for complete genomes, as well as RNA and protein data. It also offers pairwise and multiple alignments between the organisms as well as single nucleotide polymorphisms, or common nucleotide differences between organisms of the same species that have biological significance. Aside from the genomic web tool, it offers a variety of tools for genetic analysis. The European Bioinformatics Institute (EBI) similarly has a web tool called Ensembl [164]. EBI offers a set of tools and online functionality that performs similarly to UCSC counterparts. The United States National Center for Biotechnology Information (NCBI) offers a genome data viewer [26] very similar to the two previous alternatives. These organizations mirror data among each other and were initially set up as a way to distribute data effectively. Therefore, they contain the same nucleotide data sets and their differences can be attributed in their offered analysis tools. These analysis tools can make results vary in all three of these genomic tools.

2.1.2 Transcription factors

In order for transcription to occur for each specific gene inside a cell, an enzyme known as *RNA polymerase* (RNAP) must bind to the DNA to start the transcription process from DNA into RNA. In order for RNAP to bind to a specific site in the DNA, some proteins known as *transcription factors* (TF) have to locate the site first before binding themselves to the DNA. With these bindings in place, the RNAP is then recruited by the transcription factors to bind to the *transcription start site* (TSS),

and the transcription process initiates. The DNA sites where the transcription factors bind to are known as *transcription factor binding sites* (TFBS) and are typically 10 nucleotides long. Since these mechanisms are vital in the cell, the cell itself needs to safe guard against highly specific DNA sequences that would restrict binding. It is known that transcription factors are neither too specific nor too promiscuous, and their short length serves that purpose [125].

Eukaryotes and prokaryotes go through different transcription mechanisms. Prokaryotic transcription comprises of a single type of RNAP, while eukaryotic transcription is achieved by three different types of RNAP (I - III). These polymerases differ in their structure and the type of complexes they contain, as well as the class of RNAs they transcribe. RNAP I transcribes ribosomal RNAs (rRNAs), which are used to construct ribosomes that synthesize proteins inside the cell. RNA pol II transcribes RNAs that will become messenger RNAs (mRNAs) and also small regulatory RNAs - the former being the products used by ribosomes to synthesize proteins, while the latter play a role in regulatory processes such as activation and inhibition of gene transcription. Finally, RNA pol III transcribes small RNAs such as transfer RNAs (tRNAs), which help the ribosome decode the mRNA into amino acids that make up polypeptides, which in turn form the proteins in the cell.

RNAP needs multiple TFs as mediators to be able to bind to the DNA and start the transcription process. The DNA sequence where RNAP binds to start the transcription of a gene and regulates the gene's expression is known as a *promoter*. Thus the promoter can be seen as a region of gene regulation by influencing the transcriptional behaviour of the cell. One such way for regulation to occur is when a promoter is physically unavailable to an RNAP. A simple example of this phenomenon occurring on a large scale is when DNA is densely packed as seen in Figure 2.1. More commonly, DNA is packed inside a cell's nucleus depending on the specific cell's function and the tissue it belongs to. As previously mentioned, these cells have unpacked regions of DNA ready for transcription to take place for its tissue specific functionality.

TF cannot be understood functionally without accompanying detailed knowledge of the DNA sequences they bind. TFBS are frequently summarized as 'motif'—models representing the set of related, short sequences preferred by a given TF, which can be used to scan longer sequences (e.g., promoters) to identify potential binding sites [76]. Since their discovery, several organizations and researchers have begun to catalog all known TF, going as far as creating collections for specific cell lines of well known

organisms such as Homo Sapiens (human) and Mus Musculus (mouse).

There exist multiple databases of TFBS. JASPAR [39] is a collection on TFBS comprising about 1500 different eukaryotes. Their TFBS and motifs are modeled by matrices and can be converted into position weight matrices (PWM) or position specific scoring matrices (PSSM) where each row represents a nucleotide and the columns represent the position in the DNA sequence [167]. JASPAR is an open data access, non-redundant and experimentally defined database.

TRANSFAC [70] is a similar commercially available resource for eukaryotic TF and TFBS, with a toolkit for further analysis and exploration. These two previous databases are the main ones as multiple other databases are setup and forgotten after a set amount of time.

UniProbe [60] is a database that hosts data generated by universal protein binding microarray technology. This technology outputs DNA binding specificities of proteins. It contains binding data of more than 700 proteins and complexes in a collection of prokaryotic and eukaryotic organisms, including human and mouse.

ReMap [24] is an atlas of human regulatory regions collected from ChIP-seq analyses of TF, chromatin regulators, and transcriptional co-activators. ReMap's data can be accessed through UCSC Genome browser and Ensembl browser.

There are two main consortium projects that seek to produce functional annotations of whole genomes. As such, the projects try to understand the process of how cells can go from DNA to functional products such as RNA and protein. The first step to understanding this process is to comprehend transcription in all levels. These projects are ENCODE and FANTOM.

ENCODE [36] is based in UCSC and focuses on the annotation of the human genome, but also contains data of mice. It includes elements acting at the protein and RNA levels, as well as epigenetic or regulatory elements of active genes. All ENCODE data can be viewed in the UCSC genome browser. The data includes chromatin accessibility and peak analysis with DNase I hypersensitivity clusters, TF ChIP-seq, DNase, FAIRE-seq, Histone, and TFBS. It also includes multiple types of RNA-seq, DNA methylation, and regulatory marks, as well as other more specific types of data.

FANTOM [131] is based in RIKEN, a Japanese research institution. It focuses on the annotation of the mammalian genome. The project's goal is to understand the systems of life progressively, starting from the transcripts to the complete transcriptional regulatory network - in other words, how an individual life form works as

a system. FANTOM's data include a TSS prediction database using CAGE data, an atlas for promoters, enhancers, long noncoding RNAs, and micro RNAs. Similar to ENCODE, it also contains ChIP-seq and data from microarray experiments. Finally, it contains annotations for *complementary DNA* (cDNA) as well as representative transcript and protein sets.

Other notable and recent databases include GTRD, MANTA2, AnimalTFDB, TFBSDB, HOCOMOCO, and DBTSS. Most of these databases obtain their data from elsewhere and add a layer of analysis for TFBS prediction. GTRD focuses on human, mouse and other well known organisms. It uses ChIP-seq data along with DNase-seq data and creates a pipeline to derive resources to analyse regulated genes, predict TFBS and map them. GTRD contains a genome browser with all their data for convenient access and exploration [159]. MANTA2 specializes in human data and similarly to GTRD, it predicts TFBS using data from ReMap and JASPAR [40]. AnimalTFDB [58] contains TF and cofactors from about 100 animal genomes. It also contains a specialized human TF database and a tool for TFBS prediction. TFBSDB maps human genes and their motifs, TSS, and promoters from multiple sources including JASPAR, SELEX, TRANSFAC, UniProbe, ENCODE, and UCSC hg19 genome. TFBSDB also suggests an optimal promoter size in humans where the best results can be obtained [98]. HOCOMOCO [73] contains motifs for human and mouse. It uses a computational method to discover the motifs and cross validates them comprehensibly. The data used for the motif discovery method is from ChIP-seq data by the previously mentioned GTRD database. HOCOMOCO provides PWM, as well as dinucleotide PWM. DBTSS [128] as its name implies is a database of transcriptional start sites. It contains exact TSS positions in the genome based on experimentally validated sequencing technologies. This database focuses on human adult and embryonic tissue. It also integrates RNA-seq and ChIP-seq data from their cultured cell lines. The database has also recently included ENCODE epigenetic data to provide better TSS predictions.

2.1.3 Biological assays

Previously, we mentioned several sequencing methods that databases use as a data collection mechanism for computational use. The analysis of these methods provides us with a 'screenshot' of what the molecules inside the cells are doing, yielding insight into the biology and processes of the cell. There are many different types of methods

being used by researchers, and many still being developed. We will describe the ones relevant to promoter recognition.

DNase footprinting [43] is an in vitro method to identify the specific site of DNA binding proteins. It is based on the assumption that bound proteins protect the backbone of DNA from an enzyme, known as DNase I, that breaks down this backbone during contact. Thus, the footprinting method returns DNase I hypersensitive sites or regions of chromatin that are sensitive to cleavage by this enzyme [126]. This means that hypersensitive sites are DNA regions not protected by a protein binding them. Sites protected from DNase correlate to regulation regions in the DNA, as this means a protein such as a TF protected the DNA from cleavage. DNase sequencing requires the isolation of the nucleus of the cells and thus can be restrictive.

Phylogenetic footprinting [45] is a technique to identify TFBS within noncoding regions of DNA by comparing it to orthologous sequences. This assumes that regulatory sites are evolutionarily conserved, and that regulatory orthologs are known beforehand for the target sequences being investigated. This method requires careful distinction between heterologous and homologous genes, as only the latter are appropriate for use. Homologous genes are derived from a common origin as opposed to being from heterologous genes.

ChIP-seq [42] stands for Chromatin Immunoprecipitation sequencing, and is a method that is used to analyze DNA-protein interactions. It makes use of antibodies to bind to target proteins. Stretches of DNA bound to the proteins are then immunoprecipitated or separated from molecules not bounded by the antibody used. The DNA is copied thousands of times to obtain a detailed reading for the computational DNA sequencing algorithm to make an accurate consensus sequence. The sequencing resolution is often insufficient, leading to researchers proposing higher resolution methods [22].

HT-SELEX [91] is another in vitro method for the identification of DNA-protein interaction sequences. Proteins are washed with a random pool of nucleotide chains. The desired DNA sequences will bind to the proteins and the rest will be washed away. For accurate determination of the DNA sequence, amplification through PCR or *polymerase chain reaction* of the bounded nucleotide chains is required. After multiple runs, the target sequence will be acquired.

Universal PBM [8] stands for protein-binding microarrays. It is an in vitro method for characterizing TFBS. This technology makes use of double-stranded DNA microarrays where fluorescently labeled TFs are washed through the microarray chip. Some

TFs will then be able to bind to the DNA of the targetted TFBS. Scanning the microarray will reveal which TFs are bound to the TFBS by their fluorescent tag, and the sequence of the TFBS can be inferred by the complementary nature of DNA from the bounded TF.

RNA-seq [143] is a technique that can analyse the quantity and sequences of RNA in a cell or sample. It is most commonly used to find differential gene expression patterns in cells. As there are multiple types of RNA sequences, so are there multiple types of RNA-seq for each type of RNA. RNA can be encapsulated by two main types: small RNAs such as microRNAs, and long noncoding RNA. RNA-seq is constantly improving, and as new technology arises, newer methods continue being developed to provide more sensitive results. One such recent method is single cell RNA-seq [33].

CAGE [129] stands for cap analysis gene expression, and is a technique for the analysis of RNA. Unlike universal PBM and RNA-seq, CAGE is able to accurately identify the TSS and the corresponding promoters with the use of DNA sequencing. It takes advantage of the molecular structure of the RNA chain backbone to accurately find the TSS. When mRNA is formed from the transcription process of DNA, a method called *oligo-capping* can be used to replace the $5'$ end of the resulting mRNA to label its starting point. Once known, a DNA copy of the mRNA (cDNA) can be obtained and aligned to the original DNA where translation happened. This gives an accurate estimate of where the TSS is located in the original DNA sequence.

FAIRE-seq [120] stands for formaldehyde-assisted isolation of regulatory elements and is used to determine DNA regions associated with regulatory activity. In contrast to DNase sequencing, this method does not require the isolation of the nuclei, making it a more viable option.

Most assays described in this section make use of deep sequencing or next generation sequencing (NGS). This refers to techniques for DNA and RNA sequencing that generate large amounts of sequence data at high speed and low cost from a single run. A run alludes to the complete 'read' or processing of a sequence from one end to the other done by a sequencing instrument or machine. This is in contrast to *Sanger* sequencing, which is based on PCR and depends on the size of DNA sequence fragments created using special color-labeled nucleotide bases called dideoxynucleotides (ddNTPs). These sequence fragments are separated by their size through a glass capillary filled with gel using electrophoresis and the color of each fragment is read by a machine to infer the nucleotides at each position of the sequence. This process can only be done one fragment at a time using Sanger sequencing. NGS uses flow

cells which can bind to millions of DNA pieces at a time, which makes it able to read all these sequences at the same time, although the sequences generated are only 100 to 200 bases long; unlike Sanger sequencing which generates only one sequence of 700 to 1000 bases long in a single run. This makes Sanger sequencing a good choice when sequencing a short DNA region using a small number of samples, while NGS is better at big DNA regions, such as complete genomes.

2.1.4 Promoters

A promoter's importance is unquestionable as without it, transcription cannot occur since the enzyme known as RNAP will not be able to catalyze the reaction needed. Finding such promoter regions is non-trivial. As previously mentioned, promoters are not sequence specific, but rather a mixture of DNA sequences that recruit different 'chaperon' proteins such as TF (either by specific pairing or binding power/size) to bind to itself. We found that there are two main ways of characterizing or recognizing promoters: biological assays and computational approaches. Some biological assays use techniques such as knock-out to remove portions of DNA near the TSS of the gene until that gene stops being transcribed. This previous example is known as *promoter bashing* [11, 2]. Other techniques work by quantifying the expression of a gene [29, 28].There are genetic biological assays that are correlated to promoter recognition. These include knowledge that RNAP and other proteins need to physically interact with the DNA. By finding places in the DNA where specific molecules can bind, it makes it probable for RNAP to attach and start transcription. There are also epigenetic biological assays which assess changes in chromatin structure that, similar to the previous example, monitor DNA-protein interactions. Meanwhile, examples of computational approaches span from mathematical models to different machine learning techniques using data from previously mentioned genetic or epigenetic biological assays.

Promoters are DNA sequences that can, but do not always, span in length from about 100 to 1000 base pairs and are capable of binding RNAP for transcription [116]. DNA base pairs in promoters are designated a location relative to the TSS of the closest gene. Upstream base pairs are designated as negative values and downstream base pairs are designated as positive values, where +1 is the TSS. This means that 0 is not utilized in this numbering scheme.

There are three main regions that make up a promoter: the core promoter, distal

promoter, and proximal promoter [1]. The *core promoter* is located closest to its gene and contains general TFBS to create an RNAP binding site. The core promoter also contains the TSS. The *distal promoter* can generally be found thousands of base pairs upstream from the core promoter, making it the furthest located from the gene's TSS. The distal promoter contains several other TFBS which can recruit proteins to enhance or silence the RNAP's transcription process. The *proximal promoter* is located approximately 200 base pairs upstream from the TSS, and is the site where more specific TF bind. The length of these regions can vary, and although there is no consensus, the length of core promoters generally span from 50 to 100 base pairs [110]. There is no set range for proximal promoters, although they are generally comprised of few TFBS. In contrast, distal promoters have a wide range of 50 to 1500 base pairs [9].

As hinted previously, the variation in promoters is a barrier for researchers trying to decipher a general characterization for them. They contain various upstream elements, and can also contain downstream elements within the transcribed portion of a gene. A common misconception is that promoters can only work with one TSS. Since the TSS relates to core promoters, there are two main ways in which a RNA polymerase can bind to a core promoter and initiates the transcription of a gene—focused and dispersed. In *focused initiation*, transcription starts from a single base pair or within a cluster of several base pairs, whereas in dispersed initiation, there are several weak TSS over a broad region of about 50 to 100 base pairs. Focused initiation is the most predominant type of transcription in simpler organisms, such as prokaryotes. In contrast, *dispersed initiation* is observed in approximately two thirds of vertebrate genes, including homo sapiens. Regulated genes tend to have focused promoters, while constitutive genes typically have dispersed promoters [66].

General promoter identification, in-silico, has been researched using a variety of different techniques. In later chapters, we focus on identifying them *ab initio* or using genetic information only. More specifically, we only use DNA sequences to infer whether a specific sequence is a promoter or not. The following subsection describes other methods of identifying promoter sequences as they will be important for the future of this research area.

2.1.4.1 Promoter recognition methods

Recognition of promoter sequences require the acquisition and use of biological data. Promoter recognition methods (PRM) can be classified into three main types depending on the biological data they use. These types are ab initio, hybrid, and homology based [157].

We focus on ab initio promoter recognition methods, which only takes into consideration DNA sequence information as its input. In this method, it is assumed that all the data needed for promoter recognition lie in the genetic code. This type is the most general approach as DNA does not change through time, making results applicable to every cell in the organism and probably even in the species. Ab initio recognition can be further classified into search-by-signal, search-by-content, and search-by-structure based on modeling features.

Ab initio PRMs make use of three different types of features. The first type makes use of biological signals such as promoter TFBS, and contextual sequence information like nucleotide composition. This type is classified as a search-by-signal method. The second type were inspired by linguists and rely on sequences containing a DNA 'grammar' that is unique to promoters. DNA is split into different window sizes or *k-mers* to create words with the goal of interpreting DNA as a language. The k-mer amount that ca be used is limited by the computational complexity arising from the amount of combinations that leads to exponential growth. This second type is classified as search-by-content and are shown to be more discriminative compared to search-by-signal methods by achieving greater sensitivity and specificity. The third type takes into account the importance that DNA structure has on DNA-protein interactions. This method takes into account the flexibility, curvature, base stacking and free energy of DNA to recognize promoters. This method can become very complex when trying to obtain higher precision, making the method prone to feature errors if not considered carefully. A trade-off can be seen in the precision of the model and the scale of the DNA sequence being analysed.

Hybrid PRMs make use of features from ab initio PRMs such as DNA sequences, as well as experimental biological assay data such as gene expression, and histone modification data. These data can be obtained from methods described in section 2.1.3. These methods can perform better than ab initio methods alone, although the experimental biological data is only valid for specific cell types at a specific timeframe in their life. This caveat makes these methods usable in very specific circumstances and

thus not highly generalizable.

Homology based PRM uses the concept of regulatory regions in DNA as evolutionarily constrained. Genes were thought to be the only regions under selective pressure since they comprise the functional components of an organism. As mentioned previously, this theory has been discredited and regulatory regions have been found to be highly constrained under evolution. This means that regulatory regions must be relatively free of harmful mutations. In contrast, nonregulatory regions would not have this constraint as they are not necessary for an organism's survival. This type of PRM uses techniques such as phylogenetic footprinting to identify promoter regions of orthologous genes. This method is then constrained by having already promoter information for orthologous genes, which may not be available for new species.

2.1.4.2 Domains

DNA sequences of promoters vary widely between different organisms; they can even have significant variation within an organism. Different genes within an organism will contain differing promoter DNA sequences. Here we describe some differences between types of promoters. These types can be broadly classified by the biological domains in which the organism that contains the promoter belongs to.

Prokaryotes are simpler organisms than eukaryotes. As a result, they have been studied to a greater degree, making them better known and understood by researchers. For prokaryotic promoter recognition, we refer to the work by Busby and Ebright [14]. In a simplified manner, TF for prokaryotes are sigma factors. However, unlike eukaryotic promoters, they are better conserved and thus provide more sequence specificity. This specificity is possible because sigma factors bind to the RNAP to form a holoenzyme that makes it able to attach to DNA specific transcription initiation regions. Genes that are transcribed as a whole in contiguous groups are known as *operons*, while the ones that are transcribed in non-contiguous groups are known as *regulons*. As for genes that are regulated by the same stimulus, they are known as *stimulons*.

Prokaryotic promoters consist of two short DNA motifs at positions -10 and -35. The motif at -10 is known as the *Pribnow box*, usually consisting of six nucleotides (5′-TATAAT-3′) and is essential for the transcription process. The motif at -35 is also usually six nucleotides long (5′-TTGACA-3′) and controls the transcription rate. Knowing the sigma factor for a target promoter can simplify the search since the factor searches for a highly specific consensus binding site [12].

Eukaryotic promoters are much more diverse than prokaryotic ones. As such, they are difficult to characterize [96]. In eukaryotes, the core promoter frequently contains a motif known as the *TATA box*. The TATA box is usually a DNA sequence (5′-TATAAA-3′) where general TF and histones can bind. Other motifs located in this core promoter region include: GC-Box, CAAT-Box, TFIIB recognition element, and initiator [100]. These motifs are much less specific than the prokaryotic ones.

Promoters also differ structurally. In eukaryotes, DNA is packed into nucleosomes that block the recognition of the core promoter. Prokaryotes, in contrast, are not greatly hindered by RNAP ability to gain access to the DNA, as they do not possess this DNA packaging ability. Nucleosomes also play a role in the flexibility of DNA, since bent DNA is in reach of DNA much further away in the linear DNA sequence. This is noticeable by the presence of regulatory sites hundreds of base pairs upstream from TSS, unlike core promoters that are near the TSS. In their research, Kanhere and Bansal [67] analysed the structural properties of promoters. Their analysis indicates that special upstream features extend at least up to position -500 in the case of eukaryotic promoter sequences, but seem to be confined up to position -300 in the case of prokaryotic promoters. Both groups of eukaryotic promoters show the presence of a curved region considerably upstream of the TSS (>-200 bp); however, the prokaryotic promoters show the presence of a curved region closer to the TSS.

2.1.4.3 Eukaryotic promoter motifs

We previously mentioned the common motifs for eukaryotic promoters. In this section, we describe the regulatory elements that are part of promoters, as well as other regulatory elements found in noncoding DNA.

Regulatory elements in the genome can be found in two main categories. Cis-regulatory elements (CRE) are regions that regulate the transcription of neighbouring genes. In contrast, trans-regulatory elements (TRE) are regions that regulate the expression of distant genes. DNA sequences that act directly in the transcription process fall under the CRE classification. This includes core and proximal promoters with all their TFBS. The core and proximal promoters are more generally known together as the promoter. Like promoters, *insulators* are DNA sequences where TF can bind to, but also have the ability to affect gene expression by restricting the action of enhancers and silencers described below. Distal promoters are further from the TSS, but are still able to affect gene expression when binding to specific proteins.

Enhancers are a form of distal promoters that have been found to transcribe to long non-coding RNA or enhancer RNA that correlate to the expression levels of the target gene [71]. Enhancers can influence gene expression of a target gene when bound to TF called activators. Silencers are another form of distal promoters that bind to TF called repressors. Once bound, silencers prevent transcription of the target gene. Similar to a distal promoter, the locus control region (LCR) is a long-range CRE that enhances the expression of linked genes at distal chromatin sites. In many vertebrates, the sequence for the LCR is conserved, suggesting biological importance.

DNA sequences that encode TF are classified under TRE. TF act through an intermolecular process, meaning that they interact with DNA while being a protein themselves. In contrast, CRE act through an intramolecular process, meaning that the same type of molecule, in this case DNA, interacts with each another DNA molecule.

Promoters can contain a variety of TFBS. For more information on all the known elements in a promoter, please refer to Juven-Gershon and Kadonaga's work [66].

2.2 Machine learning

Machine learning (ML) is the science of creating algorithms that learn and improve their learning over time autonomously and without the use of human derived instructions. ML algorithms use real-world measures in the form of observations or data as experience for a task they perform. The algorithms learn when their performance at the task improves by relying on patterns from experience. In order to improve the task, the algorithms need a measure that quantifies the amount of improvement as time progresses [85]. While there are many approaches to developing learning algorithms, we focus here on supervised learning. *Supervised learning* makes use of data $(\mathbf{X}, \mathbf{Y})$ as training examples, denoting pairs of inputs and labels[1]. In a statistical sense, the goal when devising supervised learning algorithms is to create a *model* of the function f from the specific training data points $(x, y) \in (\mathbf{X}, \mathbf{Y})$, where x is the input of the function f and y is the corresponding label. The function f is a phenomenon we are trying to model, and can be measured to have a sense of the ML model's performance in approximating f. Knowing this function f we can calculate

[1]In the literature the output of a supervised ML algorithm is more commonly known as label, which correspond to real world observations.

y in the following way:

$$y = f(x) \tag{2.1}$$

This means that a supervised machine learning algorithm will learn function f_θ as model for function f that turns x into y as per Equation 2.1 for every training data point $(x, y) \in (\mathbf{X}, \mathbf{Y})$.

$$(\mathbf{X}, \mathbf{Y}) \rightarrow f_\theta \tag{2.2}$$

Equation 2.2 sketches how a supervised ML algorithm creates a model correlating the inputs $x \in \mathbf{X}$ to labels $y \in \mathbf{Y}$ through a series of mathematical transformations described by f. The ML algorithm, not shown here, provides the means of going from inputs and labels to the model. The mathematical transformations of the ML algorithm are initialized by the researcher. The ML algorithm then iteratively changes the parameters θ to create a model f_θ that approximates f. The parameter-changing process, known as *model fitting*, requires careful considerations for the model's learning performance. These changes continue until convergence ($f_\theta = f$), good approximation ($f_\theta \approx f$), or until the process is manually stopped. For each input at each step, the model outputs a label $\hat{y}$, after which the ML algorithm compares to given label y to compute the error of the model's output. Depending on the error, the algorithm will then adjust the model's parameters in order to minimize this error. Achieving an error of 0 means that the model can perfectly correlate its input to the input's paired label when using the training dataset. It is important to note that there are times when the model will not be able to achieve an error of 0. When this happens, it is a matter of preference to decide how close the error must be to 0 in order to stop the training process. Correlating the labels can thus be considered as a process of classifying an input according to its label characteristics. As such, these types of problems are known as *classification* problems.

2.2.1 Model learning

In order for an ML algorithm to construct a model successfully, we add some constraints to allow the algorithm to correctly *deduce* or generalize for inputs $x \notin \mathbf{X}$ that are not part of the training data. Let us take a particular object classification algorithm as an example. This algorithm is trained to classify dogs, and is given

input images $\mathbf{X} = X_{1,2,\ldots,i,i+1,\ldots,j}$. Samples $X_{1,2,\ldots,i} \in \mathbf{X}$ are of chihuahua dogs, and all others are of images without dogs $X_{i+1,\ldots,j} \in \mathbf{X}$. The algorithm is also given the respective dog labels $Y_{1,2,\ldots,i} \in \mathbf{Y}$ and non-dog labels $Y_{i+1,\ldots,j} \in \mathbf{Y}$. The algorithm learns to correlate its inputs to its labels by learning f_θ. When the algorithm is given a set of $x \notin \mathbf{X}$, such as husky dogs, the algorithm will likely fail to classify them as a dog. This happens because the set of dogs given as training input is a very localized subset, and does not capture the characteristics of all dog breeds. Therefore, training to distinguish images with chihuahua dogs from images without any dogs will result in a model that was not carefully designed to be generalizable, as it would not be able to separate dog pictures from pictures without dogs.

When discussing training generalization, we need to address the data being used for this training process. Researchers must ensure that the training data used is a statistically representative sample of the problem's population. Let us assume that an algorithm can perfectly learn to classify its given training data. How would we be able to give an estimate of its classification effectiveness on new or similar data, just like in the dog image classification example above? One way of calculating this estimate is by finding different sample data and testing the model on it. A problem with this method is that it can be difficult to acquire different data. A second solution that is widely used by current researchers is splitting the training data into three datasets:

- Testing
- Validation
- Training

Calculating the error of the classification model is then done in all three datasets in different stages. The method researchers use to split the data may vary between problems, as it depends on the amount of data and the model complexity. A common approach to splitting the data is to allocate 80% for the training dataset, 10% for the testing dataset and 10% for the validation dataset. This means that testing and validation datasets are usually evenly split. The training dataset is used to optimize the model's classification effectiveness by minimizing the training error. This ensures that the model will accurately classify the training data. The validation dataset is used at the end of a constant t amount of training steps. The validation error is then calculated on this dataset to estimate the model's capability to classify data ($x \notin \mathbf{X}$) that it has not been trained on. If the training error $train_{error}$ and validation error val_{error} are very small ($val_{error} \approx train_{error} \approx 0$), we can assume that the model is trained and potentially generalizable. During the training process, the researcher can recognize when both errors have reached a desirable value and stop the training

process when the model does not converge. Once the model has been trained, the testing dataset is used to get the generalization estimate for the problem's population as a whole by calculating the model's testing error. If the testing error $test_{error}$ is also very small ($test_{error} \approx 0$), we assume that the model is generalizable.

The previous methodology for training, validating and testing a model's generalization works well if the testing data used is a representative sample of the problem. For cases where it is unknown whether the sample is representative or not, there exists a method called *cross-validation* that takes all the data into account as testing data. With this method, a test dataset should still be held out for final evaluation, but the validation dataset is no longer needed. The basic approach, known as n-fold cross validation, takes the training dataset and splits it into n datasets. The process then follows:

```
1 foreach fold n_i do
2 |   model m is trained using the rest n − 1 folds;
3 |   m is validated on the current fold n_i;
4 end
```

There are many types of cross-validation approaches. The approach by Burman [13] that we used in our experiments is one that has been extended to account for imbalanced data in classification models by making folds that contain roughly the same amount data for each label. This method is called *stratified cross-validation*. It creates 'balanced' folds by sampling from the training dataset while taking into consideration their labels. Other approaches can also be found on Burman's work [13].

Training a model for a longer period of time will not always result in obtaining a better model. As described above, the model's classification error depends on the training dataset only. The purpose of a classification model is to correctly classify instances that are in the realm of the problem it was trained on. This means that we need a model that is generalizable. As training progresses, the training error will decrease, which might increase the testing error. This phenomenon is known as *overfitting*. This can happen when the learning process creates a complex function in the model that specifically maps the training dataset's input to its labels. Intuitively, one can imagine that the model is just 'memorizing' the data.

There is a subfield of methods for model generalization called *regularization*. The main idea of these methods is to penalize the model from learning very complex

functions in order to avoid overfitting. Regularization methods vary depending on the learning algorithms being used. The simplest and most common regularization technique is *early stopping*. As previously noted, the learning process can be stopped when the validation error reaches an adequate value set by the researcher. Early stopping makes this an automatic process in several ways. The simplest way is by keeping track of the validation error and ending the learning process when this error stops decreasing.

2.2.2 Deep learning

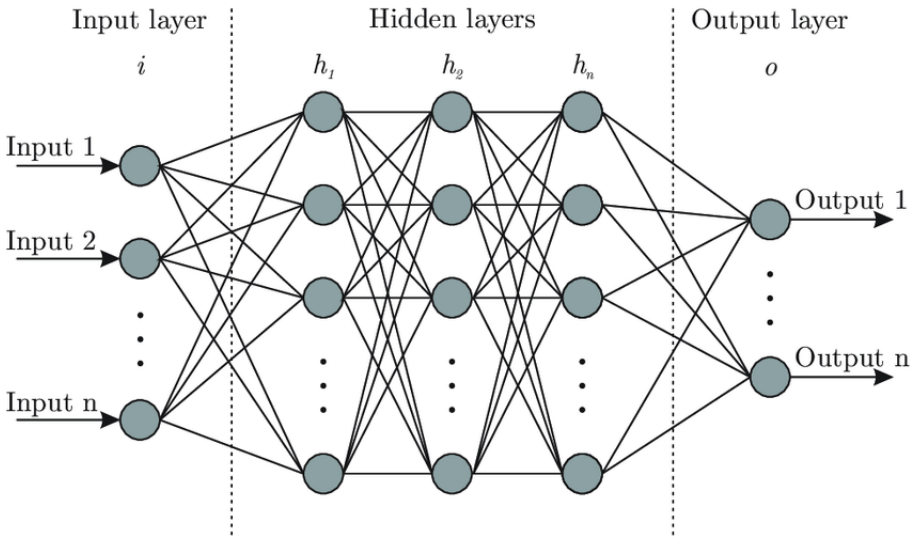


Figure 2.5: Artificial neural network architecture

Deep learning is a class of machine learning algorithms that create multi-layered *artificial neural networks* (ANN). ANN is a mathematical model loosely inspired by the neural structure of the human brain. It consists of a network of nodes or 'artificial neurons' connected to each other. Artificial neurons are the basic building blocks for ANN that transform their inputs and propagate their output to artificial neurons connected to them. A simple ANN can be seen in Figure 2.5 showing the multiple layers needed to create. ANN consist mostly of three types of layers: an input layer, an output layer, and one or more hidden layers. ANN is considered *deep* when having more than two layers of hidden neurons between the input and output layers. As the input progresses further down the network, layers transform the input into more complex and abstract objects. The layering makes deep learning effective for sophisticated nonlinear problems.

The early model of an artificial neuron is attributed to Warren McCulloch and Walter Pitts in their 1943 work [83]. The McCulloch-Pitts neural model (MPM) consists of a set of inputs $I_1, I_2, ..., I_N$ and one output $\hat{y}$. The model classifies the set of inputs into two different classes. This model can be described mathematically by:

$$Sum = \sum_{i=1}^{N} I_i W_i \tag{2.3}$$

$$\hat{y} = f(Sum) \tag{2.4}$$

where $W_1, W_2, ..., W_N$ are weight values generally in the range of either $[0, 1]$ or $[-1, 1]$ and are associated with each input I_i. These weights were set by the researchers, as there was no way of learning them through data at the time. *sum* is the weighted sum and f is a unit step function with a threshold constant T. The symbolic representation can be seen in Figure 2.6.

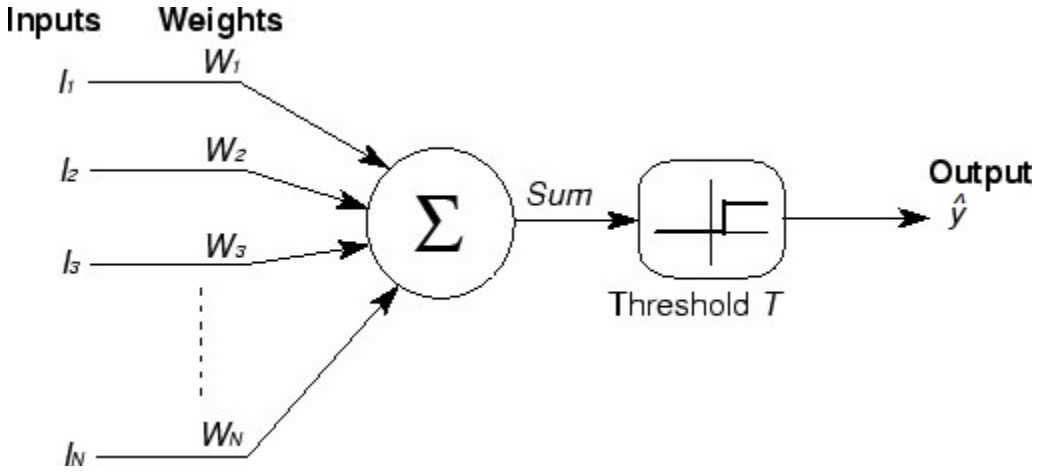


Figure 2.6: Symbolic illustration of the McCulloch-Pitts model [69]

The MPM of a neuron has substantial computing potential even though it only generates a binary output, and the weight and threshold values are fixed. Future research improved upon this by using *stochastic gradient descent* (SGD) for optimizing the weights in multiple layers, having different functions f, and interconnecting artificial neurons with other artificial neurons to obtain what we now know as an ANN.

The perceptron model [108] created by Frank Rosenblatt took MPM and expanded it such that it is able to learn the weights through a simple algorithm. Here, each input or training point is depicted as x_i as opposed to I_i from before. The weights

$\mathbf{w} = w_1, w_2, ..., w_N$ are also depicted differently from before. This algorithm can be summarized in the following steps:

```
1 Initialize w_1, w_2, ..., w_N randomly with values close to zero (e.g. [−1, 1]);
2 foreach training point x_i do
3 |   Calculate the output ŷ_i;
4 |   m Update the weights w;
5 end
```

The perceptron's output predicts a class label using a step function $f(sum)$ with the threshold constant T set to 0, meaning that one class will have an output of 1 while the other class obtains an output of -1 as in Equation 2.5.

$$f(sum) = \begin{cases} 1 & \text{if } sum \geq 0 \\ -1 & \text{otherwise} \end{cases} \tag{2.5}$$

There are multiple approaches of updating each of the weights; the earliest was done by adding a value to it. This can be written more formally as $w_i := w_i + \Delta w_i$, where the value for updating each of the weights Δw_i at each iteration i is calculated as follows:

$$\Delta w_i = \eta(y_i - \hat{y}_i)x_i \tag{2.6}$$

Here $\eta \in [0, 1] \subset \mathbb{R}$ is the learning rate constant. Note that in each iteration, all weights in weight vector $\mathbf{w}$ are updated. The perceptron was a great advancement in learning algorithms, but it came with drawbacks. Using Rosenblatt's algorithm, perceptrons could not be stacked together as ANN as we know today. This means that the learned function f would be limited to use in linearly separable data - a fact proven by Rosenblatt - as the perceptron would only converge in this scenario [108].

Adaptive linear neurons or ADALINE by Widrow [150] were the next step after the perceptron. As explained by Widrow, ADALINE are the building blocks of his neural network model that have the ability to combine with one another to create a network. His work introduced the delta rule, also known as the least mean square (LMS) algorithm, which updates the weights of a single ADALINE based on a linear activation function rather than the step function. This algorithm minimizes the sum of squares of the linear errors defined to be the difference between a desired response or output $\hat{y}$ and a label y. ADALINE is still a linear classifier, much like the

perceptron. To solve this, Widrow created a system called *MADALINE*, which was able to support multiple adaptive elements in succession of each other, much like layers in ANNs. There are three different rules to update the weights for MADALINE, each improving upon the previous rule. MADALINE rule I could not adapt the weights from the middle layer connecting to the output layer. MADALINE rule II made the model able to learn all weights in the network. MADALINE rule III achieved the ability to learn nonlinear activation functions, more specifically the sigmoid function. This last rule was found to be equivalent to backpropagation, which is described further below.

The advantage of nonlinear activation functions is their continuous property, making them differentiable, unlike the step function. This property allows researchers to define a differentiable cost function $cost(\mathbf{w})$ that quantifies the error between a model's output and its corresponding label. Minimizing this function by modifying the weights would result in reducing the classification errors made by the model.

The LMS algorithm makes use of the *sum of squared errors* (SSE) cost function. Here we give an example of this algorithm while presenting the SSE:

$$cost(\mathbf{w}) = \frac{1}{2}\sum_i (y_i - \hat{y_i})^2 \qquad , \hat{y_i} \in \mathbb{R} \tag{2.7}$$

As we can see from this cost function, a significant difference between the perceptron and ADALINE/MADALINE is that the output of the model changes from an integer to a real number. This is important because the algorithm for minimizing cost functions, known as gradient descent, makes use of these continuous and differentiable properties of cost functions. We illustrate how gradient descent makes uses of these properties along with the convexity of the SSE cost function in Figure 2.7 to minimize the cost function.

Gradient descent is an iterative algorithm where, at each step, every weight is updated by taking a step towards the opposite direction of the gradient until a local or global minimum is reached. This update can be formally written as $\mathbf{w} := \mathbf{w} + \Delta\mathbf{w}$ where

$$\Delta\mathbf{w} = -\eta\nabla cost(\mathbf{w}) \tag{2.8}$$

The step size is determined by the learning rate η and the slope of the gradient $\nabla cost(\mathbf{w})$. To obtain the gradient, we have to compute the partial derivative of the

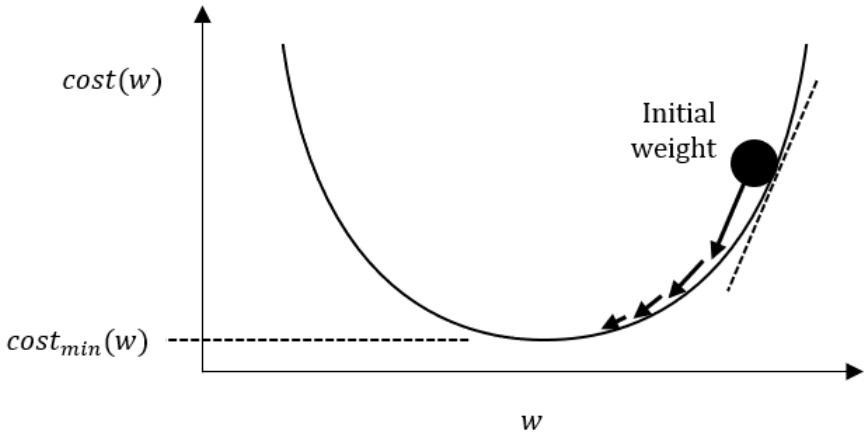


Figure 2.7: Gradient descent schematic

cost function for each weight $\Delta w_j = -\eta \frac{\delta cost}{\delta w_j}$ of each layer. Calculating the partial derivative for the weights, we get

$$\Delta w_j = -\eta \frac{\partial cost}{\partial w_j} = -\eta \sum_i (y^{(i)} - \hat{y}^{(i)})(-x_j^{(i)}) = \eta \sum_i (y^{(i)} - \hat{y}^{(i)}) x_j^{(i)} \tag{2.9}$$

for a sample point i in the training data. This learning rule looks identical to the perceptron learning rule with the exception of the sum. The sum calculates the weight updates from all samples in the training set, making it *batch* gradient descent. Conversely, calculating the weight updates from a single sample point is called *stochastic* gradient descent. This powerful training mechanism made ADALINE set the foundation for the current ANN models described in the following sections.

ANN are powerful algorithms based on having multiple layers of interconnected nonlinear artificial neurons. The function f that the ANN is trying to approximate is multivariate, and can be complex depending on the architecture. As the complexity of the cost function grows, so does the complexity for calculating the gradients using differential calculus. The architectural complexity of deep neural networks makes deriving the gradient too time consuming for naïve hand-coded derivation. There are three main methods of calculating the derivative of a function: numerical, symbolic, and automatic.

Numerical differentiation relies on the definition of the derivative from limit the-

ory:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \tag{2.10}$$

where setting h to a very small number and evaluating the function will give an approximation of the derivative. This method is the most basic as it is only an approximation, and thus inaccurate. It is used when the function is too complex for other methods, or when the function is not known and sampling from the function is the only way to learn about the function.

Symbolic differentiation manipulates mathematical expressions. It is similar to the manual pen-and-paper method where various rules such as product rule and chain rule, are used to calculate the resulting derivative. This method also relies on simplification methods to aid in the derivation process.

Automatic differentiation (AD) is similar to symbolic differentiation. The main difference is that the automatic method manipulates blocks of computer programs. The derivation rules will be set beforehand for each element of the program, and the chain rule will be used to divide complex expressions. This method was popularized for its use in ANN by Rumelhart et al. [111], in their work "Learning internal representations by error propagation," where they empirically demonstrated the the use of *Backward propagation of errors* (Backpropagation) in a multi-layer ANN architecture. Backpropagation is an ANN learning algorithm, and is a special case of AD known as reverse mode AD.

The gradients for the cost function in ANN can be represented in a matrix. Implementations for calculating the derivatives for each of the gradients creates the Jacobian matrix, a matrix with all the first partial derivatives of a function. As explained before, ANN are a composition of differentiable functions, making the Jacobian very useful for minimizing the cost function. The derivatives tells us what we want to get, but not how to compute it efficiently. AD is a method of getting these gradients efficiently by only writing the cost function in computer code. There are two main methods of AD: forward and reverse mode.

The idea of forward mode AD is to compute derivatives as it computes the forward pass. This method makes use of the chain rule property, namely that the overall derivative of the cost function is a composition of incremental computations from the forward pass. Every run of this method can only get the derivative of one of the inputs, since all intermediate partial derivatives are computed with respect to an

input x_1, calculated as $\dot{v_i} = \frac{\delta v_i}{\delta x_1}$.

The idea of reverse mode AD is to compute the output of the function while storing information about the structure of the computation graph and the intermediate variables that were computed. Reverse mode AD then walks backwards through the graph and computes derivatives of the output with respect to each of the local variables. In contrast to forward mode AD, this method computes the derivatives for all inputs at the same run as the intermediate partial derivatives are computed with respect to a local variable v_i, calculated as $\bar{v_i} = \frac{\delta y}{\delta v_i}$ by using the previously calculated output y.

To understand backpropagation, we first have to understand how ANN outputs a value $\hat{y}$ from its input by a process called *forward propagation*. Given $\hat{y}$, the error or cost function $cost(\mathbf{w})$ can then be calculated and used to train the ANN. Training occurs by modifying the weights $\mathbf{w}$ in order to minimize the cost function as depicted in Figure 2.7. The modification of the weights require the calculation of the gradients by backpropagation that are then used in optimization algorithms such as gradient descent to minimize the cost function. There are three main implementations to the backpropagation computation abstraction: source code transformation, graph-based, and tape-based.

The source code transformation (SCT) approach uses the source code of the program as a model to perform AD and obtain the gradients and function evaluations. SCT makes it possible to compute the derivatives before runtime. An example of an SCT approach is Tangent[2]. The graph-based approach uses an embedded language to specify a graph of computations that can then be manipulated to obtain function evaluations and gradients. Graph-based approaches can also compute the derivatives before runtime by using its embedded graph language. The advantage of using such a language is that it can make use of the many graph and compiler optimization algorithms available. An example of a graph-based approach is Tensorflow[3]. The tape-based approach tracks the sequence of computations during forward propagation, obtaining the function evaluations using a list or similar data structures. The list will make it possible for the approach to walk backward and compute the gradients. The advantage of this approach is that it does not require any external languages or processes, making debugging easier to manage as the backpropagation computations

[2] `https://github.com/google/tangent`

[3] `https://github.com/tensorflow/tensorflow`

happen at runtime. An example of a tape-based approach is Pytorch[4].

To reinforce the understanding backpropagation, we will go over an example on a simple neuron with two inputs and a sigmoid activation function. Activation functions will be described in more detail in section 2.2.3.1. More formally, the function that we will explore is

$$s(\mathbf{w}, \mathbf{x}) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \tag{2.11}$$

Here, the inputs $\mathbf{x} = \{x_0, x_1\}$ and weights $\mathbf{w} = \{w_0, w_1, w_2\}$ are sets that can be represented as matrices. Before we can do backpropagation, we need our derivation rules:

$$\begin{aligned} f(x) = e^x &\longrightarrow \frac{\delta f}{\delta x} = e^x & \qquad f(x) = \frac{1}{x} &\longrightarrow \frac{\delta f}{\delta x} = \frac{1}{x^2} \\ f_a(x) = ax &\longrightarrow \frac{\delta f}{\delta x} = a & \qquad f_c(x) = c + x &\longrightarrow \frac{\delta f}{\delta x} = 1 \end{aligned} \tag{2.12}$$

The first step is to do forward propagation to obtain the function evaluations or outputs, with inputs $\mathbf{w}$ and $\mathbf{x}$. In practical setting with ANNs, $\mathbf{w}$ is initialized randomly according to some distribution, while $\mathbf{x}$ is the data input. In this example we give the following inputs:

$$\begin{aligned} x_0 &= -1.0 \\ x_1 &= -2.0 \end{aligned} \qquad \begin{aligned} w_0 &= 2.0 \\ w_1 &= -3.0 \\ w_2 &= -3.0 \end{aligned}$$

Forward propagation is simply evaluating the function. Evaluating the function in the standard order of operations produces a computation graph, as depicted in Figure 2.8. Once the forward propagation has been completed and the values v_i have been obtained, we can use the output of the operations to obtain the gradients according to the derivation rules. This process traverses the computation graph in the opposite order as when it was built by forward propagation. The first step is then to compute the gradient for the output, which is $\bar{v}_{13} = \frac{\delta v_{13}}{\delta v_{13}} = 1.0$. After this, we traverse the computation graph until reaching the inputs using our derivation

[4] `https://github.com/pytorch/pytorch`

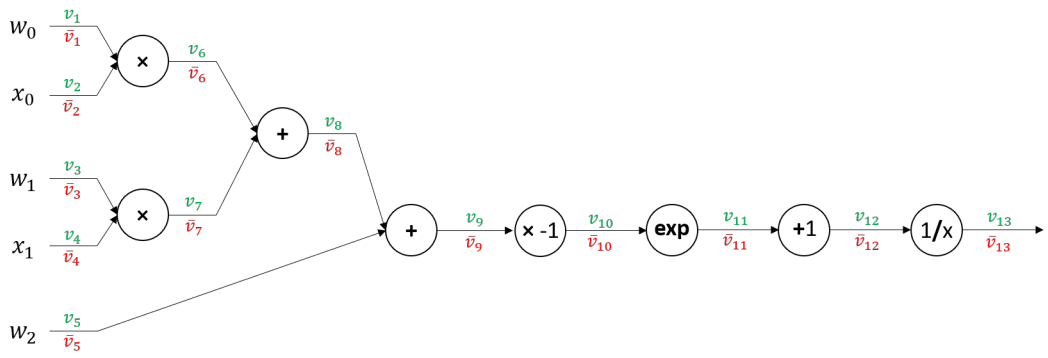


Figure 2.8: Computation graph visual aid for backpropagation

rules from Equation 2.12. As per chain rule, combining our gradients is a matter of multiplying the local gradient and the upstream gradient. For example, to obtain the gradient of the second to last operation $\bar{v}_{12}$, we multiply the upstream gradient $\bar{v}_{13} = 1.0$ with the local gradient $\frac{\delta f}{\delta v_{12}} = \frac{-1}{(v_{12})^2}$, where $f(x) = \frac{1}{x}$ and $v_{12} = 1.37$. The end result will be the gradients for each operation as depicted in Figure 2.9 in color red. The green values of the figure are the results of forward propagation.

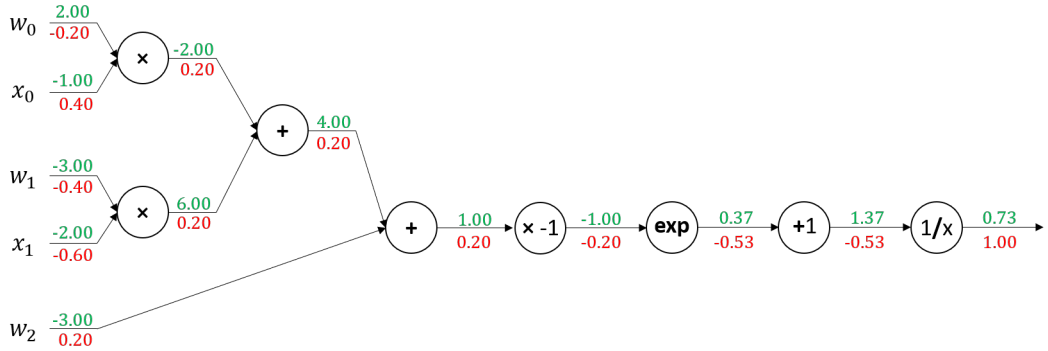


Figure 2.9: Evaluation of forward and backward propagation

2.2.3 Artificial Neural Networks

The neurons in ANNs can have many different arrangements. Neurons themselves can be interconnected in many different ways and have various activation functions. Current research focuses on choosing an appropriate architecture for learning various tasks. Having a general architecture that can learn any task is the end goal, but so far this has not been attained. ANNs have artificial neurons as their building blocks, but there are many more intricacies that make up an ANN. Artificial neurons

are comprised of an aggregation of inputs with their weights followed by a nonlinear function that outputs a value. When connecting artificial neurons together by using their outputs as other neurons' inputs, they can form neural network layers, which in turn make up the complete neural network architecture. Artificial neurons that are connected for a specific purpose are called *neural network cells*, and can be part of neural network layers.

2.2.3.1 Neural Networks building blocks

The features that are needed as parts of ANNs include activation functions, loss functions, optimizers, and regularization methods. These are described in the following subsections.

Activation Functions

As previously seen with ADALINE, the linear activation function before the output in each of the artificial neurons made it possible for the network to learn using backpropagation. Activation functions could be further improved by making them nonlinear while retaining their differentiable property. There are many different activation functions in use nowadays in the literature, and their differences lie mostly in their range of output values and their gradients. Although the specific choice of activation functions can affect a neural network's performance, Hornik [57] showed in his 1991 work that it is not the specific choice of the activation function, but rather the multi-layer architecture itself, which gives neural networks the potential of being universal approximators. In this section, we only present the nonlinear activation functions that were part of our deep learning experiments.

Sigmoid

The sigmoid function takes any real value $z \in \mathbb{R}$ as input and outputs a value in the range $[0, 1]$. It normalizes any input to a value between 0 and 1. It is most widely used as a classifier function by using a threshold to separate both classes. The sigmoid function is defined and its derivative visualized in Figure 2.10. We can notice that the gradients at both ends of the function tend to go to zero, making the *objective function* - any function that is optimized during training - minimize very slowly. This 'vanishing gradient' problem makes the training process very slow.

Softmax

The $softmax$ function is useful for multiple artificial neurons as a way of calculating the probability distribution of each target label or class over all the classes,

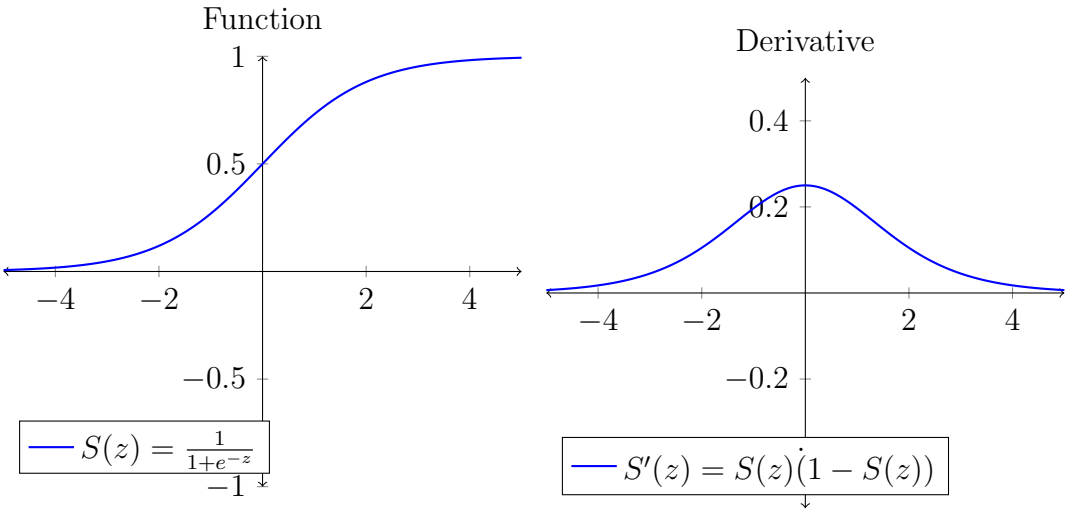


Figure 2.10: *sigmoid* function and its derivative

shown in Equation 2.13. More intuitively, it can be seen as a 'soft' or smooth version of the *max* function that is normalized to a range between 0 and 1. In this case, any real value input $z \in \mathbb{R}$ will be transformed into the range $[0, 1]$, where the sum of all outputs will be equal to one, thus making it a probability distribution. This function is similar to the *sigmoid* function for binary classification, where the threshold divides the classes. Softmax is typically used for the output layer of neural networks in classification tasks. An important note is that this function works with multiple artificial neurons, as most other activation functions work on single neurons.

$$S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}} \tag{2.13}$$

Tanh

The hyperbolic tangent or tanh activation function is very similar to the sigmoid function. It takes a real value $z \in \mathbb{R}$ and it outputs a value in the range $[-1, 1]$. Tanh is a mathematically shifted version of the sigmoid function. As such, each function can be derived from the other, shown in Equation 2.14. Apart from the different range and a steeper gradient, the main difference is that tanh is zero centered. As for the similarities, the tanh function also suffers from the 'vanishing gradient' problem. For these reasons, tanh is preferably used in hidden layers, while sigmoid is used mostly in the output layer as a classifier function. The tanh function and its derivative is

visualized in Figure 2.11.

$$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} = \frac{2}{1 + e^{-2x}} - 1 = 2 * S(2z) - 1 \tag{2.14}$$

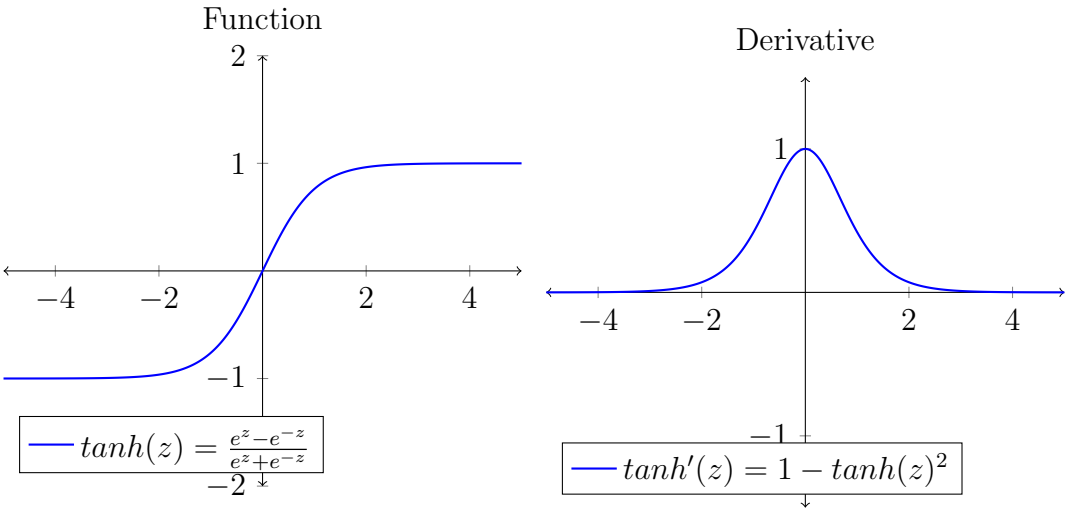


Figure 2.11: *tanh* function and its derivative

ReLU

ReLU stands for rectified linear unit and is currently the most common activation used in the hidden layers of neural networks. This function takes any real value $z \in \mathbb{R}$ and outputs the same value if it is greater than zero; otherwise, it outputs a zero. This is commonly used because of its simplicity and computational performance compared to the other activation functions. It makes the neural network sparse because only a few artificial neurons would be activated, adding to the efficiency of the learning process. There is a problem known as the dying ReLU, which is easily visible in the visualization of the function and its derivative below. This problem occurs when the function becomes negative or zero, making the gradient zero. In this case, backpropagation stops and the artificial neurons will stop responding to variations in error. In other words they are becoming 'dead' neurons. Another problem with ReLU's range of $[0, \infty)$ is that its output can create computational instability as memory overflows might happen, known as 'exploding' gradients. ReLU is formally defined and visualized in Figure 2.12.

ELU

The exponential linear unit or ELU is an activation function similar to ReLU. Different from other activation functions, it contains an extra constant $\alpha > 0$. The

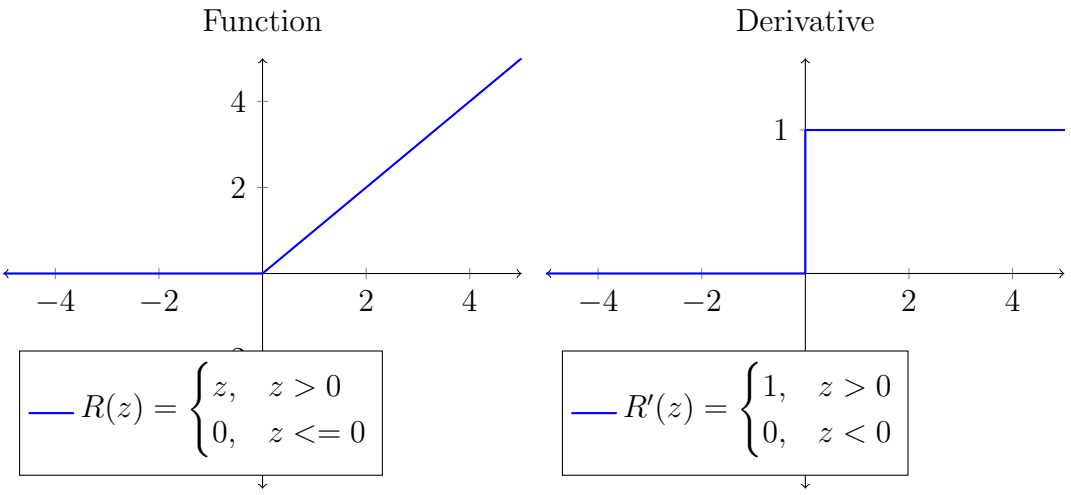


Figure 2.12: *ReLU* function and its derivative

exponential term in this function makes it more computationally expensive than the ReLU function, but making up for it in the possibility of having negative values, and thus no 'dead' neurons. The constant α is commonly set to a very small value such as 0.01 for a smooth function. Similar to ReLU, ELU has the 'exploding' gradients problem too because of the identity portion of the function. These properties make ELU very useful in hidden layers, but seldom anywhere else. The ELU function is formally defined and visualized in Figure 2.13.

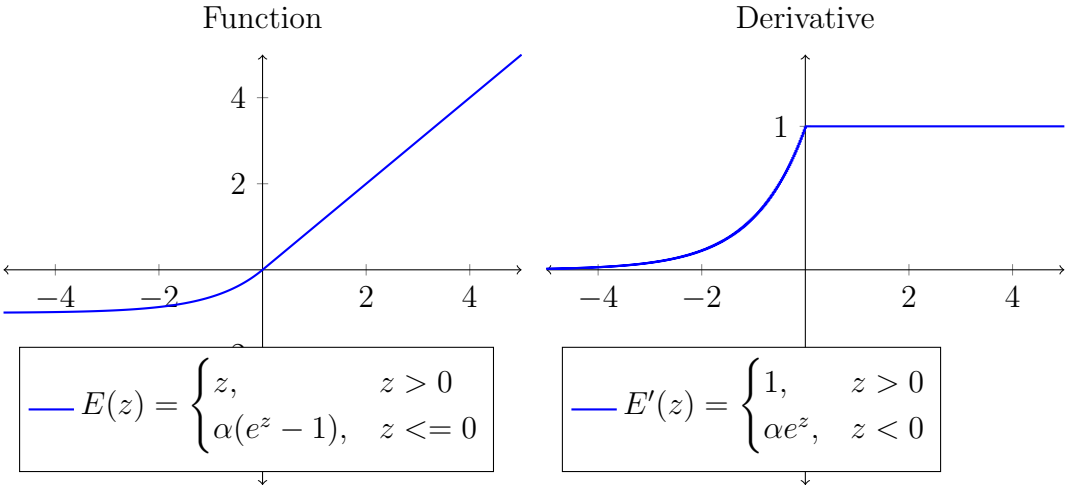


Figure 2.13: *ELU* function and its derivative

Loss Functions

Above, when talking about the error of neural networks, we describe it using cost

functions. At every training step, we must compute the error that the model has when classifying the data on the labels. Computing an error for a single training point is done through a *loss function*. The cost function is the aggregate of the loss functions for the entire training dataset. For example, we talked previously about the SSE cost function in Equation 2.7. The loss function related to SSE is the square loss given by

$$l(f(x_i|\mathbf{w}), y_i) = (f(x_i|\mathbf{w}) - y_i)^2 \tag{2.15}$$

where $f(x_i|\mathbf{w}) = \hat{y_i}$ is the output from the neural network model. Equation 2.15 depicts how the size and types of layers, and generally the architecture affecting the model's output, as well as how we measure the error for training the model, are all part of the loss function. As such, there are many different cost functions possible. Here we will describe the most common ones being used throughout the literature. Keep in mind that regularization is also part of the loss function as it affects the model f, but is usually treated separately as the regularization term can be simply added to the loss. Notably in the machine learning community, many researchers use the terms objective function, cost function, and loss function interchangeably. In deep learning frameworks, the term loss function is used. However, to avoid confusion, we will refer to cost functions as loss functions from now on in this thesis. Loss functions in deep learning also have the property of having a minimum value of zero, making the goal of the learning algorithms to minimize them.

The previous simple loss function examples are regarded as regression loss functions. Loss functions can be grouped into two main types depending on the learning task: regression and classification. Regression is used when the model learns a continuous real value from an infinite set of values, while classification is used when the model learns discrete integer values from a finite set of categories. Categorical loss functions have an extra step from regression loss function, needing to compute a score s for each possible label or class. The true label for a training point is usually denoted as $t_i = y_i$, the true label y_i we have described before. Our focused task regards to promoter classification, and thus we only describe common classification loss functions.

Cross-Entropy Loss

The logistic loss or cross entropy (CE) loss is the most common categorical loss function. The formula for CE loss is shown in Equation 2.16. This function comes

from entropy in information theory and from log likelihood in probability theory. The most important property of this function is that it heavily penalizes the classification predictions that are confidently misclassified.

$$-\sum_{i}^{Classes} t_i log(s_i) \tag{2.16}$$

Binary Cross-Entropy Loss

Binary CE loss is the special case of CE loss where there are only two classes, and can be defined as in Equation 2.17. This formula assumes two classes: C_1 with the true label $t_1 \in \mathbb{Z}[0,1]$ and score s_1, and C_2 with the true label $t_2 = 1 - t_1$ and score $s_2 = 1 - s_1$.

$$-\sum_{i=1}^{Classes=2} t_i log(f(s_i)) = -t_1 log(f(s_1)) - (1 - t_1) log(1 - f(s_1)) \tag{2.17}$$

In neural networks, BCE loss is also known as Sigmoid CE loss because it makes use of a sigmoid activation function $f(s_i)$ followed by the CE loss. This is done because of its computational efficiency when doing backpropagation by avoiding re-calculations.

Categorical Cross-Entropy Loss

Categorical CE loss is the multi-class classification case for CE loss defined in Equation 2.18. Since we are doing multiple classifications with this loss function, the true label or target is now a vector $\mathbf{t}$ where only one element is nonzero. Every element in the vector denotes a class and the nonzero element will be the true label in the training point denoted $t_i = t_p$.

$$-\sum_{i}^{C} t_i log(f(s_i)) \tag{2.18}$$

The formula above is a CE loss function with a softmax function on the scores, and $C = \{c_1, c_2, ..., c_m\}$ refers to the set of all possible classes that are part of the training data. Rewriting the softmax formula with the classification score notation, we get $f(s_i) = \frac{e^{s_i}}{\sum_j^C e^{s_j}}$. When combined, we get Equation 2.19, where s_p is the score for the positive label t_p.

$$-log\left(\frac{e^{s_p}}{\sum_j^C e^{s_j}}\right) \tag{2.19}$$

Other categorical loss functions

$$\sum_{j \neq y_i} max(0, s_j - s_{y_i} + 1) \tag{2.20}$$

There exist other categorical loss functions not covered here including hinge, focal, and KL divergence. The hinge loss comes from another type of machine learning algorithms called support vector machines, and thus also known as SVM loss. This loss tries to obtain the score of the correct category to be greater than the sum of scores of all the incorrect categories by a safety margin (usually one). Its formula is shown in Equation 2.20, where each of the label scores is denoted by s.

$$-\sum_{i=1}^{C}(1 - s_i)^{\gamma} t_i log(s_i) \tag{2.21}$$

Focal loss is a relaxation of the cross entropy loss shown in Equation 2.21. It weighs the contribution of each sample to the loss based on the classification error. Samples that are classified correctly will be weighed less, and vice versa. The problem they are trying to resolve with this loss is that of class imbalance by making the loss implicitly focus in the misclassified samples.

$$\mathbf{KL}(\mathbf{p}||\mathbf{q}_\theta) = \sum_{i=1}^{N} p(x_i) \cdot log\left(\frac{p(x_i)}{q(x_i)}\right) \tag{2.22}$$

The Kullback-Leibler (KL) divergence is a loss function for comparing two distributions shown in Equation 2.22. KL divergence is often used to approximate a simpler distribution to a more complex one by measuring how much information is lost in the approximation. It is called a divergence because the comparison is not symmetrical as in the case of distances. This loss is used in Generative Adversarial Networks, a type of neural network architecture. Therefore, there are two methods for fitting the model to a distribution, in this case fitting q_θ to p:

- $\mathbf{KL}(\mathbf{p}||\mathbf{q}_\theta)$
  - Requires normalization with respect to p (often computationally expensive)
  - mean-seeking, inclusive. Intuitively, it spread the mass of $q(x)$ everywhere that $p(x)$ has some mass, trying to fit the whole distribution.
- $\mathbf{KL}(\mathbf{q}_\theta||\mathbf{p})$

- Does not require normalization with respect to p
- mode-seeking, exclusive. Intuitively, $q(x)$ can focus on a mode from $p(x)$ instead of the whole distribution.

Optimizers

Obtaining the gradients needed to train the network is done through backpropagation. These gradients are used to update the weights or parameters of the network. Previously, we talked about the most common way to update the weights when dealing with the perceptron and ADALINE - using gradient descent - but there are many other ways to update them. These methods of updating weights are known as *optimizers*, but most optimizers are based on gradient descent. We will describe some other common optimizers used in the literature which adaptively select the learning rate for a faster learning process.

Momentum

Momentum is very similar to gradient descent and is shown in Equation 2.23. The main difference is that it adds a velocity variable that helps accelerate gradient descent. Updating the weights takes into consideration the gradient of the current step as well as the gradient of the previous time steps. This makes the weight updates move faster towards convergence.

$$\begin{aligned} v_t &= \gamma v_{t-1} + \eta \nabla cost(\mathbf{w}) \\ \mathbf{w} &= \mathbf{w} - v_t \end{aligned} \tag{2.23}$$

Adagrad

Adaptive Gradient Algorithm, or Adagrad, improved on optimizers by adapting the learning rate to the weights $\mathbf{w}$. The idea is to perform larger updates for infrequently used weights and smaller updates for frequently used weights. It was created for a sparse training data in mind, such as in embedding layers in recurrent neural networks seen in the Recurrent Neural Networks subsection of section 2.2.3.2. In gradient descent and momentum, all the weights $\mathbf{w}$ are updated the same way. In Adagrad, there is a different learning rate for every weight $w_j \in \mathbf{w}$ for every training step t.

Adagrad's weight updates are formally written as

$$\begin{aligned} g_0 &= 0 \\ g_t &= g_{t-1} + \nabla cost(\mathbf{w})^2 \\ w_j &= w_j - \eta \frac{\nabla cost(w_j)}{\sqrt{g_t} + \epsilon} \end{aligned} \tag{2.24}$$

where g_t is the sum of the squares of past gradients with respect to all parameters $\mathbf{w}$, and ϵ is a constant close to zero (i.e., $\epsilon = 1e^{-5}$). We divide the current gradient by the past gradients, and this iterative process will make the sum of gradients become larger as the training steps increase. This means that the update will get too small to be useful in long training sessions.

RMSProp

RMSProp is a slight variation of Adagrad that addresses its problem. RMSProp keeps the running sum of squared gradients but with an added variable α to let the sum decay. The weight update keeps being the same as Adagrad, and is written formally as,

$$\begin{aligned} g_0 &= 0 \\ g_t &= \alpha \cdot g_{t-1} + (1 - \alpha) \nabla cost(\mathbf{w})^2 \\ w_j &= w_j - \eta \frac{\nabla cost(w_j)}{\sqrt{g_t} + \epsilon} \end{aligned} \tag{2.25}$$

where the decay rate is initialized to approximately 0.9 ($\alpha \approx 0.9$), and epsilon is a constant close to zero as in Adagrad.

Adam

So far we have seen a method that builds up the velocity of the gradient to update the weight as a way to move faster towards convergence, and a method that adapts the learning rate to the weights of the model. Adaptive Moment Estimation, or Adam, takes these two methods and merges them to make an optimizer. Adam's weight updates are as follows:

$$\text{Momentum} \quad \boxed{m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla cost(\mathbf{w})} \tag{2.26}$$

$$\text{RMSProp} \quad \boxed{v_t = \beta_2 v_{t-1} + (1-\beta_2)\nabla cost(\mathbf{w})^2} \tag{2.27}$$

$$\text{Bias Correction} \quad \boxed{\begin{aligned} \hat{m}_t &= \frac{m_t}{1-\beta_1^{t-1}} \\ \hat{v}_t &= \frac{v_t}{1-\beta_2^{t-1}} \end{aligned}} \tag{2.28}$$

$$\text{RMSProp + Momentum} \quad \boxed{\mathbf{w} = \mathbf{w} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}} \tag{2.29}$$

We have separated the individual portions of Adam to see how the combination of Momentum and RMSProp work together. Equation 2.26 changed the velocity variable to m and added a friction variable β_1. Equation 2.27 is the same as RMSProp's g_t equation, changing α for β_2. The idea behind the bias correction equations 2.28 concerns the initialization variables, $m_0 = 0$ and $v_0 = 0$, where it makes the first update very large as a consequence of this zero initialization. It also takes into consideration β_1^{t-1} and β_2^{t-1} from the previous time step as denoted by their subscript. Finally, Equation 2.29 is the combination of the previous equations to update the weights.

Regularization

Previously, we mentioned the most common form of regularization known as early stopping. Recall that regularization methods are used to prevent overfitting. As such, regularization can span over a wide range of methods. Here we will describe common regularization methods including L1 and L2 regularization, dropout, and data augmentation. These techniques can be found in more detail in a survey by Kukacka et al. [72]. More information on their application and empirical results in a similar biomedical domain can be found in work by Sato et al. [113].

L1 regularization

L1 regularization is implemented in the loss function. It is used in regression loss functions and when used it is called Lasso regression. It updates the loss function by adding a regularization term that penalizes large weights. The reason for penalizing large weights is because this leads to an increase to the model complexity. The smaller the weights, the simpler the model, which reduces overfitting. The regularization term that is added to the loss function is

$$\lambda \sum_{i}^{N} |w_i| \tag{2.30}$$

where N is the number of elements in the set of weights $\mathbf{w} = \{w_1, w_2, ..., w_N\}$, and λ is the regularization term.

L2 regularization

L2 regularization is also implemented in the loss function. When used in a regression loss function, it is called Ridge regression. Like the L1 regularization, it updates the loss function by adding a regularization term for penalizing large weights. The regularization term used is different to the L1 regression case, making it penalize the weights differently. The regularization term added to the loss function is

$$\lambda \sum_{i}^{N} w_i^2 \tag{2.31}$$

where N is the number of elements in the set of weights $\mathbf{w} = \{w_1, w_2, ..., w_N\}$, and λ is the regularization term.

Dropout

Dropout is implemented during the training process. It is a very unique form of regularization as it involves essentially shutting down artificial neurons in the neural network. At every iteration, dropout randomly selects a percentage of artificial neurons and turns their weight to zero, essentially removing it from the neural network. This makes it so that each iteration has a different set of artificial neurons, resulting in a different set of outputs every step. It can be thought of as an ensemble method in machine learning, where multiple smaller models' outputs are aggregated, creating an overall more robust model. Dropout requires a probability parameter for selecting the amount of artificial neurons to shut down on every iteration.

Data augmentation

This regularization method is implemented in the preprocessing step of machine learning. It concerns the improvement of the data being used by getting a more representative sample to describe the problem. In some cases, obtaining more representative data can be challenging, but in the cases where modifying the current data in an algorithmic way creates more sample points for the same data, this can be a great tool. An example of such type of data is images, where manipulating them in different ways can create more data samples for a more robust model. Manipulations for image data can include rotation, flipping, scaling, and shifting.

2.2.3.2 Common Architectures

There are many supervised deep learning architectures, but the following architectures are the most commonly used due to their general applicability in varying fields and their state of the art performance for learning to solve problems. These architectures have been empirically shown to do well in certain problems. In this subsection, we describe the three main architectures currently being used in the literature. These architectures are made available in ML frameworks to be used in any combination with other layers to form novel architectures.

There are many more neural network architectures that are being created by researchers. Most other NN architectures are a combination of the three architectures we present here, but there are completely different and novel architectures too. In his two posts, Veen [139, 140] made a short but interesting review about many deep learning architectures and how they work.

Feedforward Neural Networks

A feedfoward neural network or multi-layer perceptron (MLP) is the most popular type of ANN. Its architecture consists of a set of layers connected one after the other from the input to the output layer, depicted in Figure 2.5. The most common form of an MLP is having every neuron from one layer being connected to every neuron from the subsequent layer, although there are many other ways to setup the connectivity between neurons. When every artificial neuron from a layer is connected to every artificial neuron in the subsequent layer, it is known as a *fully connected layer*. A characterizing factor of an MLP is its association to directed acyclic graphs describing the layers from input to output.

When we introduced the history of deep learning in section 2.2.2, we talked about Rumelhart's work in ANNs from the 1980s. MLPs inner workings have not changed since then, although there are two main differences since that have taken place since then. The first difference is the increase in computational power; the second is the amount of data available for use in the training process. Together, these two advances have increased the capacity of deep learning models to statistically generalize in a practical amount of time. Current MLPs are used as tools to develop probabilistic models that need to be carefully engineered. When done right, MLPs perform very well, but the difficulty in network design and current optimization algorithms make it an architecture of unfulfilled potential [49].

Convolutional Neural Networks

A convolutional neural network (CNN) is an ANN that was specially created for processing grid-like data. Most of its success has been achieved in the computer vision field. CNNs functionality has been observed in neural experiments involving the primary visual cortex. Neurophysiologists David Hubel and Torsten Wiesel experimented on cats by showing them images of simple shapes, e.g. lines, squares, and circles, and noting how the neurons reacted. Their experiments showed the principles of how a visual system functions by extracting more complex features at every step of the brain's neural network [59].

A CNN is a special type of MLP that makes explicit assumptions about their input, and has partial connectivity of their artificial neurons. CNNs get their name from their mathematical operation, *convolution*. A CNN is a neural network with at least one convolution operation in its architecture. Convolution is a kind of linear operation commonly denoted as an asterisk $*$, as in the following generic example,

$$conv(t) = (x * w)(t) \tag{2.32}$$

In CNN terminology, x is referred to as the input, and w as the kernel or filter. t is an integer time variable, and we assume that x and w are defined only at time t to get the discrete convolution,

$$conv(t) = \sum_{a} x(a)w(t-a) \tag{2.33}$$

This discrete convolution operation is defined for one-dimensional input and kernel. The convolution operation can be extended to any amount of dimensions, but the most common is the two-dimensional case which is defined in the following equation,

$$conv(i,j) = (X * K)(i,j) = \sum_{m}\sum_{n} X(m,n)K(i-m)(j-n) \tag{2.34}$$

where X and K are two-dimensional, like in the case of an image. Convolution is commutative by flipping the kernel relative to the input like the following,

$$conv(i,j) = (K * X)(i,j) = \sum_{m}\sum_{n} X(i-m)(j-n)K(m,n) \tag{2.35}$$

Cross-correlation is the same operation as convolution, but without flipping the kernel. This can be seen in Equation 2.36. A confusing part of many machine learning

frameworks lies in them implementing cross-correlation while calling it convolution.

$$C(i,j) = (X * K)(i,j) = \sum_m \sum_n X(i+m)(j+n)K(m,n) \tag{2.36}$$

Another common operation that is used in CNNs after the convolution operation is *pooling*. This pooling operation outputs a summary statistic of similar values in a grid. The are three common types of pooling: max, min, average. *Max pooling* outputs the maximum value within a rectangular neighborhood of values. *Min pooling* outputs the minimum value, and *Average pooling* outputs the mean of the rectangular neighboring values. The pooling operation has had many practical successes, but is still a target of heavy criticism within the scientific community [74].

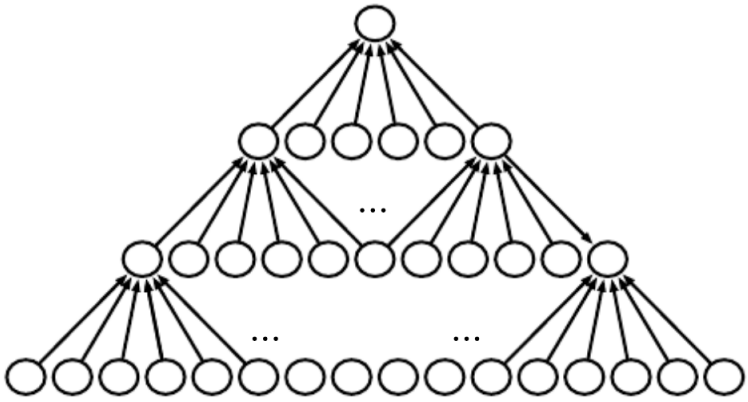


Figure 2.14: Depiction of the convolution operation on a neural network showing sharing of parameters. Artificial neurons are depicted as circles. Parameters are depicted as the lines going from neuron to neuron. The parameters shift places, so all the groups of six parameters are the same parameters, just shifted to another place.

In CNNs, the convolution operation replaces the linear operation used in MLPs (Equation 2.3). Instead of fully connected layers, it uses parameter sharing. Parameter sharing refers to using weights w_i that are multiplied with multiple different inputs x_j, creating sparse interactions and lowering the use of parameters in the network. A visualization of parameter sharing can be seen in Figure 2.14.

Recurrent Neural Networks

Language modelling is one of the most common use cases for this type of architecture for language's sequential nature. A *recurrent neural network* (RNN) is a type of ANN used for processing sequential data. Like CNNs, RNNs make use of parameter sharing to learn pieces of information occurring in multiple positions within the data. RNN gets its name from recurrent computations that form a repetitive structure in

the NN's computation graph. These repeating computations can be thought of as happening throughout time at each step of the sequence. There are many kinds of RNNs, as any NN that involves some type of recurrence is an RNN, much like CNNs with the convolution operation. Recurrent computations can be unfolded into a computation graph with repetitive structure, which can help for the intuition of how backpropagation would work on this type of NN. Figure 2.15 shows the unfolding intuition of the repetitive structure in RNNs.

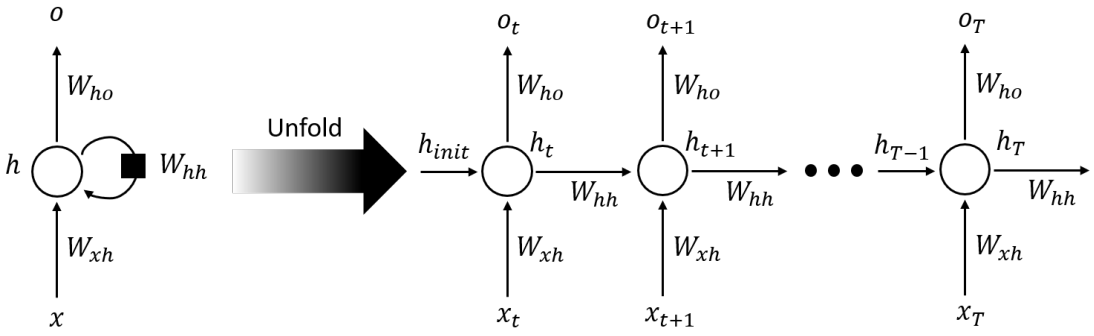


Figure 2.15: Depiction of the unfolding intuition into recurrent neural networks showing the sharing of parameters on each time step. x is the sequential input. W_{xh} are the weights connecting the inputs to the hidden state h, W_{hh} are the weights connecting the hidden states in time, and W_{ho} are the weights connecting the hidden state to the output o of the RNN cell. The operations inside the cells, the circles, are not shown as they differ in each type of cell (original RNN, LSTM, GRU).

The weights or parameters are shared across time steps, which makes the RNN backpropagation algorithm, known as *backpropagation through time* (BPTT), to propagate the error all the way to the first time step at every time step, becoming more computationally expensive as time increases. Each time step can be thought of as a layer in an MLP and thus depending on the sequence length, it can become very long. A problem that can arise from very long backpropagation is the diminishing or exploding of the gradient. This can lead to computational errors, such as memory underflow and overflow. For this reason, a truncated version of BPTT was developed, which made the gradient propagate until a constant number of time steps. This makes the weights contain a small amount of the sequence structure.

Currently, the most effective basic units in neural network sequence models are the *long short-term memory* (LSTM) and the *gated recurrent unit* (GRU) [23]. The idea behind these basic units is to prevent gradients from diminishing and exploding throughout the BPTT process. They allow the network to accumulate gradient infor-

mation over an extended period of time, and after the set amount of time has passed, it will clear the gradient. These units contain a memory state h_t, and also allow an RNN to decide when to use or discard the memory state. The state is set up as multiple units of artificial neurons and activation functions for the purpose of creating a type of memory inside the neural network. The state will become the recurrent unit or *neural network cell* that repeats for every part of the sequence. Unlike an MLP, this cell allows RNNs to contain an internal memory. One method of reducing gradient problems is to use *gradient clipping*. This method prevents the gradient from moving outside a constant range of values. This bounded gradient affects the learning process.

The original RNN cell makes use of the $tanh$ activation function with transformations and additive interactions on the sequential input and the network parameters as a form of memory. More formally described in Equation 2.37, a sequence of input vectors $(x_1, ..., x_T)$ is given, and the RNN computes a sequence of hidden states $(h_1, ..., h_T)$ and a sequence of outputs $(o_1, ..., o_T)$ every time step t from the start of the sequence (1) until the end of the sequence (T). This additive interactions make for a weak form of coupling as a memory state [68]. To create a more robust memory state, Hochreiter and Schmidhuber [56] created the LSTM cell, which is a type of neural network cell that contains multiplicative interactions to produce a memory state. LSTM is the most common RNN cell used in current literature. GRU is another type of neural network cell that contains multiplicative interactions to produce a memory state created by Cho et al. [21]. GRU is considered to be simpler than the LSTM but with similar performance [23]. Other works [68, 123, 118] give a more detailed explanation of RNNs.

$$\begin{aligned} h_t &= tanh(W_{xh}x_t + W_{hh}h_{t-1} + b_h) \\ o_t &= W_{ho}h_t + b_o \end{aligned} \tag{2.37}$$

In Equation 2.37, W_{ij} and b_j are the weight parameters and biases that are learned by the neural network, and the subscripts denote the layer or neurons they belong to. The input layer is denoted by x, the hidden layer is denoted by h, and the output layer is denoted by o. For instance, W_{xh} are the weights that connect the input layer to the hidden layer. When $t = 1$, $W_{hh}h_{t-1}$ will be replaced with an initial vector h_{init} that is either initialized as 0 or to a random value close to zero ($[-1, 1]$).

2.2.4 Tools and packages

This section relates to the practical part of the thesis. It describes the tools used for creating and evaluating the empirical tests that we present in this thesis, as well as the programming tools used when creating the deep learning models.

2.2.4.1 Python

Python[5] is a free and open-source interpreted, high-level, general-purpose programming language with a design philosophy emphasizing code readability. Its accessible nature has made machine learning researchers adopt this programming language not only for its readability, but also for its huge community. The libraries made by the community make code prototyping a fast and efficient task. Specifically for scientific computing projects, there exists different software packages that make the programming environment setup an easy task. Anaconda[6] distribution is one such software package, which contains Python, along with many scientific libraries such as NumPy and Matplotlib.

2.2.4.2 Biopython

Biopython[7] is a free and open-sourced set of libraries and applications for bioinformatics using the python programming language. It contains many tools needed for work in the bioinformatics field. Biopython is a project of the Open Bioinformatics Foundation, which is a nonprofit, volunteer-run group that promotes open-source software development within the biological research community. It offers data structures designed for genomic analysis, as well as tools for use in population genomics and structural bioinformatics. It also contains an interface to BioSQL made for supporting a shared database schema for storing sequence data.

2.2.4.3 SciPy

SciPy[8] is a Python-based ecosystem of open-source software for mathematics, science, and engineering. This ecosystem of packages is often used as a Python alternative to MATLAB because of its more modern and organized nature.

[5] `https://www.python.org`
[6] `https://anaconda.org`
[7] `https://biopython.org`
[8] `https://www.scipy.org`

2.2.4.4 NumPy

NumPy[9] is an open-source project led by volunteers made for scientific computing with Python. It contains data structures for numerical computations such as an N-dimensional array object with many useful functions for efficient mathematical and logical operations on arrays and matrices. It is mostly useful for calculating linear algebra functions and Fourier transformations. This is all made efficient with its tools for integration of Fortran and C/C++ code, which offer high-performance computations. NumPy is part of the SciPy ecosystem.

2.2.4.5 Matplotlib

Matplotlib[10] is an open-source project that is part of the SciPy ecosystem, and supported by a community of volunteers. It is a Python 2D plotting library which produces publication quality figures and interactive plotting environments. It makes the creation of plots such as histograms and scatterplots an effortless process.

2.2.4.6 Pandas

Pandas[11] is an open-source library providing high-performance, easy-to-use data structures and data analysis tools for Python. It is actively supported by a community of volunteers and, like Matplotlib, is also a part of the SciPy ecosystem. It is used mostly for its I/O tools in the handling of large volumes of data. It provides many high-performing data transformation functions as well as pre-processing tools for data intensive tasks.

2.2.4.7 Neural Network Libraries

Neural network libraries described here include Tensorflow, Keras, and Pytorch. All of these contain all the necessary building blocks in section 2.2.3.1, as well as many other desirable functions when working with neural networks. This makes neural network research efficient and less error-prone in the coding part of the research.

[9] `https://numpy.org`
[10] `https://matplotlib.org`
[11] `https://pandas.pydata.org`

Tensorflow

Tensorflow[12] is an 'end-to-end' open-source platform for neural networks. 'End-to-end' means that it is possible to use this program alone to create the whole machine learning pipeline. The pipeline refers to reading the data, processing it, and learning from it to create neural network models. After the model creation step, it can deliver useful user-friendly results or predictions to the end user. As mentioned previously, Tensorflow uses a graph-based approach for its backpropagation computations. Tensorflow's core library is made to develop and train neural network models at a low level, meaning that it is possible to work on the mathematical elements and array expressions that make up the layers of the neural networks. Tensorflow also contains an implementation of the Keras API specification, which is described below.

Keras

Keras[13] is an open-source deep learning library for Python. It is a high-level neural network API capable of running on top of TensorFlow, with a focus on fast experimentation. It allows for fast prototyping thanks to its implementations of popular deep learning architectures and layers. It also helps in the training of the neural networks, having implementations of popular optimizers, activation functions, and other pre-processing methods. It also contains popular datasets for use in the model's training and testing process.

PyTorch

PyTorch[14] is an 'end-to-end' open-source neural network framework focusing on accelerating the path from research prototyping to production deployment. Similar to Tensorflow, it is used to develop and train neural network models at a medium level. PyTorch lies between Tensorflow and Keras, providing more control and flexibility than Keras, and offering more abstraction than Tensorflow's fine or granular level of control over the neural network model's creation. Unlike Tensorflow, PyTorch uses a tape-based approach for its backpropagation computations, making runtime debugging an easy task. More recently, both PyTorch and Tensorflow allow for the transition between tape-based, or what they call 'eager' mode, and graph-based approaches.

[12] `https://www.tensorflow.org`
[13] `https://keras.io`
[14] `https://pytorch.org`

2.2.4.8 Scikit-learn

Scikit-learn[15] is an open-source machine learning library built on top of SciPy and maintained by a group of volunteers. It provides tools for efficient predictive data analysis including classification, regression, clustering, dimensionality reduction, model selection, and pre-processing. Scikit-learn's model selection tools include the cross validation used in our empirical experiments. It also includes the performance metrics we take into consideration in chapter 4. This makes sense as we could eliminate the need of redeveloping optimised functions for use in our project. Since we also used Skorch, we could transfer PyTorch's output into Scikit-learn's metric functions without any additional transformations. Scikit-learn has over 1,500 contributors with a release cycle of three months and over 100 releases to this date.

2.2.4.9 Skorch

Skorch[16] is an open-source Scikit-learn compatible neural network library. The goal of this tool is to make PyTorch and Scikit-learn work together. It does this by providing a PyTorch wrapper with a Scikit-learn interface. It abstracts away the training process of the neural network, making it simple and minimizing the risk of programming errors.

[15] `https://scikit-learn.org/stable/`
[16] `https://skorch.readthedocs.io/en/stable/`

Chapter 3

Related Work

In this chapter, we present some recent tools that have been used for eukaryotic promoter recognition. We also mention prokaryotic PRMs, as previous work on these have the potential to translate well to current studies on eukaryotic organisms, at least at the initial stage. Ab initio methods are exclusively described since they are the methods scrutinized in our work. An advantage of ab initio PRMs is their applicability in reference genomes. This means that the method would work on a wide array of genomes of different cell types that are closely related to each other, both interspecies and intraspecies.

3.1 Early classification

Ab initio PRMs started being researched in the early 1990s with initial sequencing of prokaryotic and eukaryotic genomes [25]. These PRMs were small scale analyses, as there was a limited amount of data and computational resources. The first wave of research came with the use of statistical and mathematical methods to find the TFBS in the core and proximal promoter regions. The extensive PRM review from Vanet et al. [137] describe early PRMs, focusing on prokaryotic genomes. Early eukaryotic PRMs are described in detail by Pedersen et al. [96].

The review from Vanet et al. [137] describes many methods for promoter recognition, such as pattern matching methods of TFBS in a promoter. These matching methods made use of simplified regular expressions that became more complicated as research progressed. These methods did not take into consideration the DNA binding specificities. Thus, methods similar to PWMs started being used as a better alter-

native. In these cases, matrices were used to hold properties and information in all nucleotide positions. Phylogenetic footprinting was used to extract conserved DNA features, spatial structure, or compositional properties. This method was made possible by motif analysis tools such as sequence alignment, expectation-maximization, Gibbs sampling, and Multiple Em for Motif Elicitation (MEME).

Combinatorial methods and exact algorithms for promoter recognition are also mentioned in the review by Vanet et al [137]. To identify promoters with all its elements considered simultaneously, combinatorial methods made use of heuristics such as Klingen-hoffs's method, FastM, and k-tuple enumerations. After these heuristic methods, exact methods were developed for motif extraction from conserved sequences, as well as other exact methods for detecting the extracted motifs in a DNA sequence for promoter recognition. The review by Pedersen et al. describes DNA properties and the tools exploiting these properties for promoter recognition [96]. These properties include CpG islands, DNA methylation and chromosomal structure and domains.

The data used throughout PRM research highly varies. In their 2006 review, Bajic et al. [5] test the performance of ab initio PRMs for predicting TSSs in two datasets:

- HAVANA annotations on ENCODE regions as a reference of TSSs, a dataset to assess the completeness of human genome annotations of the ENCODE regions covering approximately 1% of the human genome sequence.

- Estimated TSS locations inferred from the DBTSS and H-Invitational databases, a dataset based on full length cDNAs, which should largely correspond to genuine TSS locations as they were mostly retrieved from oligo-capped assays.

Before their review in 2004, Bajic et al. [6] made a review based on different data. In their 2006 review, Bajic et al. [5] state that "the direct comparison of the results of (the 2006) study and the one performed (in 2004) on the whole human genome is not possible simply for the reason that the reference data against which assessments are made are different. [...] the two datasets are not very similar, as we have already shown." This tells us that even for data analyses by the same authors, data can vary depending on the time-frame, as more knowledge is gained on the promoter recognition problem. An important view here is that most data used contains elements that are not biologically validated, as much is still unknown. As for the PRMs used in the review by Bajic et al. [6], only CpGProD uses statistical methods.

The review by Ohler and Niemann [92] contains several motif analysis tools derived from statistical methods that are also in the work by Vanet et al. [137], including Gibbs sampling and MEME. There were also other methods including RSA-Tools for yeast and microbial exhaustive search, BBA for phylogenetic footprinting by Bayes alignment, and Pip maker for phylogenetic footprinting by identity plots.

The review by Rombauts at al. [107] contains statistical motif analysis tools such as RSA-Tools, SMILE, R'MES, CONSENSUS, MEME, Gibbs sampling, Motif sampler, AlignACE, Improbizer, YEBIS, Bioprospector, Footprinter 2.0, and CoBind. Wasserman and Krivan [144] also contain statistical motif analysis tools within the previous list, with the exception of Transregio, CISTER, FastM, VISTA/AVID, BALSA, and LAGAN.

In his lecture notes, Sharan [116] explains statistical algorithms for motif analysis, ending in a method for identifying and visualizing regulatory sequences in promoters. The motif analysis algorithms explained include PRIMA, and MEME. For promoter analysis, there exists a probabilistic framework by Segal et al. [115], and as for identifying and visualizing regulatory sequences in promoters, Sharan et al. [117] created a tool called CREME.

3.2 Machine Learning classification

The second wave of research came with the use of machine learning methods to find TFBS and to categorize promoters. Vanet et al. [137] also mention machine learning methods in their PRM review. Machine learning methods here focused on the differences in nucleotides between coding, non-coding regions, and promoter DNA regions. Hidden Markov models, neural networks, quadratic discriminant analysis, and clustering analyses are mentioned in this review for analyzing eukaryotic genomes.

There are more reviews of PRMs employing machine learning techniques. Reviews include work by Bajic et al. [6, 5] containing PRMs such as McPromoter, Fprom, N-SCAN, First exon finder, Dragon promoter finder, Dragon gene start finder, Eponine, Neural network promoter prediction (NNPP), and Promoter2.0.

A review by Ohler and Niemann [92] contains the following PRMs: Promoter 2.0, NNPP, PromoterInspector, McPromoter, and Core-Promoter. Rombauts et al. [107] focus on plant genomes with a thorough description on PRMs, promoter databases, and motif search and prediction methods. PRMs included are McPromoter, PromoterInspector, FunSiteP, Dragon promoter finder, CONPRO, Core-Promoter, Promot-

erScan, Promoter 2.0, and NNPP. Wasserman and Krivan [144] provide a smaller review for metazoan organisms with only three methods: NNPP, PromoterInspector, and First exon finder.

Zeng et al. [163, 162] describe the following PRMs in their review:

- Signal Features: NNPP, First exon finder, CpGProd, Eponine, and Fprom.
- Context Features: PromoterInspector, KLC, PCA for dimensionality reduction of PWMs, Markov encoding, Time series descriptors, PSPA, Promoter 2.0, neural networks, SVM, Dragon promoter finder, and Dragon gene starter finder.
- Structural Features: McPromoter, ProSOM, ARTS, ProStar, and EP3.
- Ensemble methods (Mix of the above features): PromoterExplorer, CoreBoost, CoreBoost-HM, SCS, MetaProm, EnsemPro.

Li et al. [79] include a series of machine learning methods for cis-regulatory elements that play a role in gene expression regulation such as promoters, enhancers, silencers, and insulators. Unsupervised methods include: Bayesian mixture models, hidden Markov models, dynamic Bayesian networks, and spectral learning. Supervised methods include linear and logistic regression models, SVMs, decision trees, random forests, multiple kernel learning, ensemble methods, and some initial neural networks/deep learning methods. While these methods could be used as PRMs, most methods here were developed for other regulatory elements.

Yella and Bansal [157] cover most of the same PRMs introduced previously, with the exception of a structural feature-based PRM known as PromPredict by Ranganan and Bansal [102], which uses free energy of dinucleotides obtained from differential melting stability of DNA duplex as a predictor of promoters, since promoter regions should be less stable than neighbouring regions for transcription initiation to be able to occur. In later work [86], they used this PRM on Arabidopsis and rice genomes.

The most recent machine learning PRM was created by Lai et al. [75]. They developed a PRM that combines pseudo k-tuple nucleotide composition (PseKNC) with position-correlation scoring function (PCSF) to formulate promoter sequences of multiple eukaryotic and prokaryotic organisms. Lat et al. called their tool *iProEP*, and it can recognize both prokaryotic and eukaryotic promoters. To find the optimal feature subset to use for their classification model, Minimum Redundancy Maximum Relevance (mRMR) algorithm is used. SVM is used to distinguish between promoters

and non-promoters, and it was evaluated in a 10-fold cross-validation test. Their results in homo sapiens data is shown in Table 3.1.

Accuracy	Sensitivity	Specificity	AUC
93.3%	92.3%	92.7%	0.974

Table 3.1: DeePromoter dataset

3.3 Deep Learning classification

Here we present a review on recent deep learning ab initio PRMs. To the best of our knowledge, presently this is the only review on this topic. The review by Li et al. [79] for general machine learning promoter recognition includes some early deep learning methods that we will not cover here. We begin our journey into deep learning PRMs starting in the year 2016.

3.3.1 SD-MSAE

Xu et al. [153] developed a machine learning algorithm they call SD-MSAE, standing for "statistical divergence multiple sparse auto-encoders." The algorithm trains an ensemble of models ranging from unsupervised deep learning spare auto-encoders to support vector machines. The complete ensemble of algorithms can be seen in Figure 3.1.

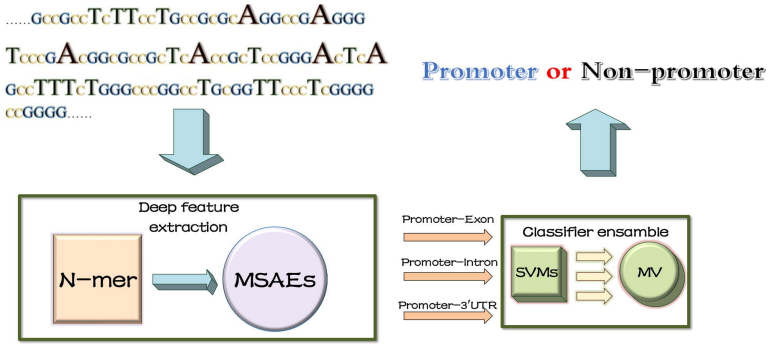


Figure 3.1: Ensemble of machine learning algorithms in SD-MSAE by Xu et al. [153]

They focus on classifying 30,964 promoters without alternative TSSs of length 251, going from -200 to +50 around the TSS obtained through DBTSS. The pro-

moters were differentiated from other genomic regions with data obtained through EID for exons and introns, UTRdb for 3′-UTR. Their work uses statistical features based on k-mers to predict and recognize promoters by representing the promoter's sequence context features. Their work specifically extracts 5-mers or less from the sequences, since these informative and discriminative features create a probability density for each type of sequence. The statistical divergence of these four probability distributions is obtained by KL divergence (KLD), J divergence (JD), and JS divergence (JSD) to select the most informative k-mers as features to discriminate between promoter and non-promoter samples. After securing the most informative k-mers, three auto-encoders are trained with these k-mers to extract deep features in order to obtain a meaningful input for the three SVM classifiers. Each of the three auto-encoders output a value going into an SVM classifier. The three SVM classifiers will then proceed to a majority voting algorithm where a DNA sequence is finally classified as a promoter if two or more SVM classifiers classify the sequence as a promoter; otherwise, it is classified as a non-promoter. The model was evaluated using 10-fold cross validation, and the fold average performance of this model can be seen in Figure 3.2, where it was compared to previous PRMs: 'K-words' [162], ME-HMM [168], and Naive Bayes classifiers [161] (NBC).

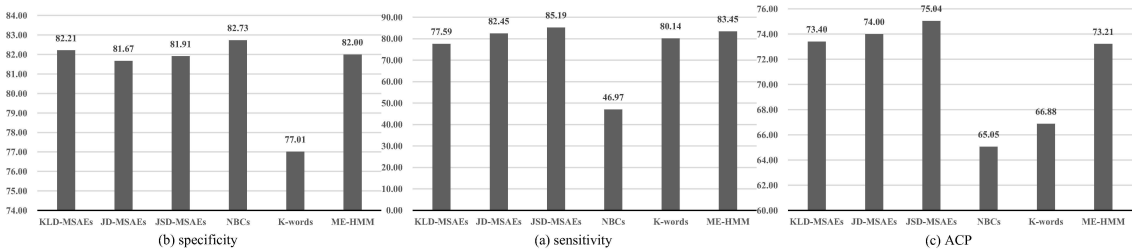


Figure 3.2: Performance comparison of K-words, ME-HMM, NBCs and SD-MSAEs [153]

3.3.2 CNNProm

Umarov and Solovyev. [135] made an end-to-end deep learning algorithm called CNNProm, which stands for convolutional neural network promoter. The model trained by this algorithm consists of a CNN architecture. It encodes the input DNA sequences into what is called a one-hot encoding scheme, i.e. a matrix where the each of the rows are unique nucleotides and the columns are the positions in the sequence. The CNN

contains 200 one-dimensional kernels with length 21 each, followed by a max-pooling layer for all 200 kernels and a ReLU activation function after the max-pooling result, finishing with a fully connected layer of 128 neurons. This last layer is connected to the output layer with a neuron that has a sigmoid activation function. A depiction of the architecture can be seen in Figure 3.3. The model was used to classify promoters of length 251 for different species of organisms in prokaryotic and eukaryotic domains. We focus on the eukaryotic domain, with a specific interest with the human species. The human promoters are further subdivided, having 1426 TATA promoters and 19811 non-TATA promoters as well as 8256 non-promoter sequences for TATA promoters and 27731 non-promoter sequences for non-TATA promoters. Promoter sequences, from -200 to +50 around the TSS, were obtained from the EPD database; non-promoter sequences were obtained from random fragments of their genes located after first exons. The resulting model was evaluated using cross validation, with 70% of the data being used as the training dataset, 20% as the test dataset, and 10% as the validation dataset. The performance results on their model on human data are as follows:

- TATA: Sensitivity=0.95, Specificity=0.98, MCC=0.90
- non-TATA: Sensitivity=0.90, Specificity=0.98, MCC=0.89

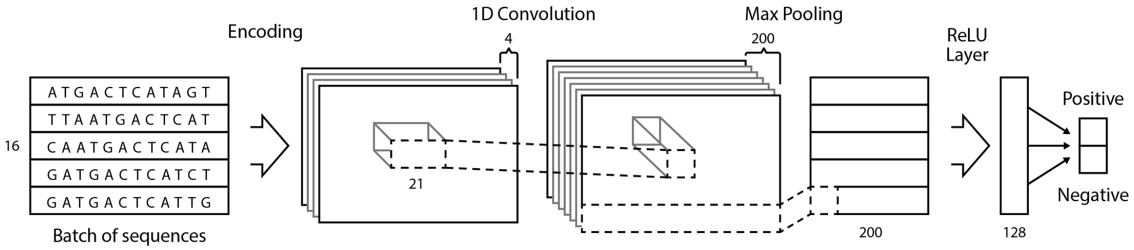


Figure 3.3: The architecture of Umarov and Solovyev's CNNProm

In their work, Umarov and Solovyev also showed how the model validated the existence of the TATA box for TATA promoters, depicted in Figure 3.3. Their model focused on a specific area (-45 to 20 relative to TSS) of the promoter where the TATA box lies, and how the model looks for a conserved sequence that highly resembles a TATA box in that position. Their work however, did not compare their model to any other existing PRMs.

3.3.3 Improved CNN

Qian et al. [100] developed a similar end-to-end deep learning algorithm, which we will call ICNN for improved CNN. As with the previous PRM, the model consists of a CNN architecture. The researchers first focused on validating the importance of the TFBS motifs that have been discovered by biologists throughout previous years. These motifs include:

- GC-Box: located -110 to -80
- CAAT-Box: located -80 to -70
- TFIIB recognition element (BRE): located -37 to -32
- TATA-Box: located -35 to -25
- Initiator (Inr): located -2 to +4
- Downstream core element I (DCE I): located +6 to +11
- Downstream core element II (DCE II): located +30 to +34
- Downstream core element III (DCE III): located +30 to +34
- Downstream promoter element (DPE): located +28 to +32

The sequences they used were of length 300, from position -249 to 50 around the TSS. To validate them, they used SVMs for classifying sequences in three ways:

- Using the full sequence as input.
- Using "element" sequences as input only. Element sequences refer to the previously mentioned TFBS motifs.
- Using only non-element sequences as input.

Their results show that element sequences accounted for most of the valuable signal within the sequence to classify promoters. With this information, Qian et al. devised their deep learning algorithm, mostly focusing on the element sequences. To focus on element sequences, they made the encoding schemes of element and non-element sequences different. Non-element sequences were encoded using one-hot encoding,

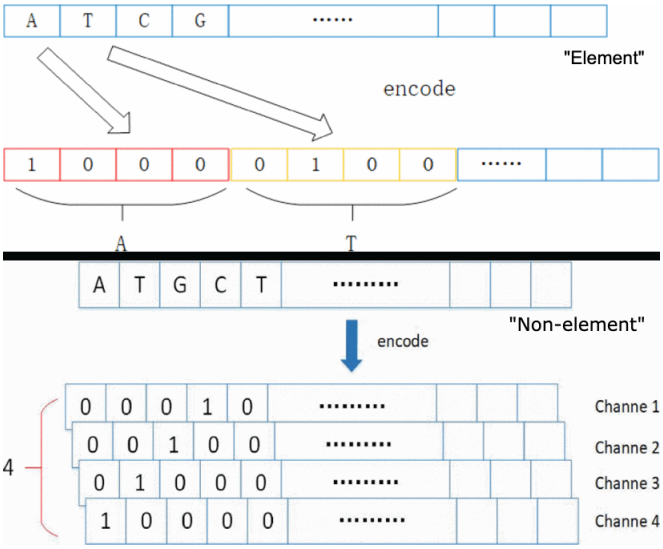


Figure 3.4: The encodings used on Qian et al.'s ICNN [100]. Element sequences encoded at the top and non-element sequences encoded at the bottom.

while element sequences were encoded as a single vector of concatenated nucleotides with each nucleotide taking four values depending on the specific nucleotide in the sequence. The encodings are visualized in Figure 3.4. Once encoded, the non-element sequences are passed to a one-dimensional convolution layer with 200 kernels of length 21, followed by a max-pooling layer for all 200 kernels. The element sequences are left encoded and concatenated to the result of the max-pooling layer. This concatenated result is then passed onto a fully connected layer of 2048 neurons with ReLU activation functions. Finally this result is passed to the output layer with a neuron that has a sigmoid activation function that classifies the promoter from non-promoters. The full architecture is visualized in Figure 3.5.

Qian et al. [100] compared their results to multiple PRMs including their previous SVM approach, CNNProm, and NNPP. Their dataset included 7156 of the most representative promoter sequences from EPD database, and 5235 non-promoter sequences from Berkeley drosophila genome project, where 4345 were intron sequences and 890 were exon sequences. The researchers used 10-fold cross validation, where 90% of the dataset was used as the training dataset and 10% as the test dataset. They did not provide the percentage of their validation dataset in their work. The comparison results can be seen in Figure 3.6, where "Our Model" refers to Qian et al.'s work [100], and "CNN-based" refers to CNNProm by Umarov and Solovyev [135].

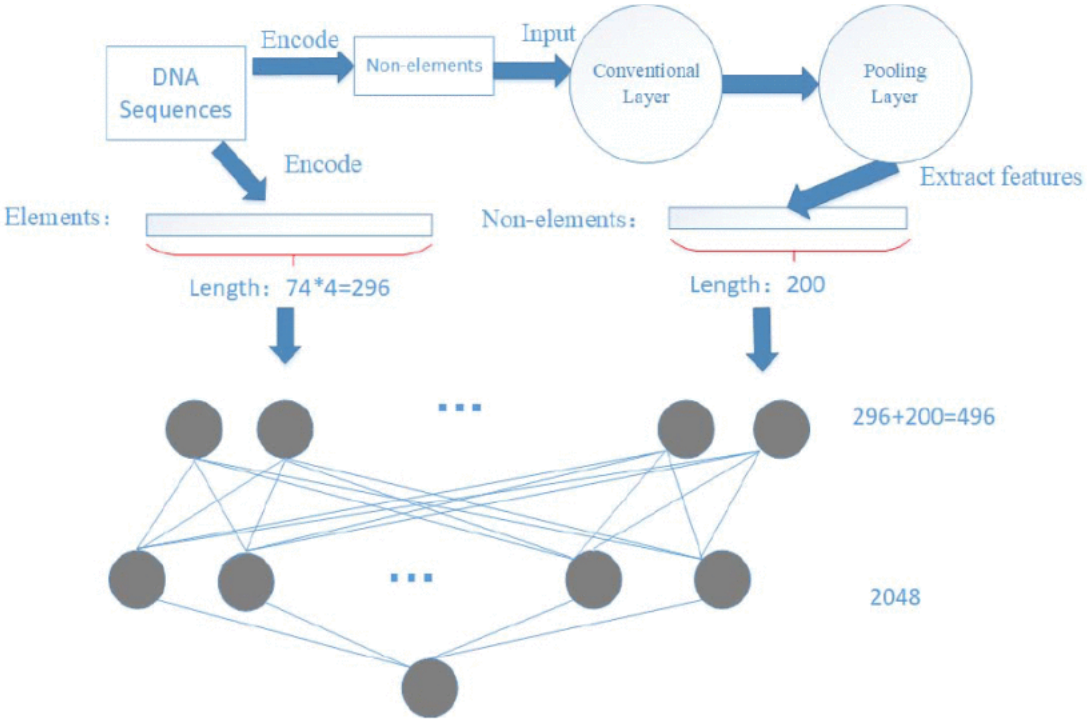


Figure 3.5: The architecture of Qian et al.'s ICNN

Evaluation / Methods	Acc	Sp	Sn
SVM	83.9%	86.1%	86.0%
CNN-based	83.1%	79.4%	81.9%
NNPP2.2	59.2%	71.7%	45.4%
Our Model	89.8%	86.5%	89.6%

Figure 3.6: The comparison results from Qian et al. [100]

3.3.4 DeePromoter

Oubounyt et al. [93] created another end-to-end deep learning algorithm called DeePromoter. The deep learning architecture of DeePromoter consists of multiple convolution layers followed by an LSTM based RNN and a fully connected layer. As with CNNProm, DeePromoter's dataset separated promoters with a TATA-box and promoters without a TATA-box. Their work focused on human and mouse promoters with equal number of promoters and non-promoter sequences for each organism. Every sequence had a length of 300 nucleotides from positions -249 to 50 relative to TSS. The specific number of sequences for each organism and promoter type is shown in Table 3.2.

Organism	Promoter Sequences	Non-promoter Sequences
Human TATA	3,065	3,065
Human non-TATA	26,532	26,532
Mouse TATA	3,305	3,305
Mouse non-TATA	21,804	21,804

Table 3.2: DeePromoter dataset

The promoter sequences were obtained from the EPD database. The non-promoter sequences were created by splitting each positive sequence into 20 subsequences and picking 12 out of the 20 subsequences at random for substitution. The 12 subsequences selected were then substituted by a set of random uniformly distributed nucleotides. The 8 remaining sequences were left intact, thus making each non-promoter sequence 32 to 40 percent similar to their corresponding promoter sequence at most. The model was validated using 5-fold cross-validation with every fold containing the same number of sequences. Three folds were used for training, one fold for validation, and the remaining fold for testing.

The specific architecture of their deep learning model consists of encoding the DNA sequences with a one-hot encoding scheme. As with CNNProm, they have different architectures for each type of promoter, but we focus on their non-TATA promoter classifier. For this classifier, Oubounyt et al. [93] used three convolution layers, with each layer having 32 one-dimensional kernels of size 27, 14, and 7, respectively. This is followed by a ReLU activation function after every convolution and a max-pooling layer of size six. The architecture after contains a layer with 50% of the neurons having dropout regularization for each max-pooling layer. The outputs of

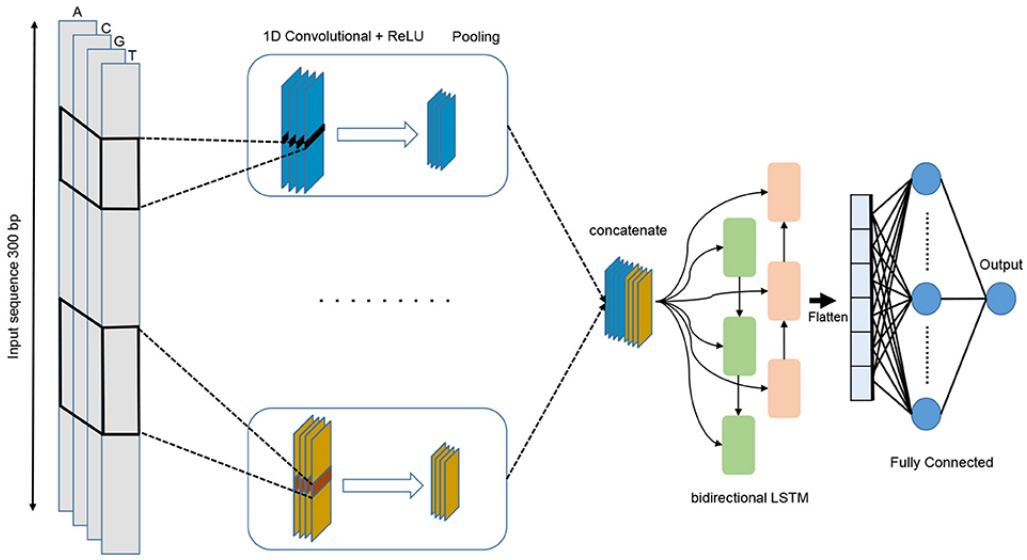


Figure 3.7: The architecture of Oubounyt et al.'s DeePromoter [93]

all three dropout regularization layers are then concatenated and used as input to a bidirectional long short-term memory (BiLSTM) [114], which are followed by another dropout regularization layer with the same properties. This is followed by a fully connected layer of 128 neurons with ReLU activation function and the output layer with a neuron that has a sigmoid activation function. The architecture is depicted in Figure 3.7.

Oubounyt et al. compared the performance of their DeePromoter model with CNNProm in Figure 3.8 for both humans and mice promoters using their dataset [93].

Oganism	Method	Precision	Recall	Mcc
	DeePromoter	**0.93**	**0.95**	**0.88**
Human TATA	CNNProm	0.75	0.91	0.62
	DeePromoter	**0.97**	**0.95**	**0.92**
Human non-TATA	CNNProm	0.58	0.83	0.26
	DeePromoter	**0.92**	0.95	**0.87**
Mouse TATA	CNNProm	0.68	**0.96**	0.56
	DeePromoter	**0.91**	**0.90**	**0.82**
Mouse non-TATA	CNNProm	0.54	0.86	0.17

Figure 3.8: The comparison results from Oubounyt et al.

3.3.5 DeeReCT-PromID

Umarov et al. [134] acknowledged the performance issues of their PromCNN model regarding false positive promoter classifications. In their most current work, they describe a way to mitigate this issue by changing how the deep learning model is trained while also changing their strategy from promoter recognition into TSS location recognition. This means that their evaluation would include any sequence within 500 nucleotides of the TSS as a promoter, and everything else as a non-promoter. Their dataset contained sequences obtained from the EPD database. Umarov et al. obtained 455 human genomic sequences from locations -5000 to +5000 relative to the TSS of each sequence. Different promoter regions were evaluated, and the researchers concluded that sequences ranging from locations -200 to 400 were the most effective promoter regions. A sequence of a given size inside the promoter region was considered to be a promoter, and sequences outside the promoter region were considered to be a non-promoter. The deep learning architecture includes a one-hot encoding scheme for each input sequence followed by two CNNs. One CNN was designed to process patterns in the sequence with positional information, while the other CNN was designed to process the GC content of the sequence, which does not require positional information. An average pooling layer was used for the GC content extraction process, which removes positional information. The output from the two CNNs was then concatenated and passed to an output layer containing two neurons with a softmax activation function to output the probability of the sequence being a promoter. A depiction of this architecture can be seen in Figure 3.9.

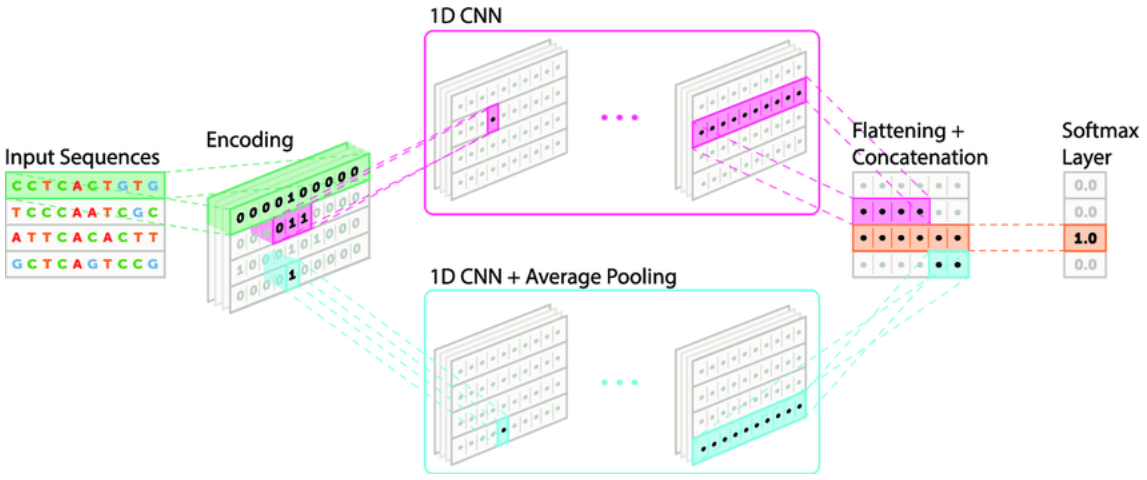


Figure 3.9: The architecture from Umarov et al.'s DeeReCT-PromID [134]

Their proposed training approach is as follows:

1. Train a model with the current negative set.
2. The model is applied to the dataset with long genomic sequences and false positives are recorded.
3. A subset of false positives with the highest scores given by the model (the ones that are most similar to the true promoters) from each long sequence are chosen for the new negative set.
4. A new negative set is then constructed by merging the previous one with the new false positives.

This training procedure is repeated until there are minimal false positives found processing the training set, "which helps the model learn deeper and less obvious features to recognize a promoter sequence" [134].

Umarov et al. compared their DeeReCT-PromID model with other PRMs that shared their same TSS evaluation method and dataset in order to obtain a fair comparison. The results are shown in Figure 3.10 and Figure 3.11.

		DeeReCT-PromID	PromCNN	PromCNN*	FPROM	FPROM*	TSSW	Promoter 2.0
Recall	TATA+	0.715	0.884	0.700	**0.908**	0.647	0.691	0.845
	TATA−	0.745	**0.948**	0.889	0.868	0.764	0.775	0.810
	BOTH	0.741	**0.940**	0.865	0.873	0.749	0.764	0.814
Precision	TATA+	**0.783**	0.118	0.242	0.236	0.491	0.252	0.107
	TATA−	**0.758**	0.127	0.320	0.227	0.476	0.259	0.104
	BOTH	**0.761**	0.126	0.310	0.228	0.478	0.258	0.105
*F*1 score	TATA+	**0.747**	0.208	0.360	0.375	0.558	0.369	0.190
	TATA−	**0.751**	0.224	0.471	0.360	0.587	0.388	0.184
	BOTH	**0.751**	0.222	0.456	0.362	0.584	0.386	0.186
Error per correct	TATA+	**0.277**	7.464	3.138	3.234	1.037	2.965	8.349
	TATA−	**0.320**	6.885	2.121	3.403	1.099	2.857	8.581
	BOTH	**0.314**	6.953	2.225	3.381	1.092	2.869	8.551
Error per 1000 bp	TATA+	**0.020**	0.660	0.220	0.294	0.067	0.205	0.706
	TATA−	**0.024**	0.653	0.189	0.295	0.084	0.221	0.695
	BOTH	**0.023**	0.654	0.192	0.295	0.082	0.219	0.696

Note: The best performance for each measure is in bold.

Figure 3.10: The comparison results from Umarov et al.

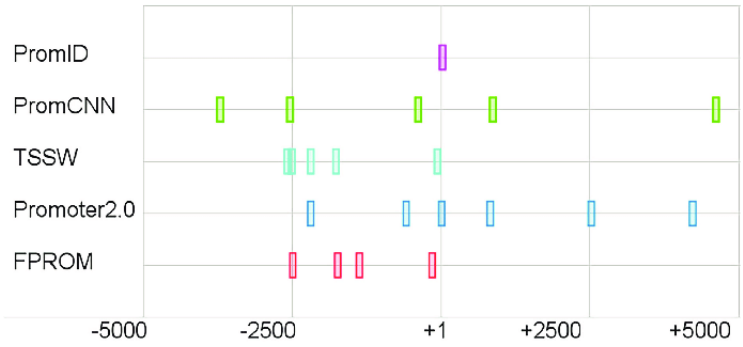


Figure 3.11: The comparison results from Umarov et al. showing places in a sequence where a TSS is recognized in different PRMs [134]

3.3.6 DCDE-MSVM

Xu et al. [154] continued to improve their SD-MSAE algorithm, coming up with what they call the "deep convolutional divergence encoding" (DCDE) method. Like their previous PRM, DCDE is also an ensemble of different machine learning algorithms used in conjunction for promoter classification. The complete ensemble of machine learning algorithms is depicted in Figure 3.12.

The data Xu et al. used for training and evaluating their DCDE model is the same as their previous work. The data includes 30964 promoters with locations from -200 to +50 relative to TSS obtained from DBTSS. As for non-promoter sequences, they randomly extracted 10000 exons and 10000 introns from EID database, and 10000 3′-UTRs from UTRdb database. All sequences had the same length of 251 nucleotides.

As with their previous work, DCDE focuses on encoding the genomic sequence first by using three statistical divergence techniques: KL divergence, J divergence, and JS divergence. DCDE uses a series of relative entropy estimator algorithms based on the statistical divergence techniques previously mentioned to obtain the most informative k-mers, where Xu et al. settled to use $k = 3$. After obtaining the most informative 3-mers with each of the three statistical divergence techniques, a CNN for each technique was used to reduce computational complexity and get more discriminative encoding features. Each CNN contained a convolution layer with ReLU activation, a max-pooling layer, and a fully connected layer. In Figure 3.12, the solid red arrow points to the depiction of the CNN architecture. The CNN is trained using 20% of the data, which include randomly selected 2000 promoters, 2000 exons, 2000 introns, and 2000 3′-UTRs. The testing set includes randomly selected

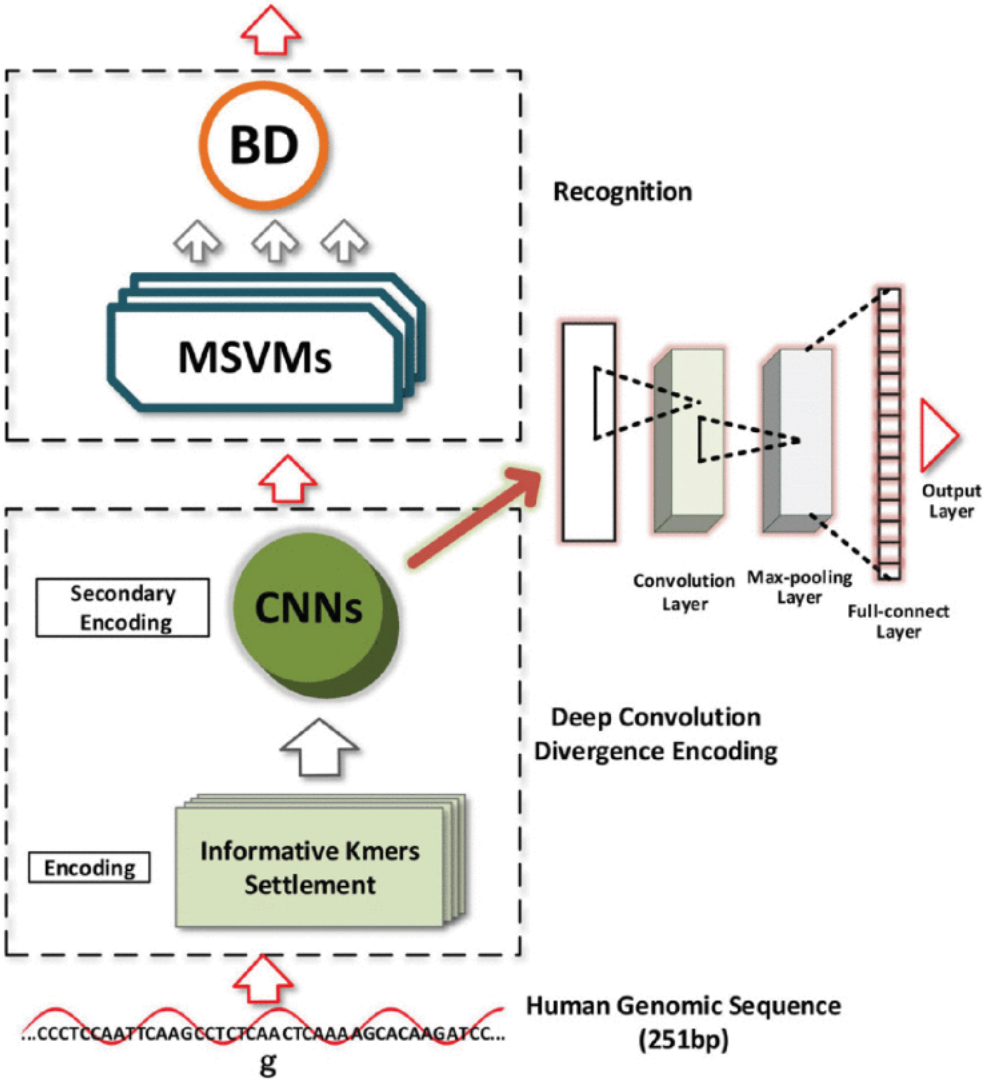


Figure 3.12: Ensemble of machine learning algorithms in DCDE from Xu et al. [154]

2000 promoters, 1500 exons, 1500 introns, and 1500 3′-UTRs. This process concludes the encoding of the DNA sequence. To classify sequences into promoter and non-promoter, multiple SVMs follow the encoding process. To aggregate the results of the SVM classifiers, a bilayer decision model is used instead of the simple majority voting algorithm from their previous work. The bilayer decision algorithm has two stages:

1. Output the probability of the sequence being a promoter as an aggregate of the three SVMs outputs in the following way:

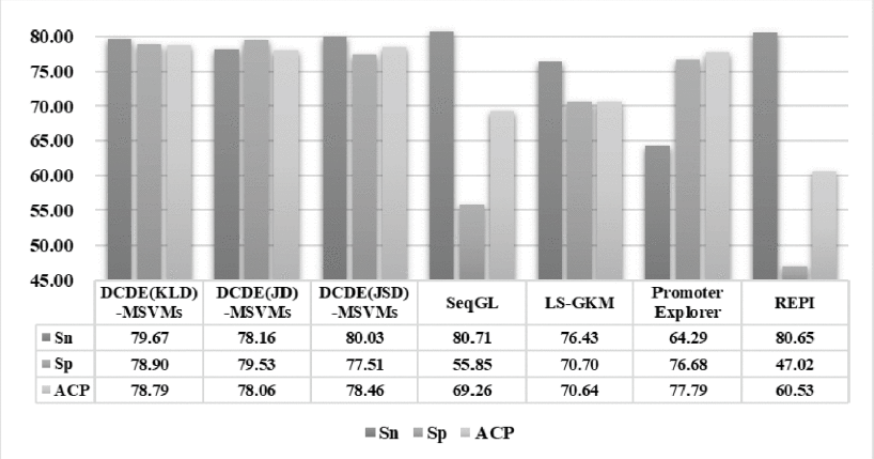
    - Output the maximum probability of all SVMs if the majority classify as

promoter.

- Output the average probability of all SVMs if there is no majority agreement on a promoter classification.
- Otherwise, output the minimum probability of all SVMs.

2. Output the classification depending on the probability of the first stage being higher or lower than a constant threshold set by the researchers.

Xu et al. compared their three models with previous PRMs using the same data for all the performance evaluation tests. The results are shown in Figure 3.13.



	DCDE(KLD) -MSVMs	DCDE(JD) -MSVMs	DCDE(JSD) -MSVMs	SeqGL	LS-GKM	Promoter Explorer	REPI
Sn	79.67	78.16	80.03	80.71	76.43	64.29	80.65
Sp	78.90	79.53	77.51	55.85	70.70	76.68	47.02
ACP	78.79	78.06	78.46	69.26	70.64	77.79	60.53

Figure 3.13: The comparison results from Xu et al. [154]

3.3.7 Prokaryotic Deep Learning Classification

For the sake of completeness, we introduce the latest prokaryotic deep learning PRMs. These PRMs focus on classifying promoter sequences taking into consideration different sigma factors. The PRMs include work by Liu et al. [81], Le et al. [77], Lin et al. [80], Tayara et al. [130], and Xiao et al. [152].

Chapter 4

Evaluation Metrics

This chapter presents the metrics used for evaluating machine learning binary classification models. We specifically describe the evaluation metrics used to compare and validate deep learning classification models designed for promoter recognition. We also describe the advantages and disadvantages of these metrics, and give suggestions for properly evaluating models that we learned from our approaches.

4.1 Binary Classification

Choosing the proper metrics for the evaluation of supervised machine learning models is a crucial step for testing the reliability of the models to perform the task that the model has been trained to perform. In our case, promoter recognition can be represented analogously to the classification of DNA sequences. For our purposes, sequences can be classified in only two possible ways - promoter or non-promoter sequences - meaning that promoter recognition can be designed as a binary classification task. There are two main forms of evaluating binary classification tasks. One way, which we will call *thresholded*, is where the user chooses a threshold. The classification is then identified depending on the resulting value from the model being lower or higher than the specified threshold t, where $0 < t < 1$. As the classification happens on two classes, there are only two possible classification results. The results are then set to 0 for one class and 1 for the other class. Models can then output a value r, where $0 < r < 1$, and the threshold would be used to discriminate between classes. For example, if $r = 0.6$ and $t = 0.5$, then the results would be classified as class 1 because $r > t$. The other way, which we will call *non-thresholded*, is to

evaluate the classifier as the threshold varies over all possible values without the need of testing the quality of a particular choice of threshold.

Binary classification is a type of supervised learning in machine learning algorithms. This means that the algorithm must be supplied with a true label for every sample of its dataset as described in section 2.2. For the following metrics, we denote $y_i \in Y$ as the true label for the i-th sample, and $\hat{y}_i \in \hat{Y}$ as the output label coming from a machine learning model for that same sample i. Y and $\hat{Y}$ are the complete set of true and output labels respectively corresponding to all the dataset samples.

The thresholded metrics that we use include accuracy, balanced accuracy, F-measures, precision, recall, Matthews coefficient correlation (MCC), and Jaccard index. We also provide a confusion matrix for easier visualization. The non-thresholded metrics that are discussed include Area Under the Curve (AUC) and Average Precision (AP).

Each metric is a function that takes as input a set of true labels Y and output labels $\hat{Y}$, and outputs a real value r. The range of r depends on the metric, and we have chosen to compile them in Table 4.1 and Table 4.2 for easier reference.

Thresholded Metric	Range
Accuracy	$[0, 1]$
Balanced Accuracy	$[0, 1]$
F-measures	$[0, 1]$
Precision	$[0, 1]$
Recall	$[0, 1]$
Matthews Coefficient Correlation	$[-1, 1]$
Jaccard Index	$[0, 1]$

Table 4.1: Range of thresholded metrics

Non-thresholded Metric	Range
Area Under the Curve	$[0, 1]$
Average Precision	$[0, 1]$

Table 4.2: Range of non-thresholded metrics

In the statistical sense, classification involves identifying the set of categories that a sample belongs in. In terms of supervised learning, a true label is known and

- The output label is positive and matches the true label, known as a true positive (tp).
- The output label is negative and matches the true label, known as a true negative (tn).
- The output label is positive and does not match the true label, known as a false positive (fp).
- The output label is negative and does not match the true label, known as a false negative (fn).

Figure 4.1: Possible classification scenarios

assumed to be given for each sample. When a model outputs a label for a sample, the algorithm that is training the model needs to know if the model made a mistake in order to correct it. The four possible scenarios that can happen in classification models are shown in Figure 4.1. In terms of promoter recognition, positives would be considered classified as promoters and negatives as non-promoters. There are two scenarios involving correct classification and two involving incorrect classification. We refer to the indicator function to separate correct and incorrect classifications. The indicator function is defined as

$$\mathbf{1}(\text{condition}) := \begin{cases} 1 & if \text{condition} = true, \\ 0 & if \text{condition} = false \end{cases} \tag{4.1}$$

4.1.1 Thresholded metrics

In the following metrics, the threshold is assumed to already be chosen by the user. Depending on the choice of threshold, the classifier gives a binary output which can be interpreted as promoter or non-promoter.

4.1.1.1 Accuracy

Accuracy measures the number of correct predictions over the total number of predicted samples. The function used to compute the accuracy is the following

$$accuracy(Y, \hat{Y}) = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} \mathbf{1}(\hat{y}_i = y_i) \tag{4.2}$$

where Y is the complete set of true labels and $\hat{Y}$ is the complete set of output labels. That is, $\hat{y}_i$ is the predicted label of the i-th sample and y_i is its corresponding true label over the total number of samples $n_{samples}$. We can also define accuracy in terms of statistical classification measures,

$$accuracy(Y, \hat{Y}) = \frac{tp + tn}{tp + tn + fp + fn} \tag{4.3}$$

For easier visualization of these classification measures, we make use of confusion matrices. In a confusion matrix, an entry (i, j) corresponds to the number of observations in group i that are predicted to be in group j. An example can be seen in Figure 4.2.

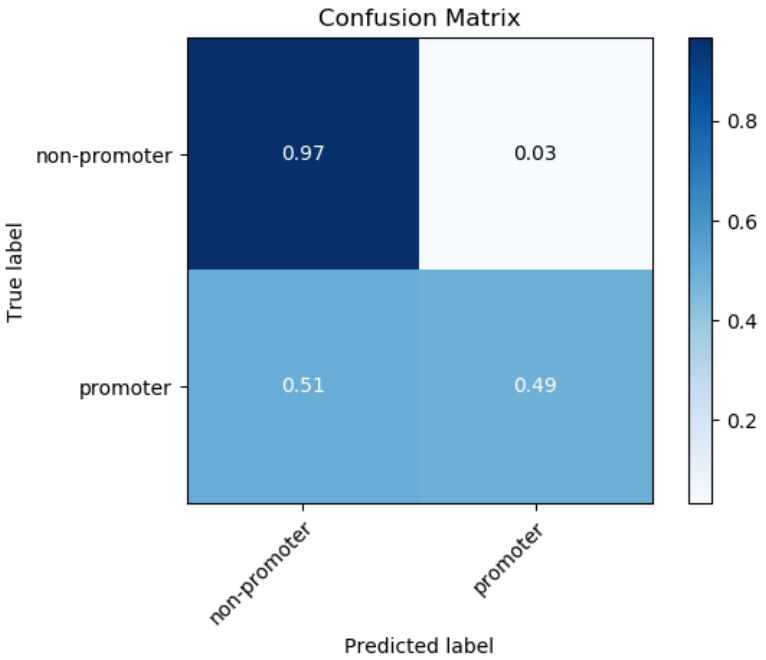


Figure 4.2: Example of a confusion matrix

4.1.1.2 Sensitivity

Sensitivity, also known as *True Positive Rate* (tpr), measures the proportion of promoters in the dataset that are correctly identified.

$$tpr = \frac{tp}{p} = \frac{tp}{tp + fn} \tag{4.4}$$

In Equation 4.4, p are all the positively classified samples by the model which comprise the tp and fn classifications. Therefore, $p = tp + fn$.

4.1.1.3 Specificity

Specificity, also known as *True Negative Rate* (tnr), measures the proportion of non-promoters that are correctly identified.

$$tnr = \frac{tn}{n} = \frac{tn}{tn + fp} \tag{4.5}$$

In Equation 4.5, n are all the negatively classified samples by the model which comprise the tp and fn classifications. Therefore, $n = tn + fp$.

4.1.1.4 Balanced Accuracy

Balanced accuracy, also known as Youden's J statistic or Youden's index, avoids inflated performance estimates on imbalanced datasets. Imbalanced datasets refer to datasets where the number of data points available for different classes is not equally or similarly proportional. For binary classification, where two classes are present, it means that the class proportions are not close to 50%. It is defined as the macro-average of recall scores per class, or equivalently, raw accuracy where each sample is weighted according to the inverse prevalence of its true class. Thus, for balanced datasets, balanced accuracy is equal to accuracy. In our case, the data is highly imbalanced as promoter sequences account for 0.1% of all sequences, which is why we use this metric on top of accuracy. To compute this metric, we need to have a balanced sample weight

$$\hat{w_i} = \frac{w_i}{\sum_j \mathbf{1}(y_j = y_i) w_j} \tag{4.6}$$

As in the previous section, y_i is the true label of the i-th sample, and w_i is its corresponding sample weight. Given the predicted sample value $\hat{y_i}$, balanced accuracy

is defined as:

$$balanced\_accuracy(Y, \hat{Y}, w) = \frac{1}{\sum \hat{w}_i} \sum_i \mathbf{1}(\hat{y}_i = y_i)\hat{w}_i \tag{4.7}$$

where w is the weight of the classes for the complete set of samples in the dataset. We also define balanced accuracy in terms of classification measures using True Positive Rate (tpr) and True Negative Rate (tnr), also known as Sensitivity and Specificity.

$$balanced\_accuracy(y, \hat{y}, w) = \frac{tpr + tnr}{2} \tag{4.8}$$

4.1.1.5 Precision

Precision, also known as *Positive Predictive Value* (ppv), is a measure of the classifier in terms of how relevant the results are. Precision decreases as the number of false positives increase.

$$ppv = precision = \frac{tp}{tp + fp} \tag{4.9}$$

4.1.1.6 Recall

Recall is a measure of how complete the results are in a classifier.

$$recall = tpr = \frac{tp}{tp + fn} \tag{4.10}$$

4.1.1.7 F-measures

The *F-measures*, more precisely F_β for any positive real β, can be interpreted as a weighted harmonic mean of the precision and recall.

$$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{\beta^2 \cdot precision + recall} \tag{4.11}$$

With $\beta = 1$, the recall and precision are equally weighted as seen below.

$$F_1 = 2 \cdot \frac{ppv \cdot tpr}{ppv + tpr} = 2 \cdot \frac{precision \cdot recall}{precision + recall} = \frac{2tp}{2tp + fp + fn} \tag{4.12}$$

4.1.1.8 Matthew's correlation coefficient

Matthew's correlation coefficient (mcc) is a measure of the quality of a classification even with high imbalances in class sizes. A coefficient of +1 represents a perfect prediction, 0 a random prediction and -1 an inverse prediction. It is defined as:

$$mcc = \frac{tp \cdot tn - fp \cdot fn}{\sqrt{(tp + fp)(tp + fn)(tn + fp)(tn + fn)}} \tag{4.13}$$

4.1.1.9 Jaccard similarity index

Jaccard similarity index, also known as *Intersection over Union* (IoU), is a statistic measure used for gauging the similarity and diversity between predicted and true labels. The Jaccard index of the i-th sample is defined as:

$$J(y_i, \hat{y}_i) = \frac{|y_i \cap \hat{y}_i|}{|y_i \cup \hat{y}_i|} \tag{4.14}$$

4.1.2 Non-Thresholded metrics

4.1.2.1 Area under the Curve

AUC stands for *Area under the Curve*. The curve we are talking about is the *Receiver Operating Characteristic* (ROC) curve. The ROC curve is created by plotting the tpr against the False Positive Rate (fpr). The fpr is calculated with the following formula:

$$fpr = 1 - tnr \tag{4.15}$$

AUC illustrates the diagnostic ability of a binary classifier system as its discrimination threshold T is varied. From Hand [52] we know that the AUC is equivalent to the probability that a randomly chosen member of one class has a smaller estimated probability of belonging to the other class, and vice-versa. This results a measure of separability between the two estimated probability distributions as can be seen next.

$$AUC = \int_{x=0}^{1} tpr(fpr^{-1}(x))dx = \int_{-\infty}^{\infty} tpr(T)fpr'(T)dT = P(X_1 > X_0) \tag{4.16}$$

where X_1 is the score for a promoter sample and X_0 is the score for a non-promoter

sample. In this probabilistic view, tpr is given by,

$$tpr(T) = \int_{T}^{\infty} f_1(x)dx \tag{4.17}$$

while fpr is given by,

$$fpr(T) = \int_{T}^{\infty} f_0(x)dx \tag{4.18}$$

where f_1 and f_0 are probability densities for X_0 and X_1 respectively if the instances belong to its respective classes.

This illustrates that we have a non-threshold case, as we include all possible thresholds in our analysis. An example of AUC is shown in Figure 4.3

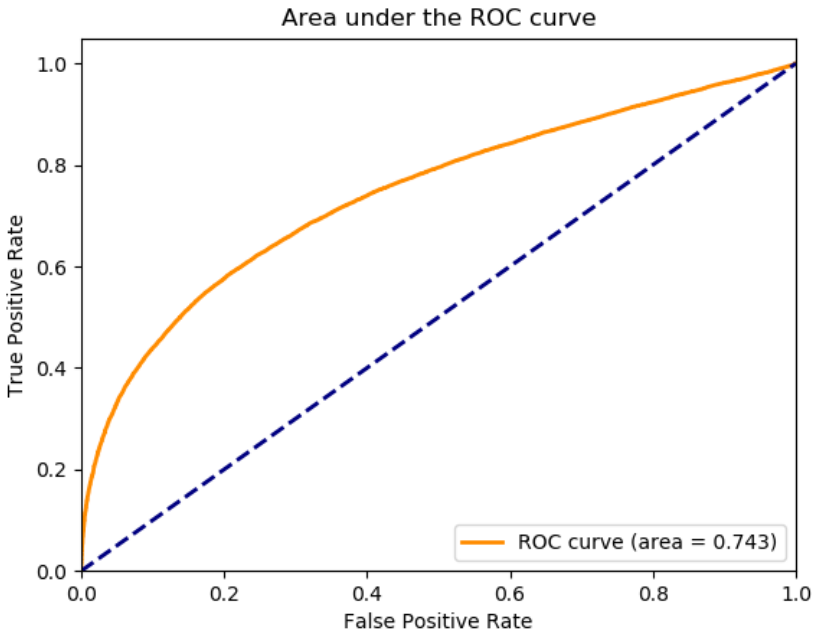


Figure 4.3: Example of a ROC curve and its AUC.

4.1.2.2 Average Precision

Like the ROC curve, the *Precision-Recall* (PR) curve computes its results over a varying discrimination threshold T. The difference is that it is plotting precision against recall instead of TPR against FPR.

Average Precision (AP) can be considered as the area under the PR curve and can be computed by:

$$AP = \int_0^1 precision(recall(T))dT \tag{4.19}$$

which can also be defined as a sum in the following way:

$$AP = \sum_n (R_n - R_{n-1})P_n \tag{4.20}$$

where P_n and R_n are the precision and recall at the n-th threshold. An example of a PR curve is shown in Figure 4.4

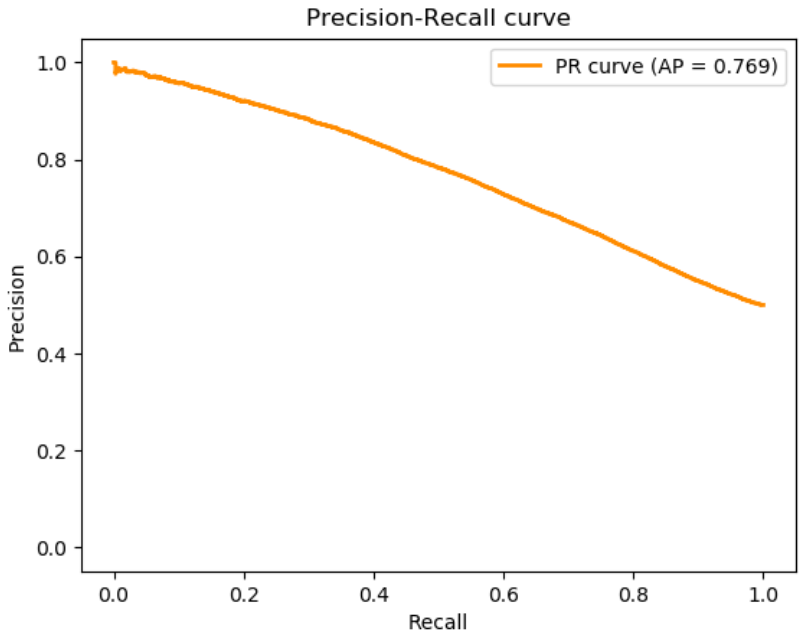


Figure 4.4: Example of a PR curve with its AP

4.2 Choosing appropriate metrics

Given a particular model, different types of machine learning models are evaluated using different metrics according to the specific properties of the model's results. Classification models encompass metrics that evaluate the correctness of the respective model when discriminating an input between different classes. For binary classifica-

tion, only two possible classes are viable for an input. As noted previously in this case, only two types of classification errors are possible. For any classification model, it is desirable to be able to classify all possible inputs correctly. In practice, this is not always possible, and as such we must decide which types of errors are more tolerable for a particular classification task. In our promoter recognition task, it is important for a model to be able to recognize promoters in a precise manner, meaning that *fp*s should be minimized. This is because the model can be used as an estimate, which can be further validated using (expensive) biological assays. If estimated sequences are *fp*s, the biological assays that would subsequently be used to validate these estimates will end up as a waste of resources. On the other hand, if *fp*s are very infrequent, the confidence in using the model will grow and might completely remove the need of biological assays for promoter validation in the future. The other type of error possibly resulting from a binary classification model is a *fn*. In promoter recognition, this means that the model would fail to identify promoter sequences. If *fn*s are high, the model would miss too many promoter sequences, making it undesirable as a model. The optimal model would include very few to no *fp*s and *fn*s. If the optimal model is not feasibly possible to create, a tradeoff would need to occur. In this case, *fn*s would be less problematic. For example, if we have a model that has very low levels of *fp*s and moderate *fn*s, we would still have a precise model. This is because we have high confidence in the model that when it classifies a promoter, it will give an actual promoter sequence even though we might not have the complete list of promoter sequences. On the contrary, if we have a model with low level of *fn*s and moderate *fp*s, our confidence drops significantly as we will not be able to tell which identified promoters are actual promoters from the resulting list. This is why in our case, *fp*s are considered high-risk, while *fn*s are not.

Since the different types of errors are weighted differently, different focus will be needed for the different thresholded metrics that evaluate these errors. Accuracy focuses on the overall correctness in the classifications made by the model. It is important to note that when having highly imbalanced datasets, accuracy becomes obsolete. This is because the result is skewed by the larger class. If all the data is classified as the larger class, the accuracy is exactly the percentage of the larger class within the dataset. In our promoter recognition task, non-promoters account for approximately 99.9%, meaning that if the model classified every sequence as non-promoter, the model's accuracy would be 99.9%. This is meaningless, as we need a model that recognizes promoter sequences. In this case, it is advisable to use

a different metric, such as balanced accuracy, to account for such an imbalanced dataset.

Sensitivity and specificity are used to measure the proportion of correctly classified data points in each class for binary classification. For promoter recognition, we are more interested in promoters being recognized as opposed to non-promoters, and thus we focus on sensitivity as our metric of importance between these two metrics. Remember that sensitivity measures how many promoter sequences were recognized out of all possible promoter sequences in the dataset. In the same manner as recall, precision should also be a focus in this early stage, as it measures the amount of recognized promoters by the model that are truly promoters. Both precision and recall focus on the promoter class, which is our main task in promoter recognition. All metrics serve a purpose, but in an early stage of promoter recognition where models might not be highly accurate, we should focus on the sensitivity and precision of the model. Once the model's classification within promoter sequences is proportionally correct, we can move onto secondary objectives, such as increasing its specificity, to create an optimal model.

F-measures combine both precision and recall into one metric, when $0 < \beta < 1$, F_β puts more weight into precision. Hence, higher thresholds would result in higher F_β scores. On the contrary when $\beta > 1$, lower thresholds result in higher F_β scores, meaning that F_β is putting more weight into recall. Previously we mentioned that when $\beta = 1$, the function will put the same weight for both precision and recall. If possible when evaluating a model with F-measures, the model should be tested on multiple thresholds to find the best possible threshold that maximizes both precision and recall. A visual representation of this can be seen in Figure 4.5, showing that F-measures are sensitive to different threshold values.

Matthew's correlation coefficient calculates how the model's classifications are correlated to the true labels. This means that a positive and negative correlation with the true labels can be known from this metric. There can be cases where negative correlation is beneficial, but it is desired mostly to obtain a positive correlation to show that the model is classifying in a correct manner. This metric is not very sensitive to threshold changes, and as such is useful when a high number of thresholds cannot be tested. This metric is also most useful when working with imbalanced datasets. The robustness of *mcc* on different thresholds is shown in Figure 4.6.

Jaccard similarity is mostly used in the task of object detection in computer vision to measure the overlap between the bounding box - to detect the object - and the

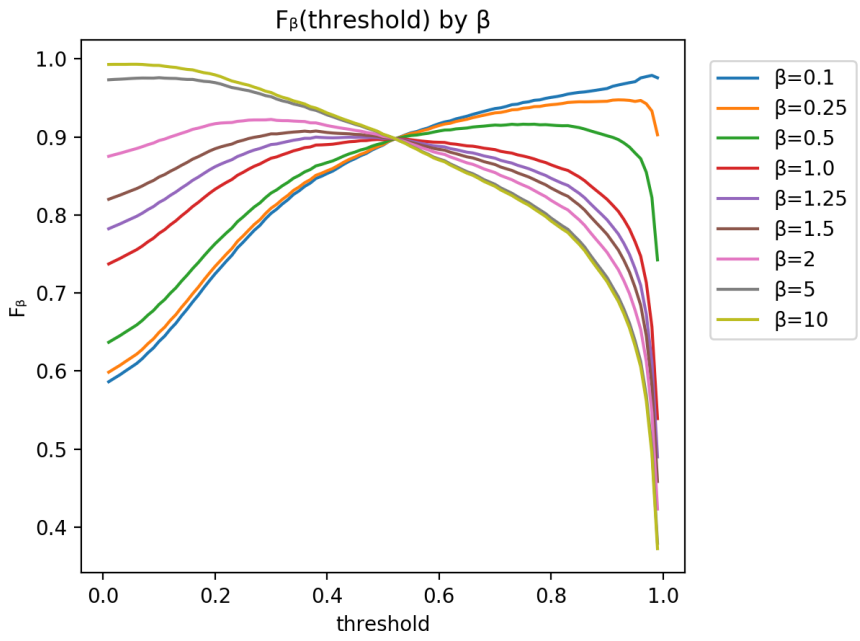


Figure 4.5: Visual representation of F_β scores in multiple thresholds

object's location. This metric can be used in binary classification as a way to measure how similar the classification results are to the true labels, but can be extended to promoter recognition where a bounding box is used to detect a promoter inside a DNA sequence. To the best of our knowledge, this method of promoter recognition has not been attempted, and such metric can translate between both methods to provide a comparison between them.

Non-thresholded metrics show how a model behaves in multiple threshold settings. Visualizing these metrics can give an overall view of how a model trades-off the different metrics being compared. In ROC curves, the comparison happens between the TPR and the FPR, while the PR curve compares precision and recall. It might seem like these metrics are the best way to show a model's capabilities, but unfortunately, current computation limits the amount of thresholds a model can be validated on, and the model can only be validated on a single threshold - which decides the classification result that is validated. Thus, these metrics can be utilized as a threshold selection mechanism for the model as part of the training process of the model. The model should be subsequently tested on a held-out dataset to estimate the expected performance of the classifier with the threshold [38].

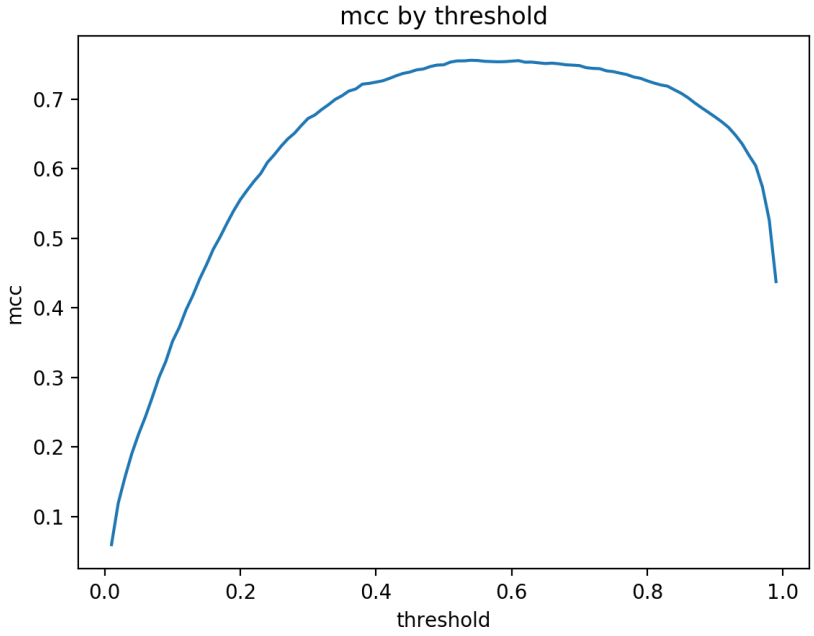


Figure 4.6: Visual representation of *mcc* values in multiple thresholds and β values

Chapter 5

Natural Language Approach

In this chapter, we describe our procedural approach involving the research of deep learning ab initio PRMs in detail. The goal of this first approach was to understand how the neural networks were tackling the human promoter recognition problem. We start by acquiring a dataset fit for the human promoter recognition task. We then describe the encoding methods used on data for translating it into inputs accepted by deep learning algorithms. We follow by describing methods to interpret deep learning architectures and our choice of interpretation method: attention. We also briefly describe methods for data preprocessing for machine learning algorithms with specific properties that compromise PRMs. We end with empirical experiments and results for several deep learning architectures exhibiting the above methods.

5.1 Datasets

As a first step towards understanding the ab initio promoter recognition problem, we must delve into data obtained from nature, which previous researchers have already analysed. This can help us find properties that were missed previously or help reinforce the previous research with our findings. We also obtained this data as a method of comparing multiple deep learning architectures to Qian et al.'s PRM, which was the latest ab initio deep learning PRM at the time. Therefore, the data we obtained for our first approach into understanding PRMs comes from Qian et al. [100]. As described in chapter 3, the data consists of 7156 human promoter sequences from EPD and 5235 human non-promoter gene sequences from BDGP.

Finding the data was a non-trivial process, and as such we will explain the exact

details to obtain it. For the human promoter sequences, we accessed from the EPD website[1] the Homo sapiens data through their EPDnew database, which brings us to hsEPDnew, or the homo sapiens (human) curated promoter database. The current version of this database as of this writing is version 006 with a coverage of 29598 promoters spanning 16455 genes. The genome assembly used for this data is from GRCh38/hg38 with annotations from Gencode version 28. To select the promoters, we used their EPD selection tool with the default properties for selecting all types of human promoters. This selection tool gave us the complete list of 29598 human promoters, which we extracted using their sequence extraction tool from location -249 to +50 for every sequence, resulting in 29598 sequences of length 300 as a FASTA file. The FASTA file format is a popular text-based format for representing either nucleotide or peptide sequences following the standard IUB/IUPAC codes with some exceptions. More information on FASTA files can be obtained from NCBI's BLAST documentation[2]. The sequences from the FASTA file were then randomly sampled to obtain the 7156 human promoters.

Extracting the 5235 human non-promoter sequences took more effort as the download link was not easily accessible. To obtain these non-promoter sequences, we visited the Berkeley Drosophila Genome Project website[3]. We then browsed the Analysis Tools section under Software Tools, where we found M.G. Reese's neural network promoter prediction tool. Their tool contains the training set used for their neural network for both human and drosophila melanogaster. As we are interested only in human promoters, we followed the link to the "Representative Benchmark Data Sets of Human DNA Sequences" where the collection of human and additional eukaryotic promoter regions could be found. These non-promoter sequences contain coding (CDS) and non-coding (intron) sequences from the 1998 GENIE data set. More information on this dataset can be obtained from the README file[4].

5.1.1 Data Encoding

In order for the data to be processed by machine learning algorithms, it must be represented in a state that can be used as an input to the algorithms. The data

[1] `https://epd.epfl.ch/`

[2] `https://blast.ncbi.nlm.nih.gov/Blast.cgi?CMD=Web&PAGE_TYPE=BlastDocs&DOC_TYPE=BlastHelp`

[3] `https://www.fruitfly.org/`

[4] `https://www.fruitfly.org/data/seq_tools/datasets/Human/promoter/README`

obtained in section 5.1 is comprised of string or text-based nucleotide sequences. We have to encode these sequences in a form that conveys their meaning without creating assumptions. For example, a representation method that attaches an assumption to the data is one where each nucleotide is represented by a number, e.g. $A \rightarrow 0$, $C \rightarrow 1$, $G \rightarrow 2$, $T \rightarrow 3$. Nucleotides do not have an order in that way, and thus can be considered nominal or categorical data for our analysis purposes. Qian et al. explored two methods of encoding nucleotides as nominal data [100]. This can be reviewed in Figure 3.4. The first one is their "Element" encoding, which uses a single vector to encode all nucleotides in the sequence. The encoded "Element" vector is then four times the size of the original sequence, where every four positions encodes for a single nucleotide. The second encoding is their "Non-element" encoding, which uses a matrix of channels or rows for each possible nucleotide of the sequence while the columns are indicative of the nucleotide position in the sequence. Their "Non-element" encoding is the same as the widely used encoding named *one-hot encoding*. In experiments described below, we refer to one-hot encoded sequences as "No embeddings".

5.1.1.1 Embeddings

A *word* or a *term* is a sub-string present in a string [82]. In DNA sequences, a term can be at the character-level or at a k-mer level. Each character-level term represents a nucleotide in the sequence, and each k-mer level term represents a k-tuple sub-sequence of contiguous nucleotides in the sequence. Terms have been traditionally encoded as discrete symbols that cannot be compared with one another in a conceptual way [106]. This traditional encoding has the property of having all terms be orthogonal and equidistant to one another, which provides no straightforward way of measuring similarity between terms [97]. The previously described traditional encoding is called *one-hot encoding*. In contrast, *word embeddings* (also known as *distributed term representations*) encode each word or term as a low-dimensional vector [84]. This provides word embeddings the ability of encoding semantic and syntactic similarity in the vector space. An example of word embeddings in the vector space is shown in Figure 5.1.

Neural networks have been used for creating word embeddings, although they are not the only algorithms that have been used for this task. The field that focuses on searching for information in documents, which include the creation of word

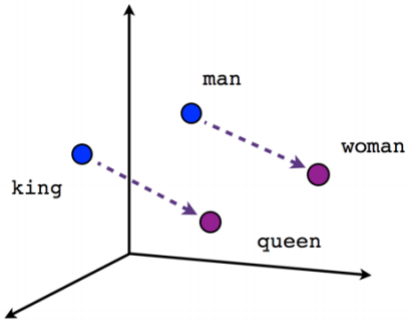


Figure 5.1: Word embedding example showing vector space similarity between terms

embeddings, is information retrieval. A thorough review on this field using neural networks, known as neural information retrieval, was made by Zhang et al. [165]. Neural information retrieval methods typically use a large unlabeled corpora and learn the embeddings in an unsupervised way. For our purposes, we explored two highly successful neural network algorithms made for learning word embeddings.

Mikolov et al. [84] proposed a method called word2vec, which in its initial stage consisted of a three-layer neural network with one input layer, one hidden layer, and one output layer to learn the word embeddings. The authors proposed two variants for the training process to create a language model with the neural network. Both variants take a corpus of the language that has to be modelled and use a sliding window with an arbitrary number of terms to create the training data. *Skip-gram* takes a term from the sliding window to predict the surrounding context or terms. *Continuous bag-of-words* (CBOW) does the opposite of skip-gram, meaning that cbow predicts a term given the surrounding context or terms. A comparative study by Naili et al. [88] concluded that CBOW is more efficient on tasks with frequent terms, while skip-gram is more efficient with infrequent terms.

The training process for word embeddings using neural networks can take a huge amount of computation time. This is why the authors - and other researchers who have trained their word embeddings in a specific corpus (e.g., News articles) - usually release the learned embeddings for others to use without the need to train them. One such model of learned embeddings using a DNA corpus that is useful in our work is *dna2vec* by Patrick Ng [90]. Dna2vec is based on word2vec with the skip-gram variant. The terms dna2vec uses for the training consists of different nucleotide k-mer lengths

($3 \leq k \leq 8$). The training consists of the same neural network architecture as in word2vec. With the learned embeddings, Ng found that the cosine distance between the vectors correlated to the similarity between k-mers on global alignment proposed by Needleman-Wunsch [89].

Pennington et al. [97] proposed an alternative to word2vec called *Global Vectors for Word Representation* (GloVe). Although mathematically similar to word2vec, the optimization or training process is completely different. GloVe learns based on a term co-occurrence matrix and trains the word embeddings so their difference predicts co-occurrence ratios using a log-bilinear model. Instead of a neural network algorithm for its learning, GloVe uses a hybrid machine learning method based on the co-occurrence statistic matrix. But as with other learned embeddings, the embeddings created by GloVe are released and openly shared to avoid the recomputing time. A key difference between these two word embedding models is that instead of the local information from contextual windows used for language modelling by word2vec, GloVe takes into consideration the global count statistics based on word frequency. Empirical evidence [141] shows that both GloVe and word2vec perform roughly the same when used for language modelling tasks.

Language modelling under the natural language processing (NLP) field requires the use of terms to make sense of the language that is being investigated. Using single DNA nucleotides gives us only four terms to decipher the language. However, as with letters in human languages, nucleotides are unable to convey the full meaning of the language; the conglomeration of letters (words and sentences) also play a significant role in conveying information. This is the reason for using k-mers in DNA to provide this hierarchical structure in language. We know that Nierenberg and Matthaei found evidence supporting this language structure within the genetic code in codons [16], which are used for translating RNA into protein. A higher-ordered structure can also be seen in groups of codons that form into completed proteins. Word embeddings take these DNA k-mers and transform them into a vector space where k-mer comparisons and meaning can be derived from these vectors. In section 5.4, we test a few number of k-mers to see which ones represents DNA better. We could only test a few k-mers because the amount of terms increase exponentially as the length of k-mers grow. The amount of terms follow a combinatorial explosion of 4^k, where k is the length of the k-mer. This in turn, increases the computational load for neural networks since a higher amount of terms need to be processed.

5.1.2 Imbalanced data

One way to look at the sentence or higher-ordered structure of DNA sequences is by their differing functionality. For example, some DNA sequences are genes, others are gene regulation sequences such as promoters. The closer we get to finding all higher-ordered structures in DNA, the better we will understand the language of DNA. Currently the effort is in proving machine learning methods for understanding already known structures, but we anticipate that this can be applied to learn new language structures within DNA in the future. A recurring problem within language, and more specifically in DNA, is that the 'sentence' structures we are trying to understand have a very low rate of appearing within corpora in a language (for DNA, the corpora are genomes). In classification problems such as promoter recognition, the unequal rates of the classes (in our case, promoters and non-promoters) refers to *imbalanced data*.

Imbalanced data creates problems when training machine learning models, as the data distributions are heavily skewed towards the majority class. Krawczyk suggested that good results can be obtained, regardless of class disproportion, if all classes are well represented and come from non-overlapping distributions [65]. Regarding promoter recognition, the distributions of k-mers found in promoters and non-promoter sequences overlap and promoters account for less than 0.1% of most organisms' genomes. This extreme imbalance makes the problem difficult to solve.

5.1.2.1 Challenges

In his work about data mining, Weiss [148] discusses problems with imbalanced data and the difficulties it creates. This work translates to machine learning algorithms and as such, we briefly discuss it. The difficulties are categorized into the following:

- Evaluation metrics

  Researchers need to be careful when considering the evaluation metrics that are used to guide the training of machine leaning models.

- Lack of data

  For any problem, the lack of data will impact the training of the models, as the model will not be able to give a precise answer with confidence. This lack of data can come in an absolute sense from not having a representative sample of the class, or in a relative case when a class is minimally represented compared to the other classes.

- Inappropriate inductive bias

  Induction, or generalizing from specific data, requires a set of assumptions to allow machine learning models to be able to predict outcomes from inputs that have not been learned. In other words, these assumptions create a bias within the model. This bias may come in different forms for machine learning algorithms, and for deep learning, it comes in the form of neural network architectures.

- Noise

  Noisy data affects the learning of a machine learning model, but more so when having imbalanced data, as the minority class can get impacted and lost in the noise.

Several methods for addressing these difficulties are discussed by Weiss [148]. More specific methods for addressing imbalanced data in deep learning models are discussed by Johnson and Khoshgoft [65] and Dong et al. [30].

5.1.2.2 Addressing the problem

The methods we explored to address the imbalanced data problem involve sampling. *Under-sampling* eliminates some majority class samples to decrease the variation amount between classes, while *over-sampling* duplicates minority class samples. Over-sampling was used in anomaly detection as a way for learning algorithms to cope with the huge imbalance in classes. It is a technique that puts more emphasis on the minority class by re-sampling it as the model trains, or in other words, duplicating these data points in the dataset. In our case, the chromosomal DNA data has about 50 times the amount of non-promoter sequences as promoter sequences. This technique will penalize the classifier more when making an incorrect choice for the minority promoter class, preventing overfitting of the majority non-promoter class. Both methods decrease the level of class imbalance within the data. The main drawback for under-sampling is the discarding of potentially useful data, which degrades the model's performance. Drawbacks from over-sampling include the increase of training time, as well as the potential to overfit. These sampling methods need to be carefully implemented in the training algorithm as it can skew the direction of the evaluation results if done poorly [136].

5.2 Interpreting deep learning models

Deep learning models are treated like black-box function approximators that map a given input to a classification output. Knowing what the deep learning model is learning on its training process and how the model decides the output depending on the input can help understand and explain the results to get a better intuition about the problem at hand. Typically statistical methods, such as accuracy and precision, are used to evaluate the output's confidence of a deep learning model. This notion of confidence of the model's output should also include the ability of researchers to understand the workings of the model, by making sure that the model is doing what the researcher wants it to do. Being able to obtain the detailed insights into the inner workings of the model makes a model interpretable. Chakraborty et al. [17] summarize a deep learning model's interpretability in two main ways:

- Model Transparency

  includes the ability to manually reproduce every calculation done by the model given the input data. Transparency also includes the ability to gain intuitive reasoning for the model's parameters, as well as the ability to explain the workings of the learning algorithm.

- Model Functionality

  includes the ability to provide a semantically meaningful description of the model's output. Functionality also includes the ability to visualize the parameters of the model to explain how it works, and finally the ability to explain the local changes introduced by specific input.

Recent years have seen an explosion of research for deep learning interpretability. For the most current techniques on machine learning and deep learning interpretability, we recommend to refer to the following reviews [17, 48, 166, 3, 15, 87].

In our work, we focus on interpreting the model's functionality as that would also shine light into the model's transparency by giving intuition to the parameters. The method we use to interpret the model's results and explain what they learn is *explicit attention*. This method is explained in detail below.

5.2.1 Attention models

Neural network architectures can contain attention methods to provide a pathway for information to flow through the network. They were originally designed to allow sequential models such as RNNs to access relevant information by prioritizing a set of positions on the input. Another design goal of attention is to allow deep learning models to process sequential data in a non-sequential order to increase the speed of training. Although attention was not designed to create human-comprehensible explanations, the information pathway they form allows for attention parameters to be used as a form of explanation.

When deep learning was first used for NLP tasks, many different architectures for language understanding and translation started to be proposed. One important architecture is the sequence-to-sequence [21] (seq2seq) model, which consists of an encoder-decoder type of architecture. In other words, the model takes an input sequence and encodes it into a vector. The following part of the neural network works as a decoder by taking the encoded vector and decoding it to a level that is understandable as part of a known language (e.g., back to the original input's language, or translate it into another language). More formally, the encoder is an RNN that takes an input sequence consisting of terms $x_1, x_2, ..., x_T$, where T is the length of the sequence. Each input is then encoded into a fixed length vector ($x_1 \rightarrow h_1, x_2 \rightarrow h_2, ..., x_T \rightarrow h_T$), where the fixed length vectors $h_1, h_2, ..., h_T$ are a form of aggregate of previous vectors, ending with the complete encoded sequence h_T. The decoder is also an RNN which takes the single fixed length vector h_T as its input and generates an output sequence $y = y_1, y_2, ..., y_{T'}$, where T' is the length of the output sequence. The output sequence is created in a similar way to the encoder, except that instead of encoding into fixed length vectors $h_1, h_2, ..., h_T$, the fixed length vector h_T gets decoded into fixed length vectors $s_1, s_2, ..., s_T$ that produce the output sequence y. A depiction of this process is shown in Figure 5.2.

There are two main issues with the seq2seq model. First, the encoder has to compress all the sequence information into a fixed length vector h_T, which may lead to information loss. Second, the seq2seq model is unable to align the input and output sequences, losing structural information. To solve these issues, an attention mechanism was devised by Bahdanau et al. [4]. A simplified example of the attention mechanism is depicted by Chaudhari et al. [18] on Figure 5.3, and consists of an additive attention layer parameterized by a simple feed-forward NN. This means that

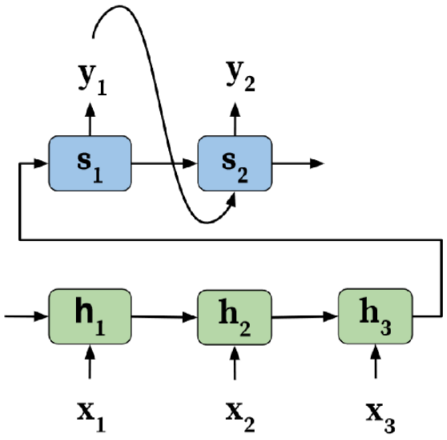


Figure 5.2: Sequence-to-sequence language model depiction by Chaudhari et al. [18]

all the encoded fixed length vectors $h_1, h_2, ..., h_T$ get summed at each decoding step $s_1, s_2, ..., s_T$ to provide the sequence context that was missing from the seq2seq model.

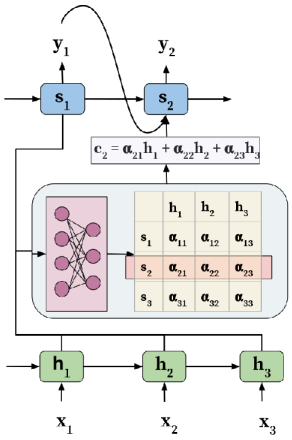


Figure 5.3: Encoder-decoder architecture with attention language model depiction by Chaudhari et al. [18]

The actual encoder-decoder architecture containing an attention mechanism that Bahdanau et al. proposed is called RNNsearch [4]. To produce the context vector c_t, this attention mechanism makes use of an alignment model to obtain a set of scalar values e_{ti} that scores the matching between inputs around position i and outputs around position t. The alignment model is followed by a softmax activation to normalize the scores that are used to produced the sequence context. Figure 5.4 illustrates the architecture of RNNsearch. More formally the context is obtained in

the following manner:

$$
\begin{aligned}
e_{ti} &= f(s_{t-1}, h_i) \\
a_{ti} &= \frac{exp(e_{ti})}{\sum_{j=1}^{T} exp(e_{tj})} \\
c_t &= \sum_i a_{ti} \cdot h_i
\end{aligned}
\tag{5.1}
$$

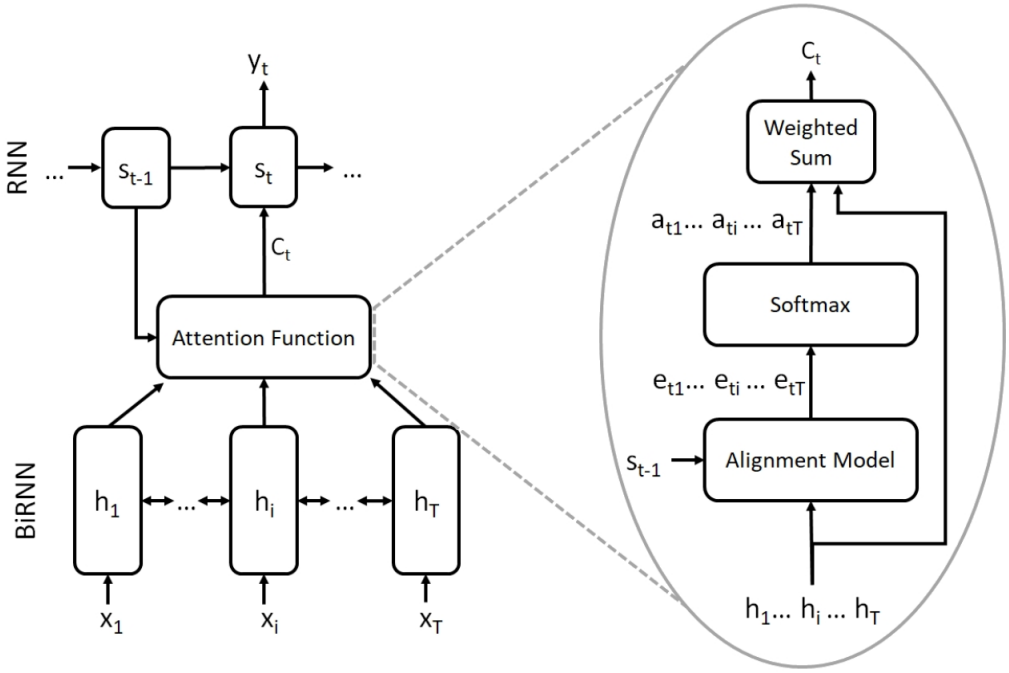


Figure 5.4: RNNsearch architecture with attention model by Bahdanau et al. [4]

There are many different types of attention mechanisms employed throughout the literature. The different attention mechanisms are used for different tasks and a comprehensive review and intuitive explanation of attention in deep leaning can be studied in other works [18, 44, 149].

The problem of machine translation which encoder-decoder architectures are made to solve do not readily work for our promoter classification task. The main difference in the architecture stems from replacing the decoder part of an encoder-decoder architecture by a fully connected layer. This fully connected layer will get its input from an attention layer, which will provide the necessary context when classifying the sequence using an activation function such as softmax. An example of an attention-based RNN

architecture for classification, based on context-aware attention mechanism by Yang et al. [156], is shown in Figure 5.5. In this figure, V_l represents the context vector, which can be interpreted as a fixed query that is randomly initialized and jointly learned with the rest of the attention layer parameters.

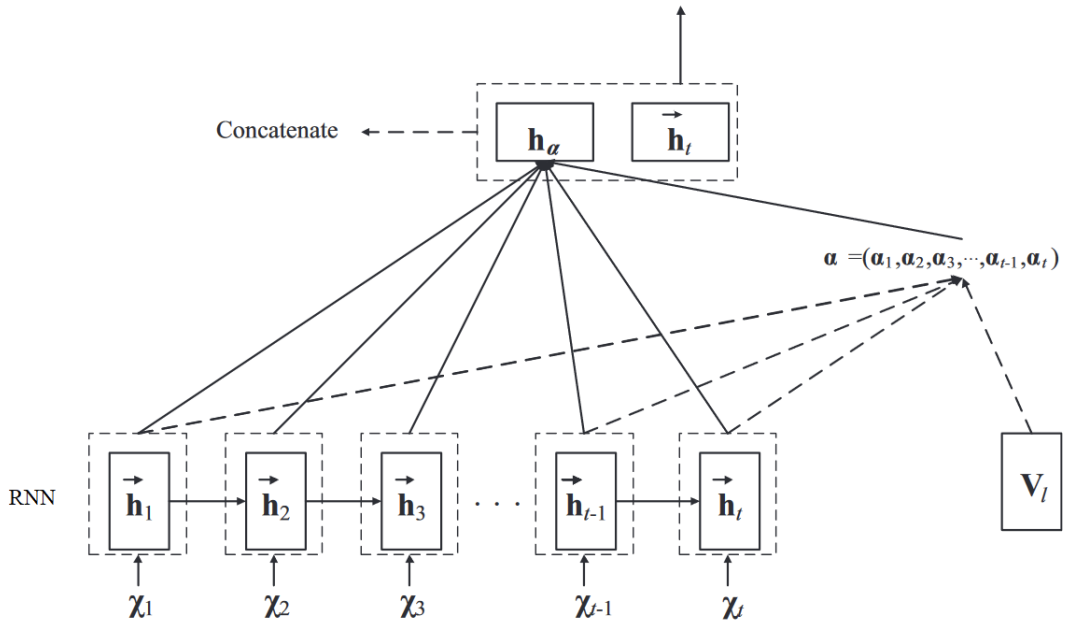


Figure 5.5: Example of an attention-based RNN architecture for classification

Above, we focused on attention methods on RNN architectures. Advances in attention methods for use in CNN architectures have also been explored in the literature. Some reviews are included here [18, 27], with a comparison of CNN and RNN attention based models by Blohm et al. [10].

5.3 Evaluated architectures

For this first approach, we tested four deep learning architectures made for NLP tasks. These models include LSTM, Convolutional-LSTM, Hierarchical Attention Network, and Attention-LSTM. The goal for all models is to predict whether the input sequence was a promoter or non-promoter. The input sequences consisted of strings of 300 nucleotides. Every model was tested with no-embeddings, GloVe embeddings, and dna2vec embeddings. The embeddings were arranged by k-mers ranging from 3-mer to 6-mer. With all the previous combinations, we ended with a total number of 36 models trained.

5.3.1 Long short-term memory

LSTM [56] was introduced previously (see Recurrent Neural Networks subsection of section 2.2.3.2). Recall that LSTM is the part of an RNN architecture that repeats, called a neural network cell. A significant difference from the original RNN architecture is that it keeps track of the cell state (C_t) just as with the hidden state h_t. An LSTM cell is comprised of three types of gates to allow the flow of sequential information in the cell state to be learned by the neural network model.

The first step of the LSTM cell is to decide which information is going to be disposed, which is done by the "forget gate" (f_t). As a cell is repeated through time, every timestep t will have the same process. The forget gate is a layer with a sigmoid activation function which takes as input a hidden state h_{t-1} and the data input x_t. This gate will decide which part of the sequence the cell state keeps as it gets multiplied by the output of the forgot gate. More formally, the forget gate is written as,

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{5.2}$$

The second step of the LSTM cell is to decide what new information will be stored in the cell state. This step is done by the "input gate" (i_t), which works very similarly to the forget gate. The input gate is a layer with a sigmoid activation function which takes the hidden state h_{t-1} and the data input x_t to decide which part of the sequence to keep. The cell also contains a bias b_f, which is learned in the process as any other weight. As these biases are often learned alongside the weight parameters, they are sometimes disregarded in the explanation. This is followed by a layer with a tanh activation function that creates a vector of new candidate values $\hat{C}$ to be added to the cell state depending on the previously computed sigmoid layer. This series of operations will then decide what will be stored in the cell state. The process is formally described as,

$$\begin{aligned} i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\ \hat{C}_t &= tanh(W_{\hat{C}} \cdot [h_{t-1}, x_t] + b_{\hat{C}}) \end{aligned} \tag{5.3}$$

The previous cell state is then updated using the gates previously mentioned in

the following way,

$$C_t = f_t * C_{t-1} + i_t * \hat{C}_t \tag{5.4}$$

The final step of the cell is to decide the output. This output is based on a filtered version of the cell state by using an "output gate". As before, it has a sigmoid layer to decide the parts of the cell state to output followed by a tanh layer to create a normalized vector of the cell state that will be used as the output depending on the sigmoid layer. This output will also be used as the hidden state of the cell in the following timesteps. More formally this is written as,

$$\begin{aligned} o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\ h_t &= o_t * tanh(C_t) \end{aligned} \tag{5.5}$$

5.3.2 Gated recurrent unit

The gated recurrent unit [21] (GRU) model was not part of our the evaluation process, but it is part of a more complex architecture called hierarchical attention network which is explained below. GRU was also previously introduced, and can be considered a variant of LSTM. The GRU cell combines the forget and input gates into a single "update gate." The main difference is that the cell state and hidden states are merged and considered as the hidden state of the cell, which results in a simpler cell with very similar capabilities. A GRU cell can more formally be expressed as,

$$\begin{aligned} z_t &= \sigma(W_z \cdot [h_{t-1}, x_t]) \\ r_t &= \sigma(W_r \cdot [ht-1, x_t]) \\ \hat{h}_t &= \sigma(W \cdot [r_t * ht-1, x_t]) \\ h_t &= (1 - z_t) * h_{t-1} + z_t * \hat{h}_t \end{aligned} \tag{5.6}$$

A comparison of different LSTM variants, including GRU, was made by Greff et al [50]. Their work explains networks using these cells in great detail, and also provide experimental comparisons of the variants in different tasks.

5.3.3 Convolutional long short-term memory

Sainath et al. [112] explored the advantages of both CNNs and RNNs with LSTM cells to come up with a unified architecture of both called *convolutional, long short-term memory, fully connected deep neural network* (CLDNN). For simplicity, we call it *convolutional long short-term memory* (CLSTM). The authors argue that CNNs are good at reducing invariance to small shifts in the data thanks to its local connectivity, weight sharing, and pooling operations. Added to these CNN advantages, LSTMs are useful for temporal modeling. With the great feature capabilities from CNNs, their output can be passed onto LSTMs for modelling the temporal or sequential nature of data. Their work focuses on speech recognition tasks, but also applies to textual data because of its sequential properties. This was further validated in related work to promoter recognition by Oubounyt et al. [93].

In their work, Sainath et al. [112] describe how neural networks outperformed state-of-the art Gaussian mixture models and hidden Markov models. Recall from chapter 2 that those models were previously the state-of-the art for promoter recognition. For speech recognition, neural networks were able to make improvements over these previous state-of-the-art models, with models such as CNNs and LSTMs. The experiments by Sainath et al. showed that the combination of these two models further improves performance. Their architecture consisted of two convolutional layers followed by a linear layer for dimensionality reduction for the output. This linear layer was then followed by a series of LSTM layers and fully connected layers that provided the output of the network. The linear layer was only added to decrease the training computation time and could be discarded, as it did not change the networks classification performance. A depiction of the architecture is shown in Figure 5.6.

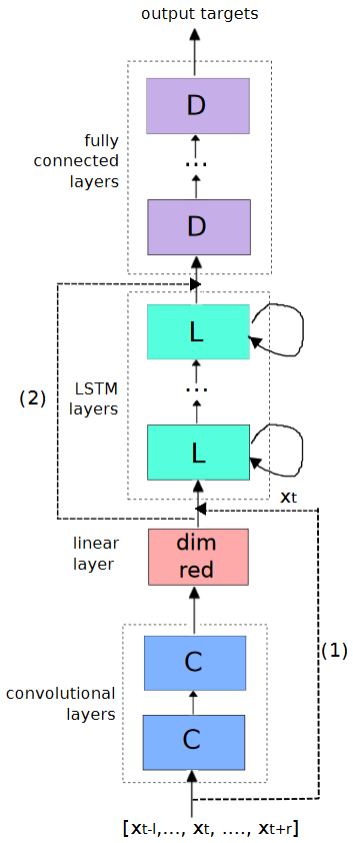


Figure 5.6: CLDNN architecture by Sainath et al. [112]

5.3.4 Attention long short-term memory

In their work for relation classification, Zhou et al. [170] proposed an *attention-based bidirectional long short-term memory network* (Att-BLSTM), which we refer to as attention long short-term memory (ALSTM) for simplicity. The network is made to capture the most important semantic information in sentences. We explore the idea that this architecture could translate to our promoter recognition task when considering DNA sequences as sentences. The architecture consists of an embedding layer followed by a bidirectional LSTM. The bidirectional LSTM is an architecture that makes use of two LSTM sub-networks for the forward and backward input sequence respectively. More formally, the output of the i^{th} term would be obtained by,

$$h_i = [\overrightarrow{h_i} \oplus \overleftarrow{h_i}] \tag{5.7}$$

Here, the $\oplus$ symbol denotes element-wise sum to combine the forward and backward LSTM sub-networks' outputs. The output from the bidirectional LSTM would

then go into an attention mechanism similar to the machine translation attention from Bahdanau et al. [4]. More formally, the attention mechanism accepts a vector $H = [h_1, h_2, ..., h_T]$ (LSTM output) as its input, where T is the length of the input sequence. The final representation h^* of the sentence is formed by a weighted sum of the LSTM output vectors H in the following way,

$$\begin{aligned} M &= tanh(H) \\ \alpha &= \sigma(W \cdot M) \\ r &= H \cdot \alpha \\ h^* &= tanh(r) \end{aligned} \tag{5.8}$$

To make the classification $\hat{y}$ of a sentence S with the sentence representation h^*, a softmax activation layer is used as in Equation 5.9. A depiction of the complete model is shown in Figure 5.7

$$\begin{aligned} \hat{p}(y|S) &= softmax(W_S \cdot h^* + b_S) \\ \hat{y} &= \arg\max_y \ \hat{p}(y|S) \end{aligned} \tag{5.9}$$

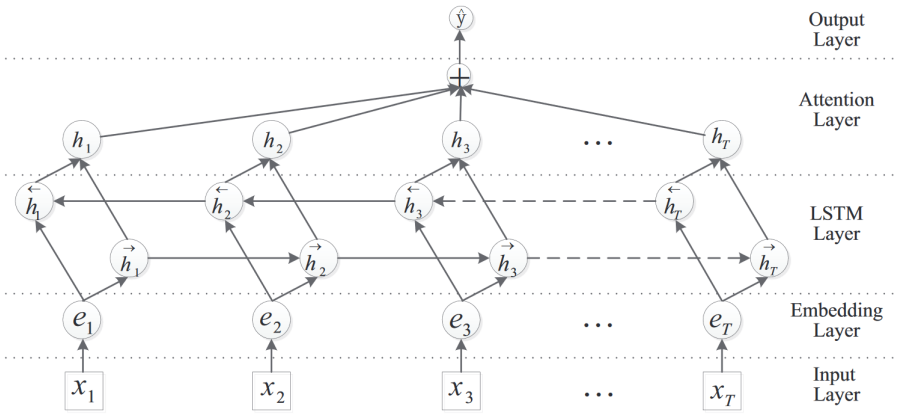


Figure 5.7: Att-BLSTM architecture by Zhou et al. [170]

5.3.5 Hierarchical attention network

Yang et al. [156] proposed a method of document classification by understanding language in different hierarchical structure levels with their *hierarchical attention network* (HAN) model. The hierarchical structure contains two levels of attention mechanisms applied at the word and the sentence level of documents. These levels enable the model to attend differentially to content when constructing the document representation. HAN utilizes GRU as its RNN cells to encode the input sequences or documents. The word level contains an both encoder and an attention part. The encoder part contains a bidirectional GRU to get annotations of words by summarizing information from both directions of a word. Since not all words contribute equally to the representation of a sentence meaning, an attention mechanism is used to extract the important words and aggregate their representations to form a sentence vector s_i. More formally the attention is calculated by,

$$\begin{aligned} u_{it} &= tanh(W_w \cdot h_{it} + b_w) \\ \alpha_{it} &= \frac{exp(u_{it} \cdot u_w)}{\sum_t exp(u_{it} \cdot u_w)} \\ s_i &= \sum_t \alpha_{it} \cdot h_{it} \end{aligned} \tag{5.10}$$

Here, i represents the i^th sentence in the document, and t represents the words in the sentence. Finally u_w is the word level context vector which can be seen as a high level representation of a fixed query asking for the informative words for the word that is being currently processed. This vector is initialized randomly and jointly learned during the training process. The sentence level does the same process previously mentioned for the word level using the sentence vectors s_i as the inputs. The sentence level then outputs a document vector v which is then used to classify the input document with a softmax activation. A depiction of the HAN architecture is shown in Figure 5.8

For promoter recognition, the two-level hierarchical structure of HAN can be used for the single nucleotide level of DNA sequences followed by the k-mer level ($3 \leq k \leq 6$). The nucleotide level would be synonymous to the word level of HAN and the k-mer level would be synonymous to the sentence level.

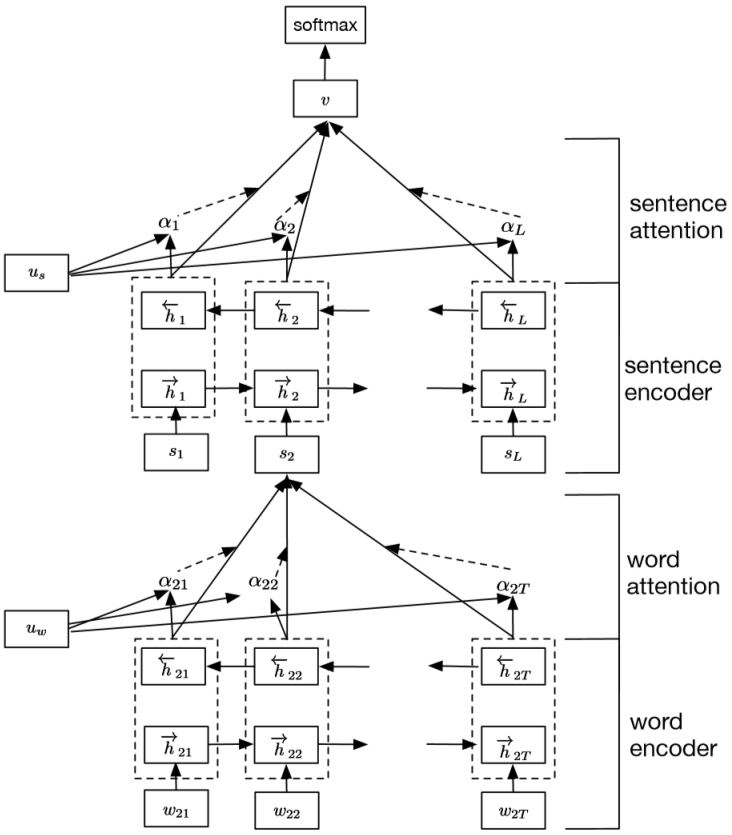


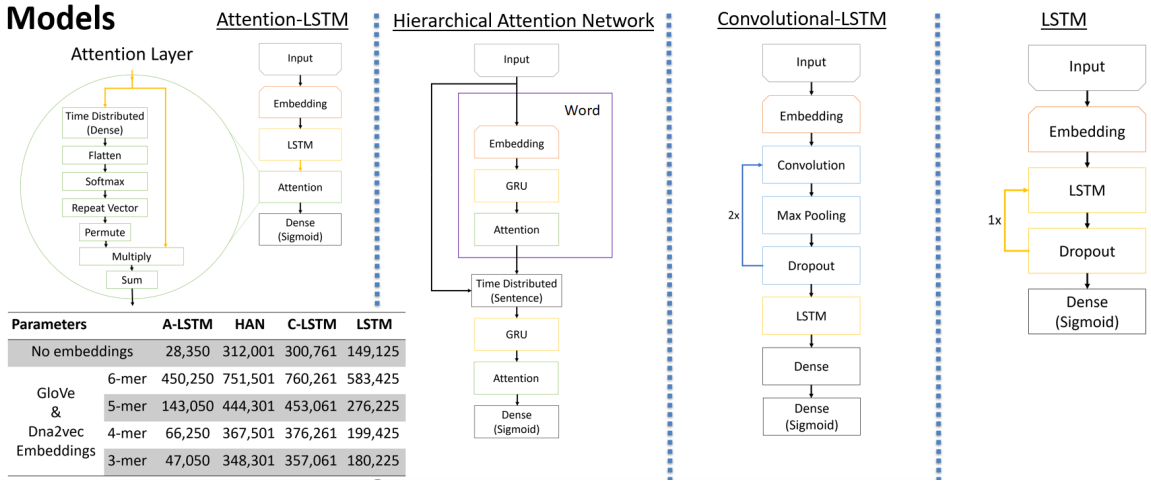
Figure 5.8: HAN architecture by Yang et al. [156]

5.3.6 Implementation

We implemented the models described in the previous section, with the exception of the GRU model, to compare them on the promoter recognition task. All the models implemented contain an embedding layer depending on the embedding used. For the models with no embeddings, the embedding layer was removed and a one-hot encoding was used in place. All the models also utilize the same output layer consisting of a "dense" or fully connected layer with a sigmoid activation function. The sigmoid activation would be used classify the input sequence as a promoter or non-promoter depending on the value. Since the sigmoid output is in the range [0,1], the cutoff threshold chosen for a promoter classification was ≥ 0.5. All models were validated using 10-fold cross validation with the same distribution of training and validation data as Qian et al. [100] with 90% of the promoters and non-promoters as the training dataset and the remaining 10% as the validation dataset. The cross validation process took approximately 12 hours to complete on a regular linux computer with 8 cores

and a graphics card for each of the 36 models.

The LSTM architecture we implemented consists of a bidirectional LSTM layer followed by dropout and the output layer. The CLSTM architecture we implemented is similar to the one described previously but we removed the linear layer between the convolutional layers and the LSTM layers. Specifically our implemented CLSTM architecture consists of three convolutional layers with max pooling and dropout followed by an LSTM layer and a fully connected layer that goes into the output layer. The HAN architecture was implemented as the previously described architecture by Yang et al. with added embeddings. The ALSTM architecture implemented followed the architecture by Zhou et al. that was explained previously. All implemented models are depicted in Figure 5.9. The implementation were done using Keras with a Tensorflow backend. The naming conventions used in Figure 5.9 follow Keras layers API. Keras was used as a first approach for its simplicity and easy to use API that enabled us to implement on the models quickly. This makes implementing the literature models in an uncomplicated manner. The number of parameters in the architectures implemented is also shown in Figure 5.9 to provide an idea of the models complexity.



Parameters		A-LSTM	HAN	C-LSTM	LSTM
No embeddings		28,350	312,001	300,761	149,125
GloVe & Dna2vec Embeddings	6-mer	450,250	751,501	760,261	583,425
	5-mer	143,050	444,301	453,061	276,225
	4-mer	66,250	367,501	376,261	199,425
	3-mer	47,050	348,301	357,061	180,225

Figure 5.9: Depiction of the architectures of our implemented models, with their number of parameters or weights

5.4 Results

This section contains the results from all 36 NLP models implemented, providing an analysis of the four architectures, three embeddings, and four k-mer settings.

First, we have aggregated all models' results into a single chart for a complete visual comparison of these models using different metrics. Results from TensorBoard include the training process for models with different embeddings for comparison purposes. Comparisons of all architectures using four different k-mers are provided to illustrate how the different k-mer settings can affect the models' results. Comparisons between the different k-mers in the four model architectures are also shown to visualize the variability of the models' results depending on the k-mer and type of embedding. Finally, all four models for each embedding have been aggregated for an architecture-agnostic view of k-mer and embeddings. All thresholded metrics use a threshold of 0.50 to separate promoters from non-promoters.

Interestingly, of all the models in the complete comparison, the HAN model did best without embeddings. The hierarchical architecture makes the need for embeddings unnecessary, as the first hierarchical level can function as such. The CLSTM model also exhibits this pattern since the convolutional layer can act as an embedding for subsequent layers. This suggests that convolutional models without embeddings can provide similar functionality to 3,4-mer embeddings, and thus perform accordingly. We believe this happens because the convolution filters act as a form of k-mer embedding.

We now present a note for the bar and line chart figures (Figures 5.10, 5.15, and 5.16) from this chapter. These figures exhibit comparisons between categorical data (models) for different metrics. The thresholded metrics are shown as lines, while the non-thresholded metrics are shown in bars. We decided to use lines instead of dots to make the comparison more visible between the same metric. Each dots represent a value of a metric. The connectivity between them does not make it continuous data.

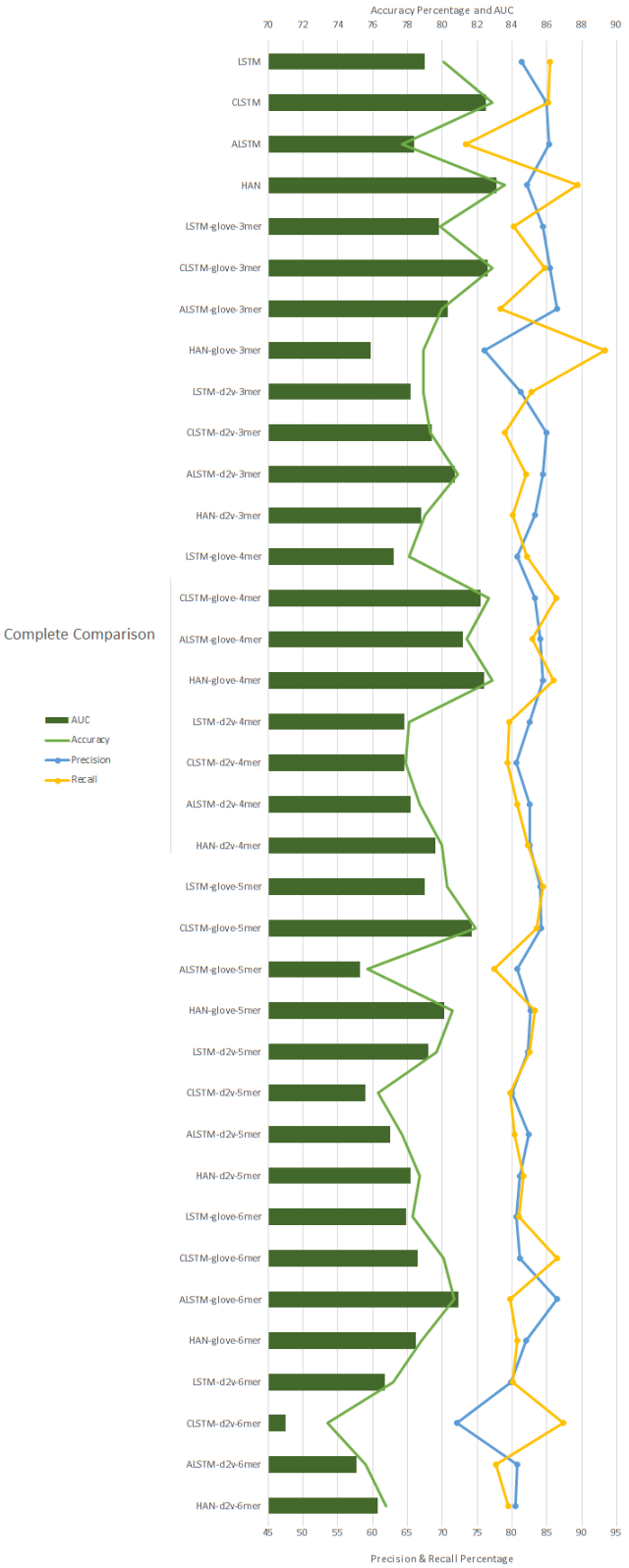


Figure 5.10: Comparisons of all the different NLP models implemented

5.4.1 Models training

Models were trained for 100 epochs. Every epoch trains the model on the complete training dataset. As the method of optimization for neural networks makes use of gradient-based optimizers, the model requires several epochs to approach an optimal solution. Note that models without embedding have slower training, and therefore cannot reach the level of training accuracy and loss as models with embeddings. This performance difference is not visible in the validation dataset, as models with embeddings tend to overfit. The overfitting of the models can be seen since the validation loss tend to increase after several epochs for all embedding types. These results are shown in Figures 5.11, 5.12, 5.13, and 5.14.

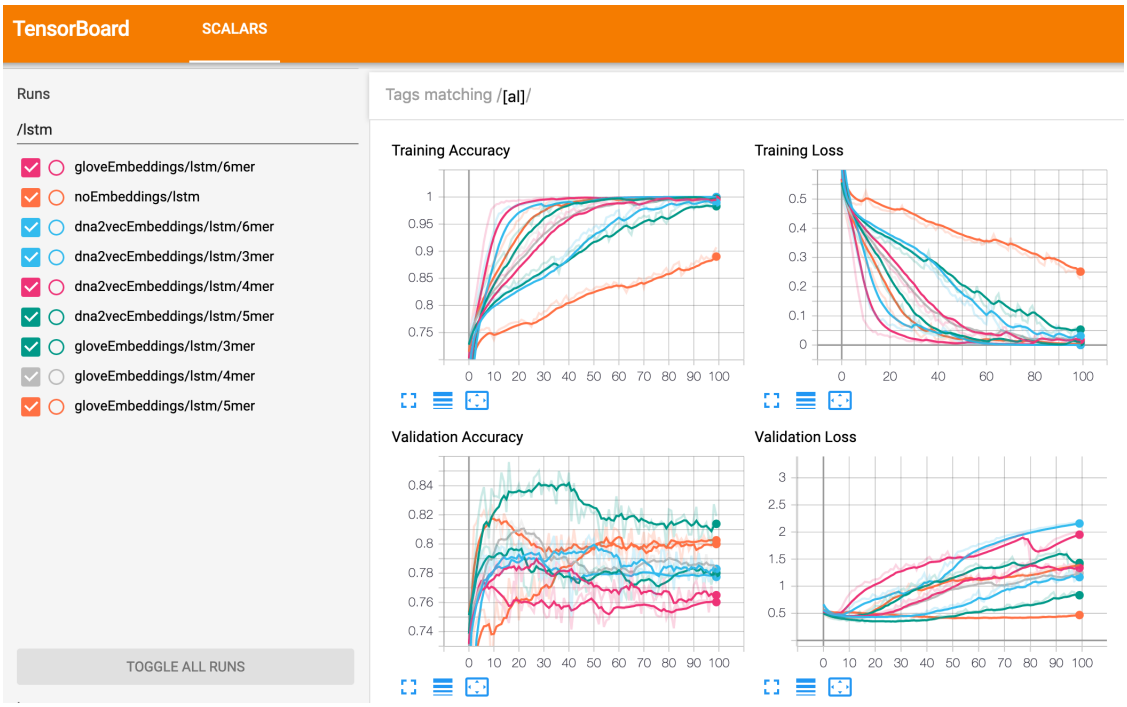


Figure 5.11: Training process and results for LSTM models

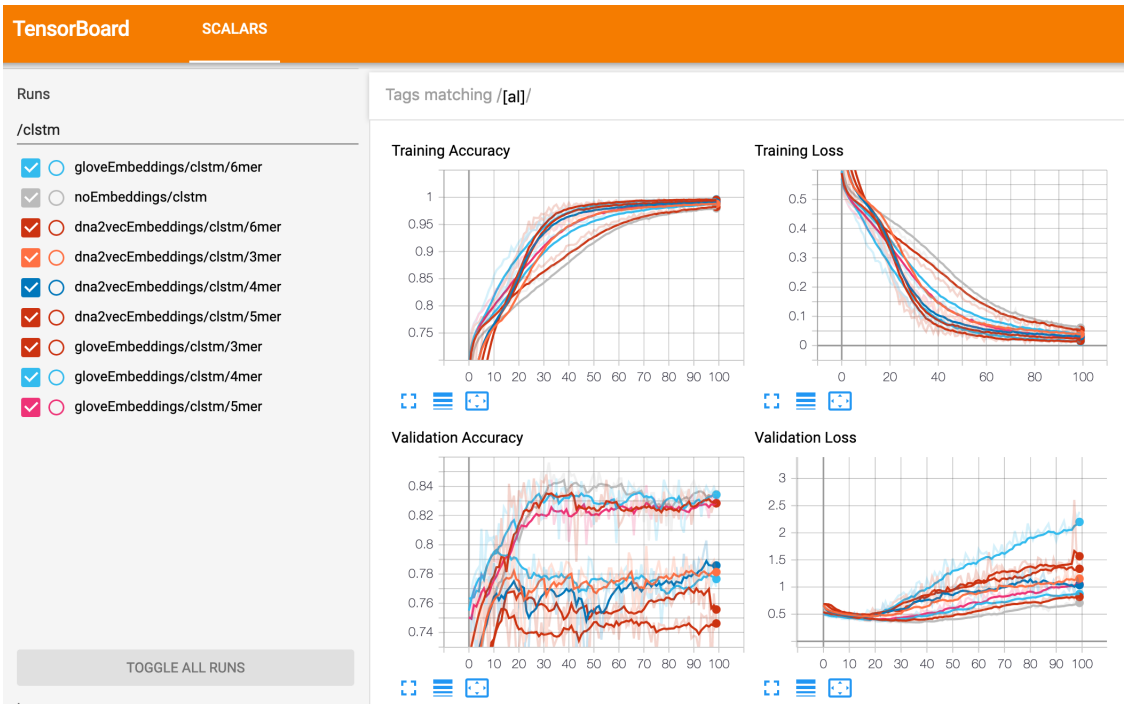


Figure 5.12: Training process and results for CLSTM models

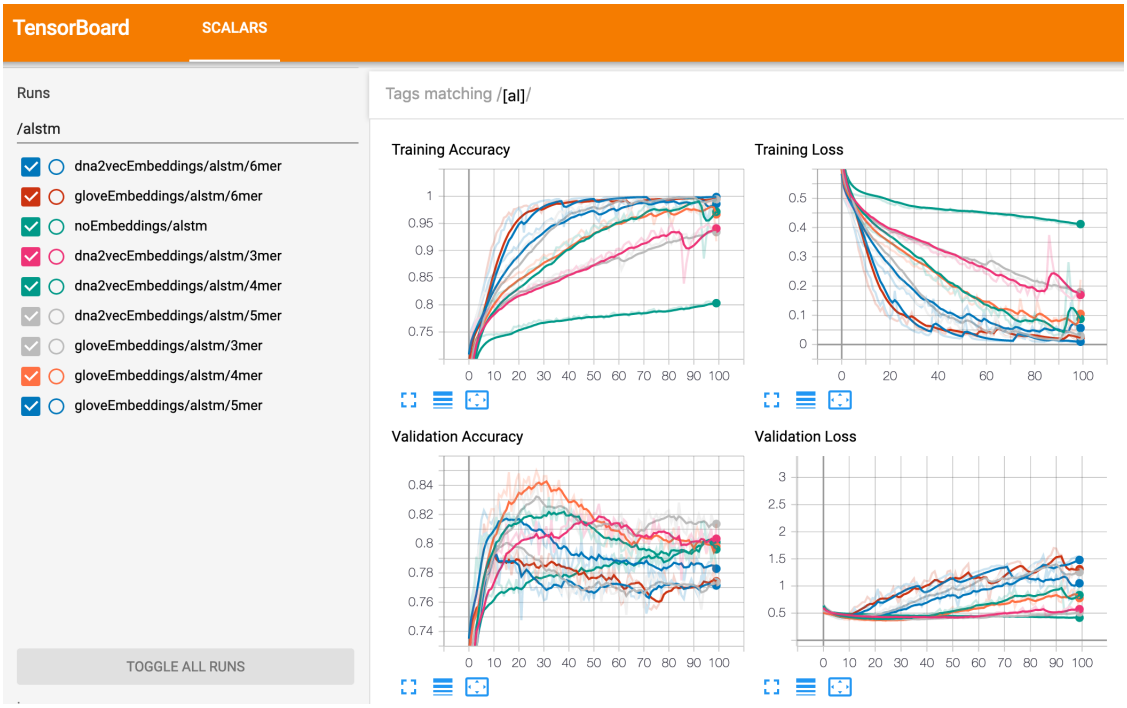


Figure 5.13: Training process and results for ALSTM models

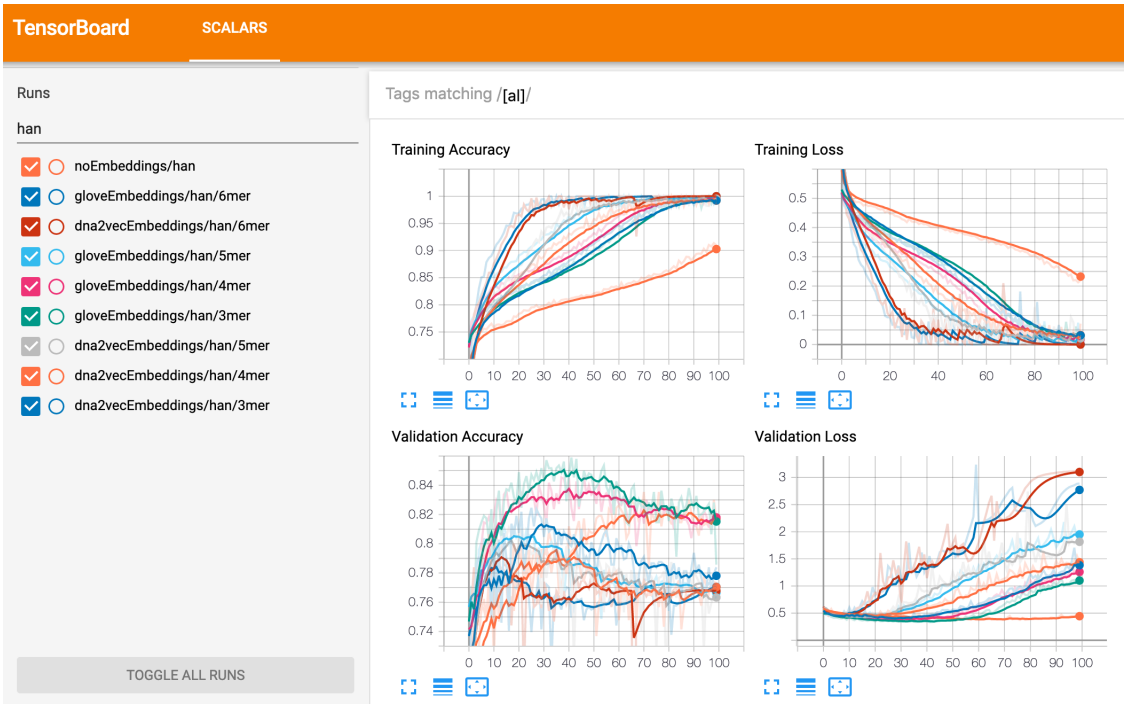
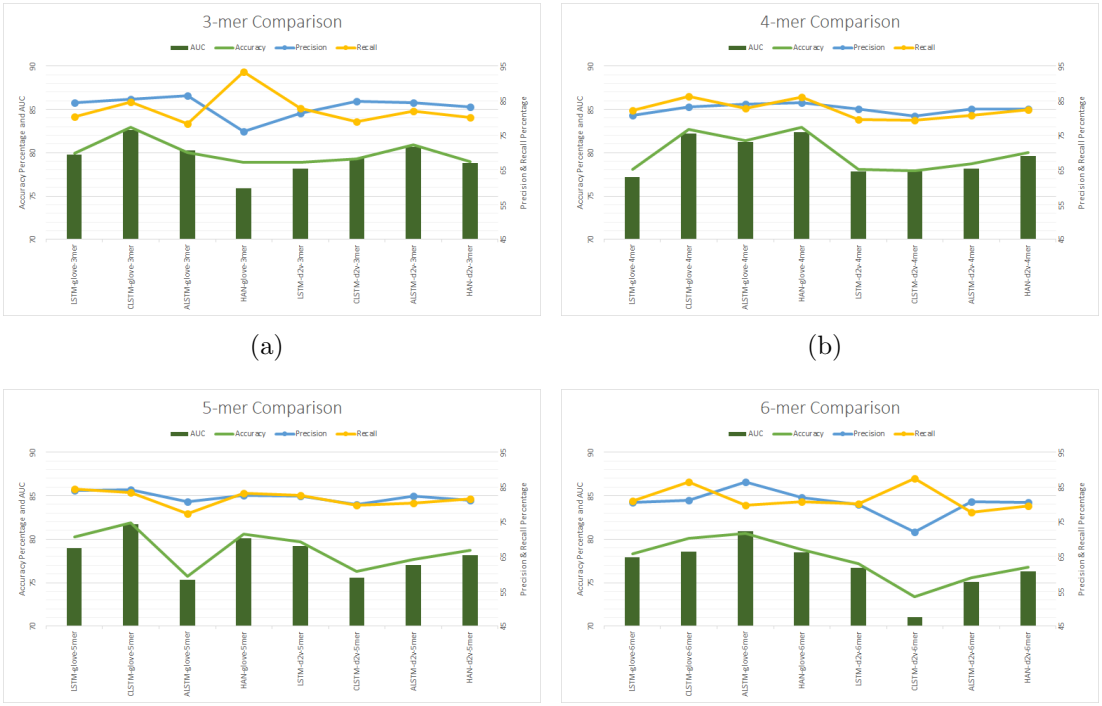


Figure 5.14: Training process and results for HAN models

5.4.1.1 k-mer comparison

This comparison in Figure 5.15 exemplifies that there is no best architecture across all k-mers. The most robust architecture seems to be CLSTM with GloVe embeddings, as it comes first and second in all instances. When architectures do not include embeddings, k-mers are not taken into consideration and thus are excluded. In most of our models, GloVe seems to be the better embedding. This might be in part that the dna2vec embeddings used were pretrained using other dataset sequences, while GloVe embeddings were obtained by processing the dataset sequences used for training the models.



(a) (b)

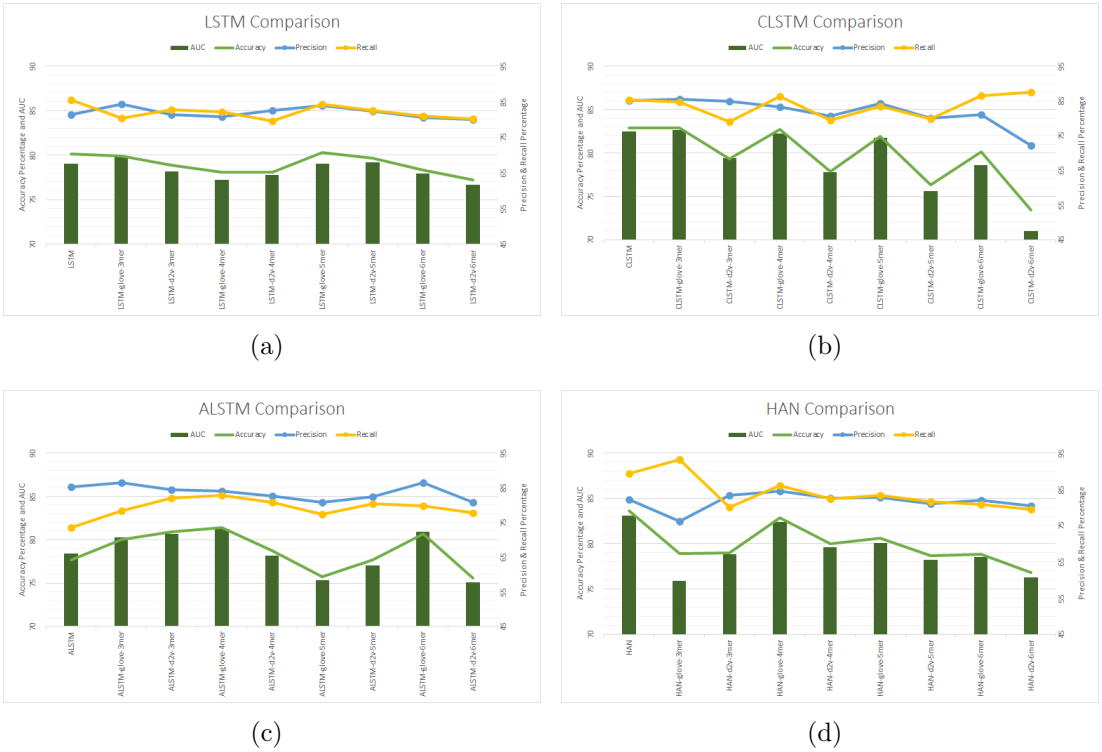
(c) (d)

Figure 5.15: Comparison of every implemented NLP architecture by k-mer embedding: (a) results from 3-mer embeddings; (b) results from 4-mer embeddings; (c) results from 5-mer embeddings; (d) results from 6-mer embeddings.

5.4.1.2 Comparison of model architectures

Architecture specific results in Figure 5.16 show that LSTM is not strongly dependent on embeddings. The reason might be that the LSTM architecture cannot appropriately model this dataset, and therefore reached a limit without the embeddings. Other

architectures contain more variation when using certain embeddings. ALSTM shows higher precision when compared to recall. This is important for promoter recognition, since higher precision of the models lower the presence of undesirable FPs.



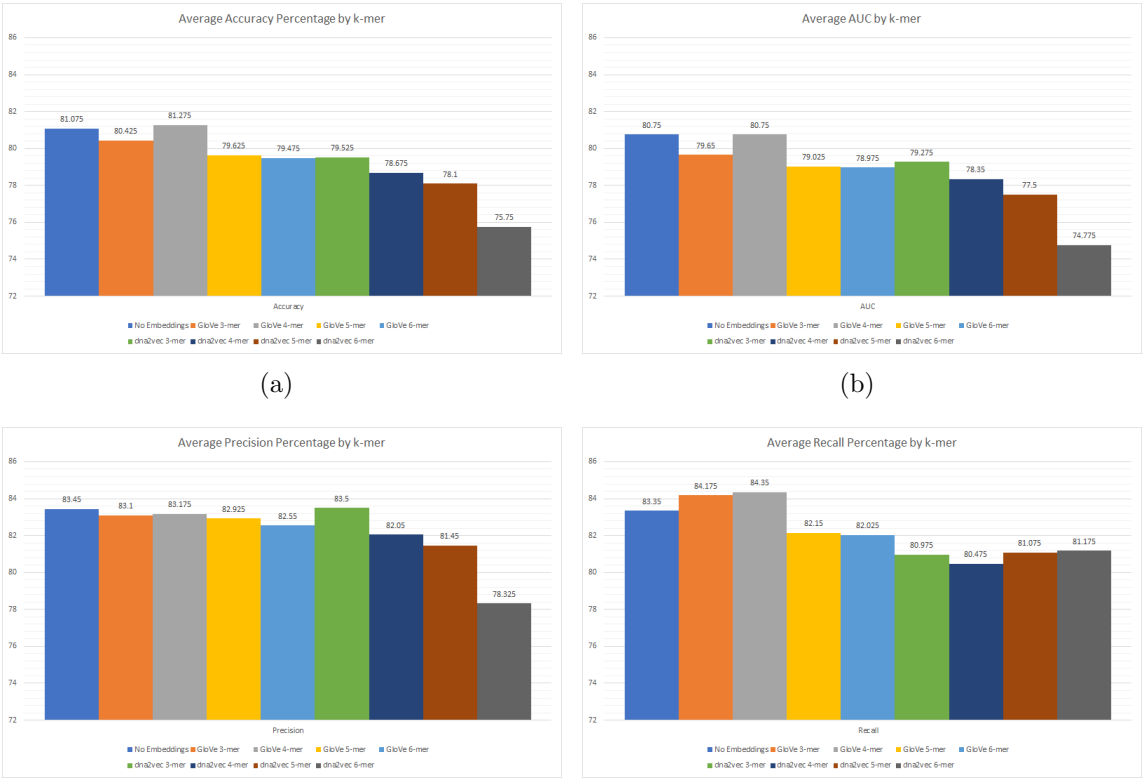
(a) (b)

(c) (d)

Figure 5.16: Comparison of every k-mer embedding by implemented NLP model: (a) results from LSTM models; (b) results from CLSTM models; (c) results from ALSTM models; (d) results from HAN models.

5.4.1.3 Aggregated results

Every embedding and k-mer setting is part of four different architectures. In Figure 5.17, we aggregated the four architecture models to provide an architecture agnostic comparison for the embeddings. GloVe embeddings with 4-mers tend to be the most robust setting, with no embeddings as a close second option. An interesting point is that lower k-mer sizes tend to perform better than higher k-mer sizes across both GloVe and dna2vec embeddings for ab initio promoter recognition.



(a) (b)

(c) (d)

Figure 5.17: Comparison of each model's results aggregated by embedding: (a) accuracy; (b) AUC; (c) precision; (d) recall.

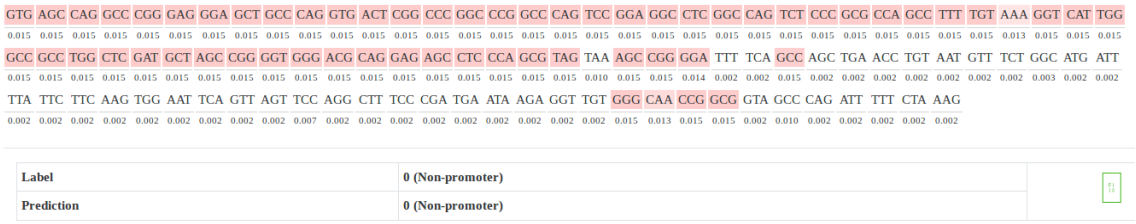
5.4.2 Attention visualization for model interpretability

This section reports some DNA samples that were analysed by the 3-mer GloVe AL-STM model. Two samples cover very confident ($> 99\%$) classifications by the model, one promoter and one non-promoter. Four samples cover regular confidence ($> 50\%$) classifications by the model. Finally, two samples cover borderline classifications, meaning that they were very close to the classification threshold of 0.5. This means that the confidence was very low ($< 5\%$).Recall that the closer the score of the classification is to 1 (promoter) or 0 (non-promoter), the more confident the model is of the classification. In constrast, the closer the classification score is to 0.5, the less confident the model is.

The attention to each k-mer is visualized for each of the DNA samples. Promoters classified with the highest confidence tend to be attended uniformly as a whole. Promoters classified with high confidence tend to be attended higher in the first 200

nucleotides. Likewise, promoters classified incorrectly with high confidence tend to be attended higher in the first 100 nucleotides. Attention spans the range of $[0, 1]$, and we can see in our results that attention is weak (< 0.20) for most k-mers.

The following attention visualizations (Figures 5.18, 5.19, 5.20, 5.21, 5.22, 5.23, 5.24, and 5.25) were made using Neat Vision[5] by Christos Baziotis. Each figure shows the given label y of a DNA sequence, along with the model's prediction $\hat{y}$. On the lower right of each figure, there is coloured square indicating whether the sequence predicted by the model is correct (green) or incorrect (red).



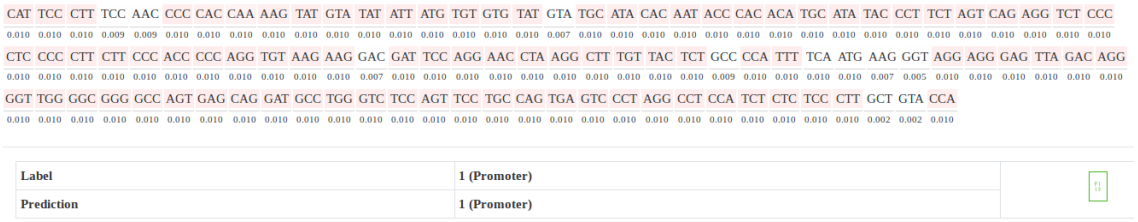
(a)

```
sample_7>HUMPIM1A
-Pim1 oncogene, in healthy cells is involved in cell survival
-conserved across bilateria (symmetrical animals)
-300bp flagged as promoter by ENSEMBL 6:37171492-37171791
-labeled non-promoter, predicted non-promoter with score 0.1329
```

(b)

Figure 5.18: Sample 7 tested on ALSTM 3-mer model: (a) the visualization of the attention mechanism in the model; (b) the report with the description of the sample and its recognition results.

[5] https://github.com/cbaziotis/neat-vision

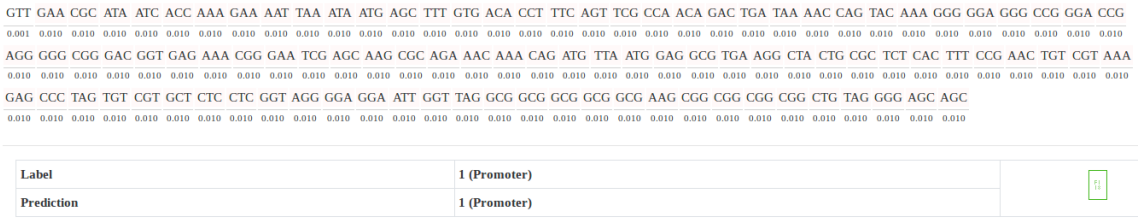


(a)

```
sample_64>FP017913 DGKA_2
-diacylglycerol kinase alpha
-conserved across teolosts (bony vertebrates)
-entire 300bp region is NOT flagged as a promoter by ENSEMBL
    12:55930913-55931212
-however it is within 5kb of a promoter flagged region. DGKA has 49
    transcript variants with a variety of different transcriptional
    start sites.
-labeled promoter, predicted promoter with score 0.5009
```

(b)

Figure 5.19: Sample 64 tested on ALSTM 3-mer model: (a) shows the visualization of the attention mechanism in the model; (b) shows the report with the description of the sample and its recognition results.

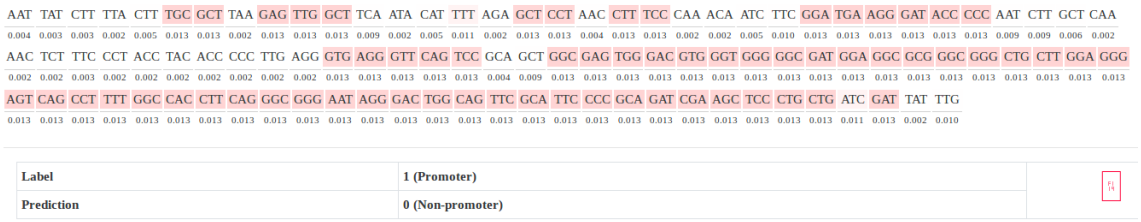


(a)

```
sample_98>FP020583 RNF111_1
-ring finger 111, is a transcription factor involved in signaling and
    development
-conserved across teleosts (bony vertebrates)
-entire 300bp region is flagged as promoter by ENSEMBL
    15:58987446-58987745
-labeled promoter, predicted promoter with score 0.9999
```

(b)

Figure 5.20: Sample 98 tested on ALSTM 3-mer model: (a) shows the visualization of the attention mechanism in the model; (b) shows the report with the description of the sample and its recognition results.

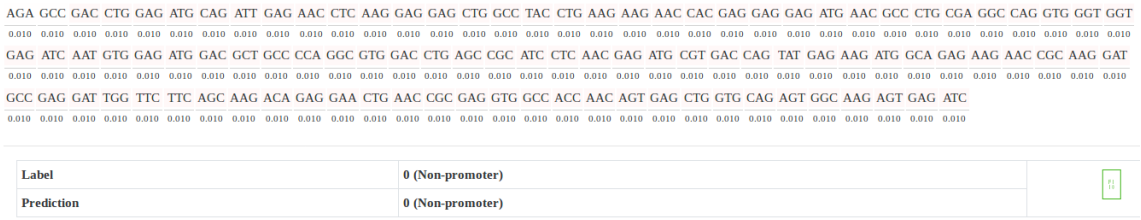


(a)

```
sample_366>FP003412 MEIS1_5
-Meis homeobox 1 gene
-involved in patterning and development, is a transcription factor
    itself
-conserved across amniota (gestate their young inside the body)
-entire 300bp region is flagged as promoter by ENSEMBL
    2:66433172-66433474
-labeled promoter, predicted non-promoter with score 0.1107
```

(b)

Figure 5.21: Sample 366 tested on ALSTM 3-mer model: (a) shows the visualization of the attention mechanism in the model; (b) shows the report with the description of the sample and its recognition results.

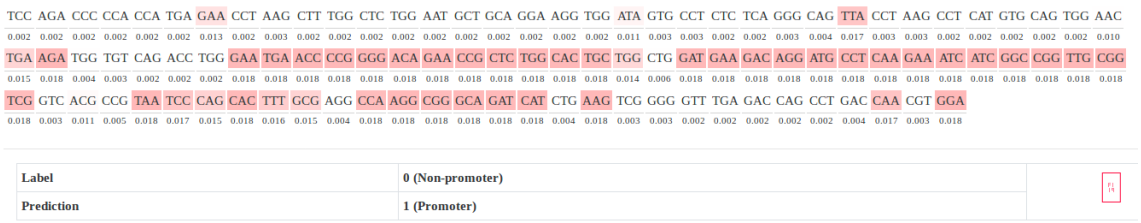


(a)

```
sample_509>HSCYTOK17
-part of the coding sequence for cytoskeletal keratin 17
-within an exon
-conserved across teleosts (bony vertebrates)
-entire 300bp region is NOT flagged as promoter by ENSEMBL
    17:41621527-41621826
-labeled non-promoter, predicted non-promoter with score 0.0003
```

(b)

Figure 5.22: Sample 509 tested on ALSTM 3-mer model: (a) shows the visualization of the attention mechanism in the model; (b) shows the report with the description of the sample and its recognition results.

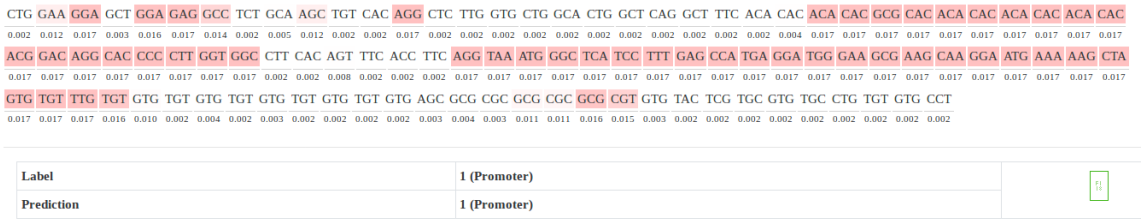


(a)

```
sample_637>HUMRBPA CDS 1
-retinaldehyde binding protein 1, intron
-conserved across teleosts (bony vertebrates)
-entire 300 bp region is NOT flagged as promoter by ENSEMBL
    15:89216519-89216819
-labeled non-promoter, predicted promoter with score 0.9922
```

(b)

Figure 5.23: Sample 637 tested on ALSTM 3-mer model: (a) shows the visualization of the attention mechanism in the model; (b) shows the report with the description of the sample and its recognition results.

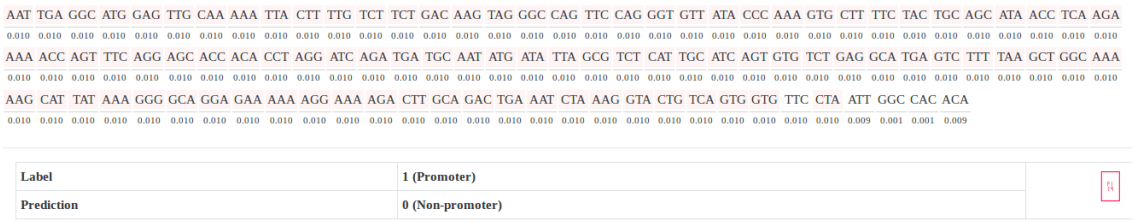


(a)

```
sample_643>FP027086 WISP2_3
-cellular communication network factor 5 (CCN5), in the WNT inducible
    signalling pathway
-conserved across amniota (gestate their young inside the body)
-300bp flagged as promoter flanking by ENSEMBL 20:44715013-44715312
-labeled promoter, predicted promoter with score 0.9867
```

(b)

Figure 5.24: Sample 643 tested on ALSTM 3-mer model: (a) shows the visualization of the attention mechanism in the model; (b) shows the report with the description of the sample and its recognition results.



(a)

```
sample_1146>FP000309 RCC2_3
-regulator of chromosome condensation 2 (RCC2), required for mitosis
-conserved across Eukaryota
-entire 300bp region is NOT flagged as promoter by ENSEMBL
    1:17441425-17741724
-however it is immediately adjacent to a promoter flagged region. RCC2
    has three transcripts, this one noncoding and derived from the 3'
    end of a protein coding transcript.
-labeled promoter, predicted non-promoter with score 0.4938
```

(b)

Figure 5.25: Sample 1146 tested on ALSTM 3-mer model: (a) shows the visualization of the attention mechanism in the model; (b) shows the report with the description of the sample and its recognition results.

Chapter 6

Comparison of Approaches from the Literature

In this chapter, we describe our implementations of the current literature approaches. We specifically investigate DL promoter recognition approaches, for their improvement of the recognition task in recent years. By implementing them, we can analyse and compare them without restrictions, as two of these methods reside as online tools with computational and interface restrictions, and one method does not offer its tool.

Results from the natural language approach showed that current NLP DL models cannot extract additional information from a limited amount of data to give an increased recognition performance in human promoters. Our objective is to find a suitable DL promoter recognition model, and understand how the different models might be able to improve their recognition performance. We approached this objective by exploring and reproducing the work in the literature, as their methods were reported to do a better job at promoter recognition than the models we previously investigated.

6.1 Approaches

For the reproduction of these approaches, we chose the three latest DL papers for ab-initio promoter recognition on human promoters. These three pieces of work are included in chapter 3 and we refer to them by the same names as in chapter 3 for easy reference here:

- Umarov and Solovyev's work (CNNProm) "Recognition of prokaryotic and eu-

karyotic promoters using convolutional deep learning neural networks" [135].

- Qian et al.'s research (ICNN) on "An improved promoter recognition model using convolutional neural network" [100].
- Work by Oubounyt et al.'s (DeePromoter) "DeePromoter: Robust promoter predictor using deep learning" [93].

Commonalities between the three research papers include the use of promoter database (EPDnew) by Dreos et al. [31]. This does not mean that the three papers use the same datasets for training their models. In fact, their models contain different samples from that database for use in their training and evaluation procedures. Sequences from CNNProm were 251 nucleotides long, as opposed to the 300 nucleotide long sequences used by ICNN and DeePromoter. For easier accessibility, we have provided their human promoter evaluation results in the following tables.

CNNProm		
CC	Sensitivity	Specificity
0.895	92.5%	98%

*Results are an average of both human TATA promoters and human non-TATA promoters.

Table 6.1: Evaluation results from Umarov and Solovyev

ICNN		
Accuracy	Sensitivity	Specificity
89.8%	89.6%	86.5%

Table 6.2: Evaluation results from Qian et al.

From Tables 6.1, 6.2, and 6.3 , we can see that it is difficult to compare these methods to select the best human promoter recognition model, since evaluation metrics shown are different for each proposed model. Furthermore, although some metrics are the same, their naming can be confusing. For example, in Table 6.1, CC is the same metric as MCC from Table 6.3. Both of these metrics refer to Matthew's correlation coefficient. Also, the values are incomparable since the results were obtained

	DeePromoter	
MCC	Precision	Recall
0.90	95%	95%

*Results are an average of both human TATA promoters and human non-TATA promoters.

Table 6.3: Evaluation results from Oubounyt et al.

by testing different data samples in terms of both size and quality as noted in chapter 3. As the promoter datasets utilized for the three model's training stems from the same database of promoters, the main difference in the data originates from the non-promoter sequences showing in Table 6.4.

Non-promoter Sequences	
CNNProm	Random fragments of their genes located after first exons
ICNN	Unrelated coding and intro sequences from the 1998 GENIE multiple exon genes set
DeePromoter	Synthetically constructed DNA sequences based on promoter sequences

Table 6.4: Differences in non-promoter sequences for the three DL models being compared

6.2 Comparison of approaches

A direct comparison from DeePromoter and ICNN with CNNProm can be seen from the results in Figure 3.8 and Figure 3.6 respectively. From these results, it appears CNNProm does not perform as well as both DeePromoter and ICNN. The performance difference in these results lie in the testing methodology for obtaining these results. All three models were trained using the data that they released with their published work. The differences takes place in the data used for testing and evaluating the models. DeePromoter and ICNN have an advantage, as the training data that comprise their model come from an overarching dataset that was split for training and testing purposes. The testing sample for both DeePromoter and ICNN is similar to the training sample, thus making the comparison unfair. To fix this, we reproduced

the three model architectures, and trained and evaluated each model using the same data. These models were then compared and their results are shown in Tables 6.5, 6.6, and 6.7.

CNNProm model architecture					
ICNN dataset			DeePromoter dataset		
Balanced Accuracy	F1 Score	Recall	Balanced Accuracy	F1 Score	Recall
91%	93%	95%	59%	68%	88%

Table 6.5: CNNProm architecture by Umarov and Solovyev trained and evaluated on ICNN and DeePromoter datasets

ICNN model architecture					
CNNProm dataset			DeePromoter dataset		
Balanced Accuracy	F1 Score	Recall	Balanced Accuracy	F1 Score	Recall
86%	84%	80%	64%	69%	80%

Table 6.6: ICNN architecture by Qian et al. trained and evaluated on CNNProm and DeePromoter datasets

DeePromoter model architecture					
ICNN dataset			CNNProm dataset		
Balanced Accuracy	F1 Score	Recall	Balanced Accuracy	F1 Score	Recall
52%	73%	97%	50%	61%	96%

Table 6.7: DeePromoter architecture by Oubounyt et al. trained and evaluated on ICNN and CNNProm datasets

6.2.1 Reproduction of approaches from literature

In this subsection we describe the architectures of CNNProm, ICNN, and DeePromoter that were reproduced and compared. CNNProm had different architectures for the training of different types of promoters. As we are focusing on human promoters in this section, we regarded the human TATA and human non-TATA architectures

from Umarov and Solovyev's work. The differences between these two architectures stem from the number of convolutional kernels and the pooling kernel size. We reproduced both architectures and found that the human TATA architecture produced better results consistently across all metrics. Therefore, the results from Table 6.5 stem from testing the human TATA architecture.

Our reproduced human TATA architecture of Umarov and Solovyev's CNNProm is as following:

1. The input sequence is converted into a one-hot encoded matrix.
2. The matrix is passed onto a convolutional layer with 200 kernels of size 21 each, followed by a ReLU activation function and a max-pooling layer with pooling kernels of size four.
3. The output from the pooling layer is passed onto a fully connected layer with 128 neurons and a ReLU activation function.
4. The output from the fully connected layer is passed onto the output layer containing two neurons with a softmax activation function to estimate the probability of the sequence being a promoter.

Qian et al. proposed a single ICNN architecture, which we implemented in the following way:

1. The input sequence is separated into element and non-element sequences.
2. Non-element sequences are encoded into a one-hot encoded matrix.
3. The matrix is passed onto a convolutional layer with 200 kernels of size 21 each, followed by a ReLU activation function and a max-pooling layer with pooling kernels the size of the convolutional layer output.
4. Element sequences are encoded using the procedure described in chapter 3.
5. Both encoded element sequences and the pooling output are concatenated into a single vector which is passed into a fully connected layer with 2048 neurons and a ReLU activation function.
6. The output from the fully connected layer is passed onto the output layer containing two neurons with a softmax activation function.

Similar to Umarov and Solovyev, Oubounyt et al. proposed multiple architectures of their DeePromoter model. This time, we used their human non-TATA model which contained an extra convolutional layer. The reproduced model architecture is described below.

1. The input sequence is converted into a one-hot encoded matrix.
2. The matrix is passed onto a convolutional layer with 32 kernels of size 27, followed by an ELU activation function, and a max-pooling layer with kernels of size six.
3. The pooling output is then passed onto a dropout layer for regularization.
4. Steps 2 and 3 are repeated two more times with convolutional kernels of size 14 and 7 respectively, and everything else staying the same.
5. The output from the previous three dropout layers are concatenated and passed into a bidirectional LSTM layer with 32 nodes or cells in each direction.
6. The output from the LSTM layer is passed onto a fully connected layer with 128 neurons, followed by an ELU activation function and a dropout layer.
7. The output of the fully connected layer is finally passed onto the output layer containing two neurons with a softmax activation function.

The reproduced architectures follow from the architectures they are based on. A depiction for all these architectures can be seen in chapter 3 for better visual understanding.

The reproduction of these architectures was implemented using Pytorch and Skorch. Previously in chapter 5, we had been using Keras with a tensorflow backend to create deep learning models. We switched to Pytorch 1.0, as it became stable coming out of beta testing. Keras provided us with a high-level layers API that could be used in a plug-and-play manner. The model creation was a matter of choosing the layers to be used in a model's architecture, but the training process and development code base became difficult to maintain. Like Keras, Skorch is a high-level framework for Pytorch that brings the training and evaluating code as a complete package. This is because we had to create the complete training and evaluation code for the models in Keras, and as time progressed, the focus slowly switched to maintaining a

working code base instead of researching the deep learning models. We decided to give Pytorch and Skorch a chance, as it was created specifically for rapid prototyping and experimentation through its flexible and productive programming model. Development became easier in Pytorch because unlike Tensorflow, it was designed with a tape-based approach for its AD. Tensorflow was designed with a graph-based approach AD, meaning that the computation graph is statically made and optimized at the beginning of the program, making debugging a laborious process. The tape-based approach from Pytorch creates dynamic computation graphs, and makes debugging a straightforward process while the program is running in real time. More detailed comparisons of these deep learning frameworks have been published by Simmons and Holliday [119] and posted online[1][2].

6.3 Results from comparison

The results previously summarized in section 6.2 are visualized for different metrics in this section. All three models were implemented and trained on their respective data from the literature. Once trained, each of the models were subsequently tested on data used to train the other models. The results are shown in Figures 6.1, 6.2, 6.3, 6.4, 6.5, 6.6. Ideally, we would be seeing similar results for the models since they were trained to solve the same promoter recognition task. Figures include non-thresholded metrics including AUC and PR curve, a comparison between precision and recall, as well as a confusion matrix. The comparison between precision and recall is presented as an alternative view to the PR curve which facilitates the visual comparison between the models. Complete reports including all evaluation metrics from chapter 4 are included in Appendix B.

[1] `https://pathmind.com/wiki/comparison-frameworks-dl4j-tensorflow-pytorch`
[2] `https://www.netguru.com/blog/deep-learning-frameworks-comparison`

6.3.1 CNNProm model tested on DeePromoter data

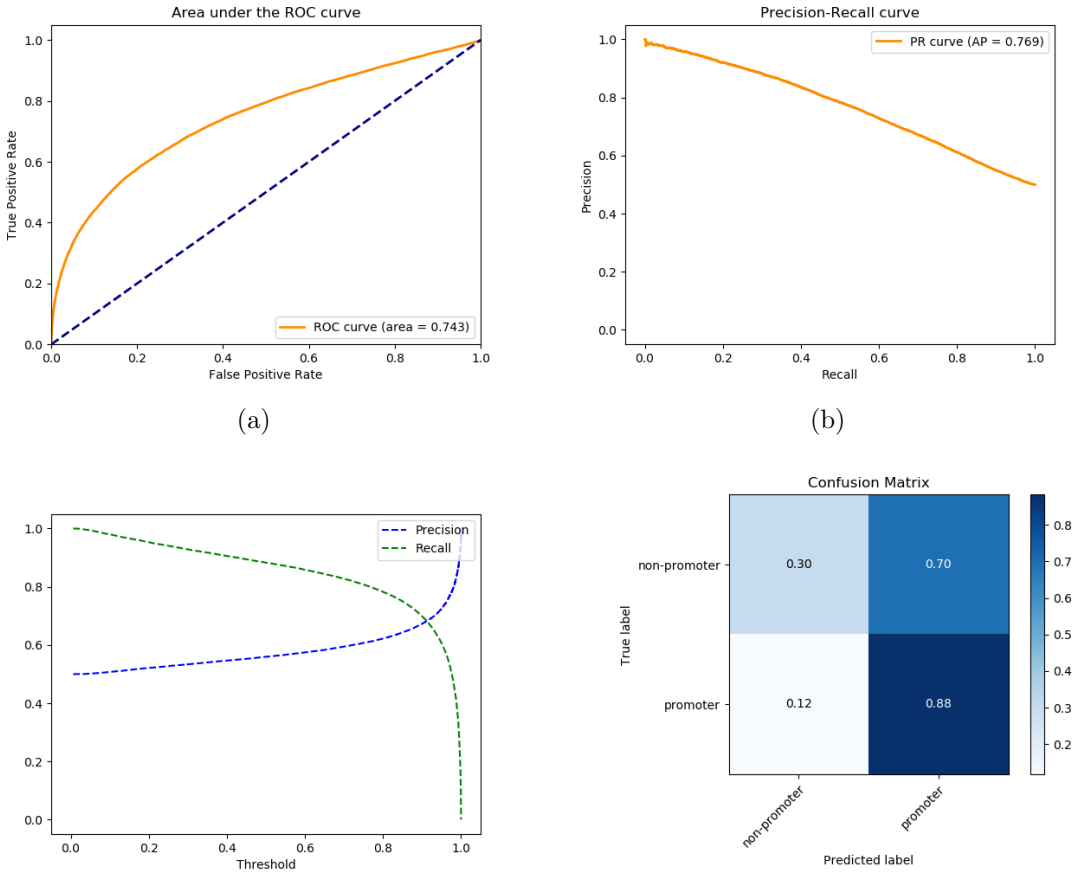


Figure 6.1: Results from the reproduced CNNProm model tested on DeePromoter data: (a) shows the AUC; (b) shows the PR curve; (c) shows precision and recall by threshold; (d) shows the confusion matrix.

6.3.2 CNNProm model tested on ICNN data

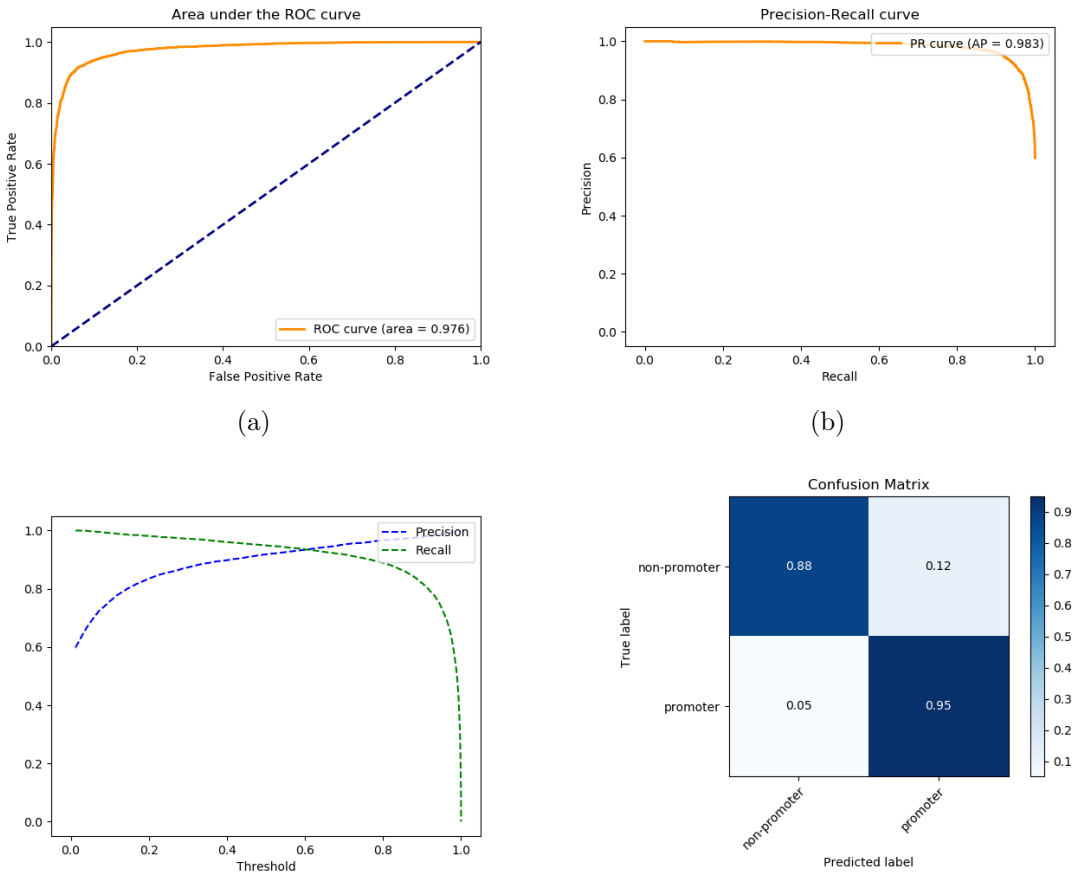


Figure 6.2: Results from the reproduced CNNProm model tested on ICNN data: (a) shows the AUC; (b) shows the PR curve; (c) shows precision and recall by threshold; (d) shows the confusion matrix.

6.3.3 ICNN model tested on DeePromoter data

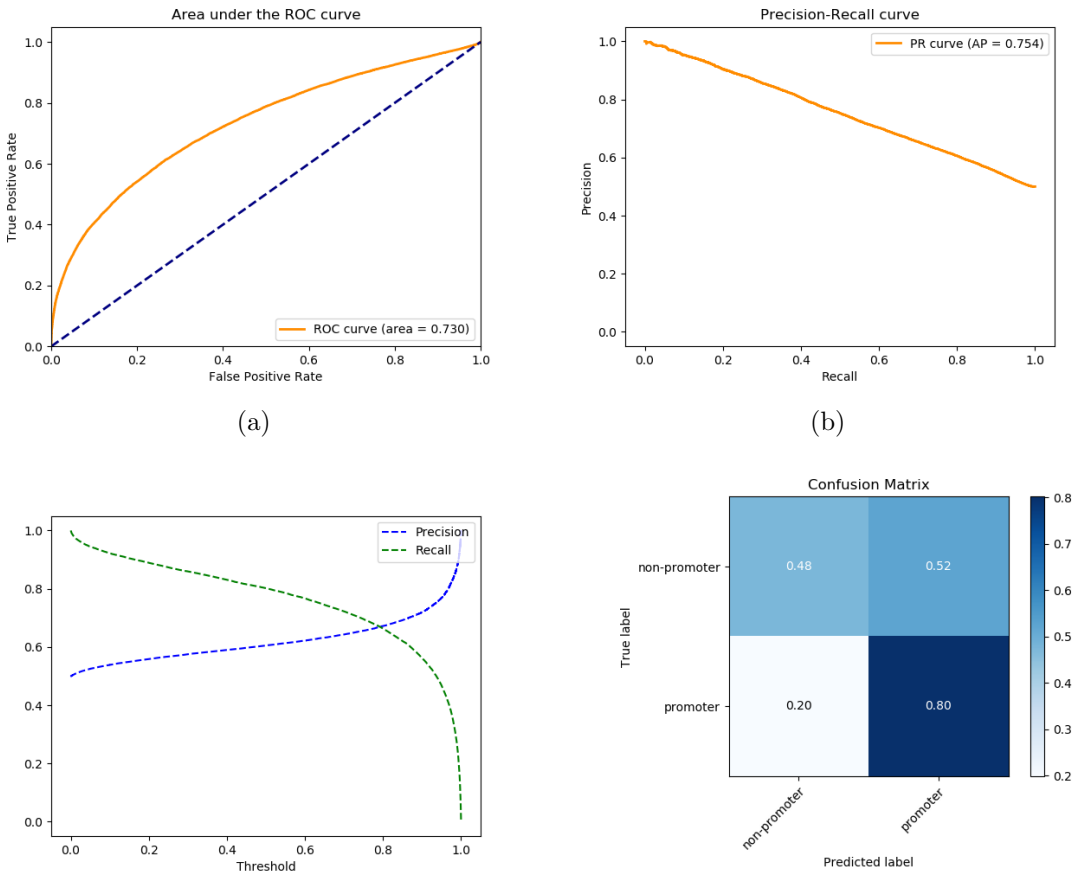


Figure 6.3: Results from the reproduced ICNN model tested on DeePromoter data: (a) shows the AUC; (b) shows the PR curve; (c) shows precision and recall by threshold; (d) shows the confusion matrix.

6.3.4 ICNN model tested on CNNProm data

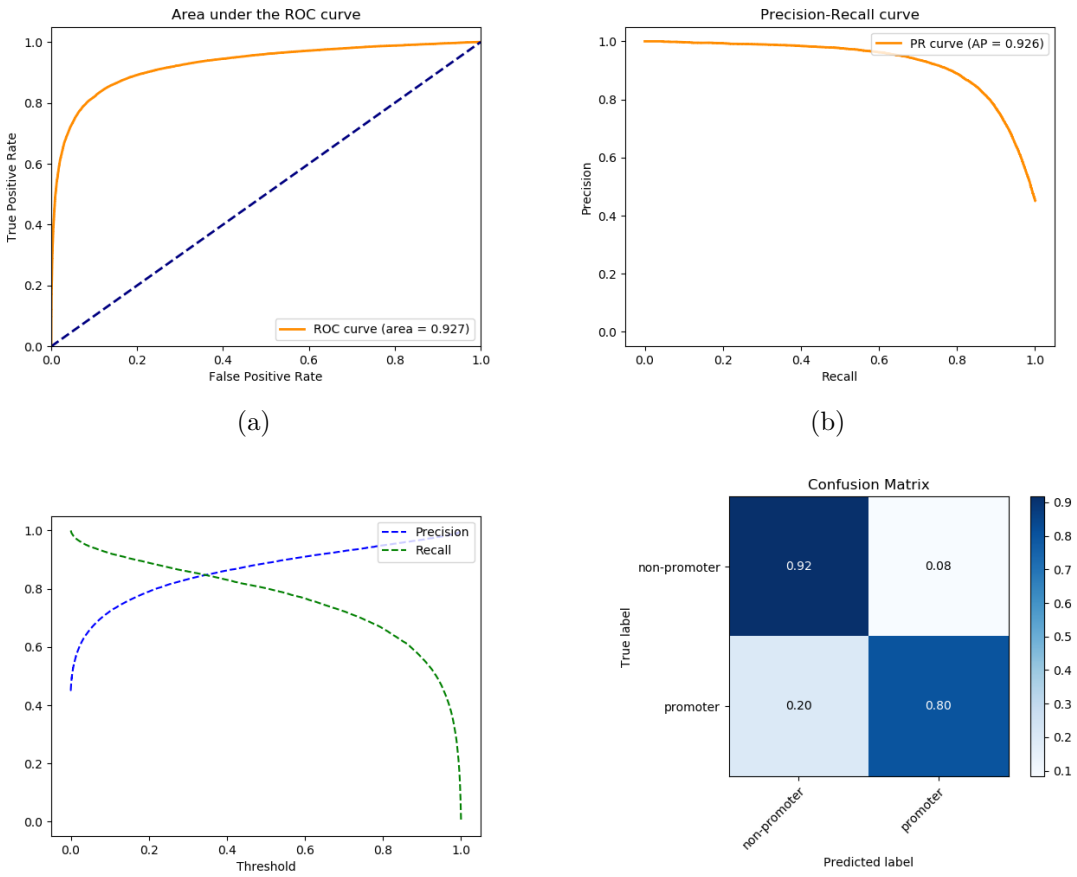


Figure 6.4: Results from the reproduced ICNN model tested on CNNProm data: (a) shows the AUC; (b) shows the PR curve; (c) shows precision and recall by threshold; (d) shows the confusion matrix.

6.3.5 DeePromoter model tested on ICNN data

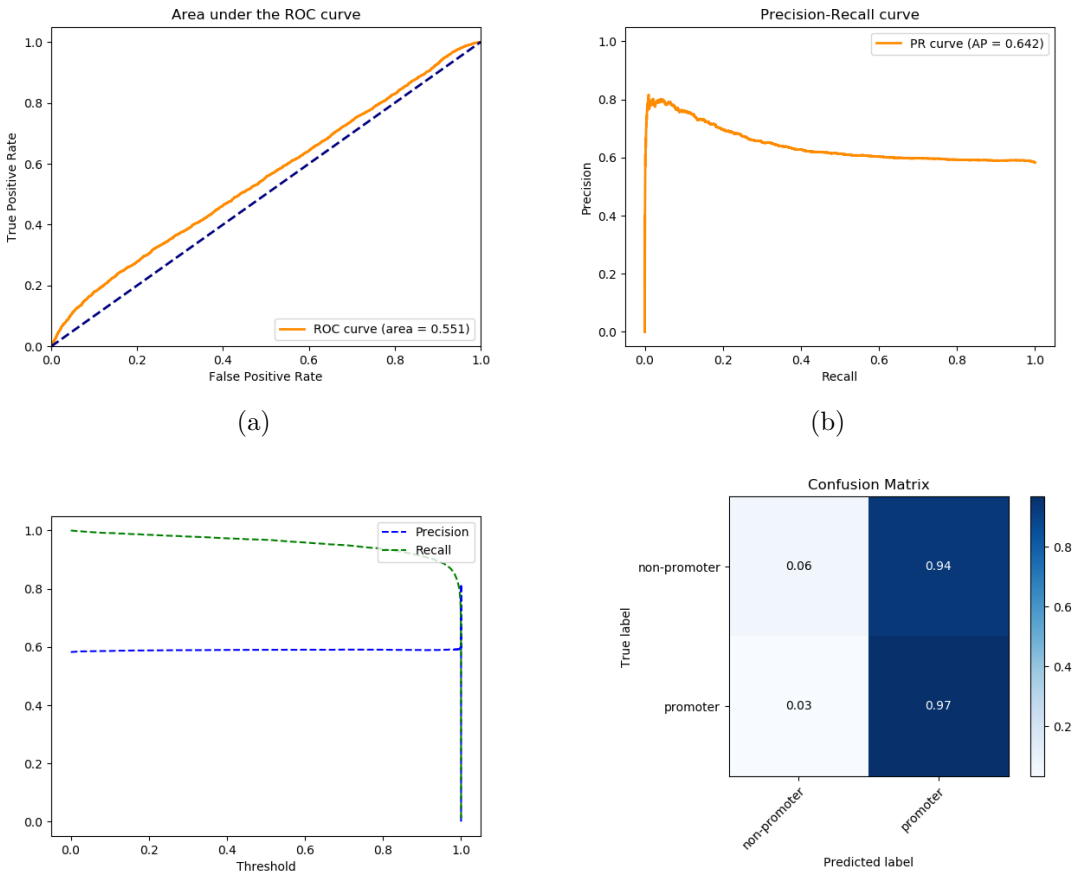


Figure 6.5: Results from the reproduced DeePromoter model tested on ICNN data: (a) shows the AUC; (b) shows the PR curve; (c) shows precision and recall by threshold; (d) shows the confusion matrix.

6.3.6 DeePromoter model tested on CNNProm data

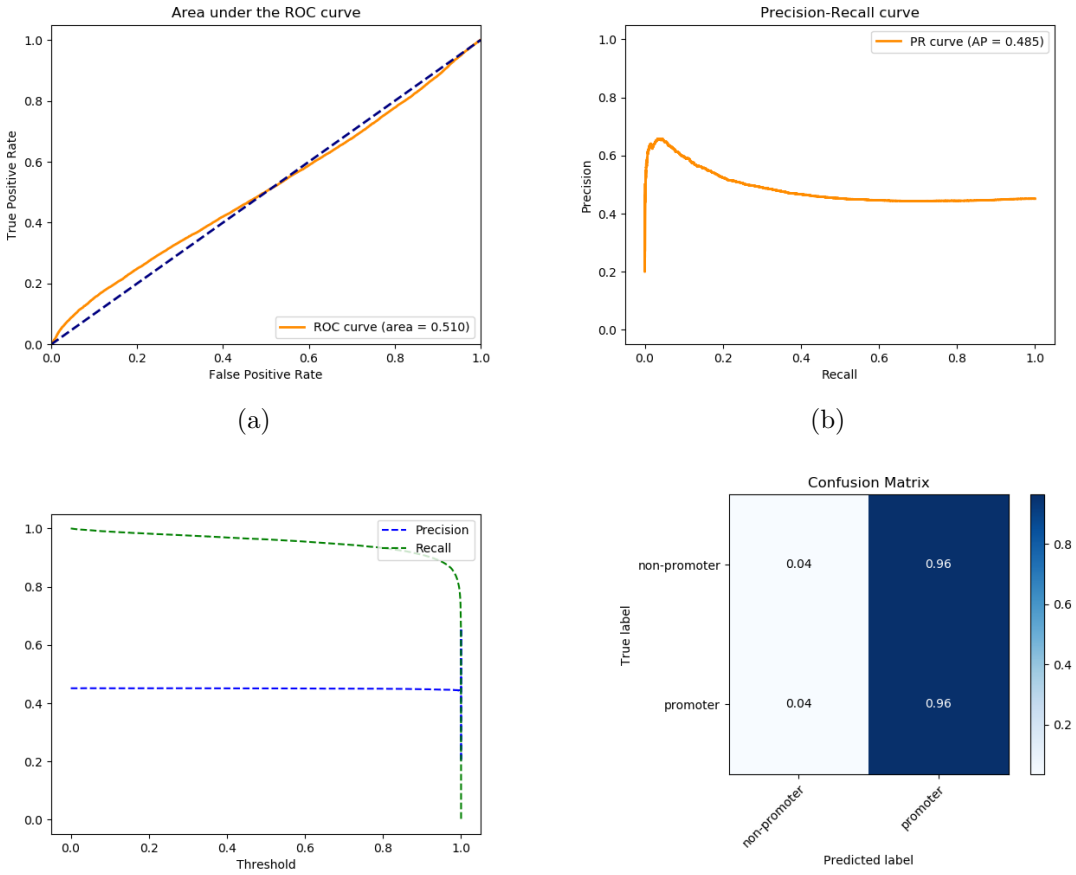


Figure 6.6: Results from the reproduced DeePromoter model tested on CNNProm data: (a) shows the AUC; (b) shows the PR curve; (c) shows precision and recall by threshold; (d) shows the confusion matrix.

6.3.7 Discussion

Note that the promoter datasets for all models are samples from EPD. This means that the training and testing datasets can have overlapping promoter sequences, which inflates the results. This can be especially noticeable in the recall values being fairly high. Low values in the balanced accuracy and F1 score metrics show that the different non-promoter sequences have a big impact on the performance evaluation of the models.

The model that shows the most drastic results come from the DeePromoter model, seen in Table 6.7. This is not surprising, as this model uses synthetically created DNA sequences in contrast to sequences from human organisms. These synthetic sequences are made by random uniform sampling of nucleotides on more than 50% of each non-promoter sequence, which makes it unlikely to be a sequence found in nature. Unlike the synthetic sequences used to train the DeePromoter model, both CNNProm and ICNN models are trained on DNA sequences found in random genetic sequences of the human genome. The use of similar non-promoter genetic sequences for the training of both CNNProm and ICNN mean that both models would perform well on each others datasets. Keep in mind that the performance of the models would be inflated for the reason stated above. It might be confusing to see CNNProm obtaining better results in this case, as opposed to the results shown by Qian et al. in Figure 3.6. Recall that CNNProm's dataset is larger, with a combined number of 57,224 TATA and non-TATA sequences. In contrast, ICNN's dataset is comprised of only 12,391 sequences, making it easier to get a higher score in this smaller dataset. These results are shown in the left part of Table 6.5 and Table 6.6. Most models tested on different data do not work as well as previously reported in their papers, which tells us that the models are not generalizable to recognize promoter sequences. This means that the samples used to train and test the models are either not representative, or the models are not recognizing promoters as intended. It is also probable that some models learned to recognize what is **not** a promoter instead, which would make the models unreliable.

Chapter 7

Testing Methodology

This chapter expands upon chapter 6 by providing a testing methodology for supervised machine learning models on the promoter recognition task. This testing methodology is used to test the three previously implemented DL models from the literature in order to evaluate their performance on the different metrics shown in chapter 4. Recall that our goal is to have a DL PRM that can perform well in genomes that have not been annotated using biological assays. The literature comparison results do not give us a conclusive result on which model could perform competently on this promoter recognition task. We thus created a new testing methodology that can be performed for any organism with a known genome and its promoter annotations as a way of testing and verifying any ML PRM's performance. This testing methodology can be used to detect how well a ML PRM can recognize promoters for a known genome. Once the PRM's performance is deemed sufficient, the ML PRM can be used to recognize promoters in the unannotated genome of organisms with similar promoter structure.

The testing methodology we propose comes from the need to fine or recognize promoters in a sequenced genome of organisms where promoters have not been previously annotated, or are still partially annotated. We provide two methods to utilize our testing methodology. These methods differentiate from one another in their procedure to generate the testing dataset. The first method utilizes sequence alignment while the second uses promoter annotations. The testing dataset created provides an accurate representation of a promoter recognition problem that PRMs are meant to help solve. The test involves using a reference genome and its respective promoter sequences. For our first method, we use the UCSC homo sapiens genome assembly

GRCh38 (hg38)[1] as the reference genome, along with hg38 promoters obtained from EPDnew database[2]. For our second method, we use the previously mentioned hg38 reference genome, along with its promoter annotation from MGA database[3]. There are two methods that create the resulting testing datasets made from a genome and promoter (or promoter annotation) database. They can be created in two different ways, described in the following sections.

In this chapter, the reference genome is usually separated into chromosomes because they can contain differences that might affect promoter recognition. Examples of such differences include:

- Structural properties, including amount of heterochromatin and euchromatin, and other secondary constricts, as observed in satellite chromosomes.
- Chromosome classification, including metacentric, submetacentric, acrocentric and telocentric types.
- Chromosome sizes
- Amount of genes and promoters in the chromosome

These differences might provide minimal variation on promoters, but it is still worth noting them. We focus on human chromosome 1 and 2 for their similar structure, classification, size, and high number of genes and promoters.

7.1 Sequence alignment method

The sequence alignment method was named accordingly by its use of local sequence alignment algorithms. The alignment algorithm requires two files to return a result. The first is a file containing the promoter sequences obtained from a promoter database. The second file contains a reference genome which the algorithm will then align to promoter sequences from the first file. Note that the first file must specifically contain annotations made for the reference genome of the second file; otherwise, the alignment will not work. Here, the promoter annotations we obtained come from EPDnew, which are made for the reference genomes from UCSC and thus deemed

[1] `http://genome.ucsc.edu/cgi-bin/hgGateway?db=hg38`
[2] `https://epd.epfl.ch/human/human_database.php?db=human`
[3] `https://ccg.epfl.ch/mga/hg38/epd/epd.html`

compatible. Our sequence alignment method requires a tool called Basic Local Alignment Search Tool (BLAST), which finds regions of local similarity between both DNA and protein sequences. As we are interested only in nucleotide sequences, we make use of Nucleotide BLAST (NBLAST). NBLAST uses a database of nucleotide sequences to compare to input sequences, which calculates the statistical significance of their similarity. The database we create for NBLAST is based on the promoter database from EPDnew. To run BLAST on a computer, BLAST+ software is necessary and can be obtained from NCBI's website[4]. A user manual for BLAST+ can also be found online at NCBI's website[5]. Our process for creating an NBLAST database from an EPDnew database in BLAST+ and create a testing dataset is the following:

1. Browse the Eukaryotic Promoter Database (EPDnew) website[6] to obtain the promoters of a desired model organism. Note the genome assembly used to create the promoter database of an organism. In our case, we chose the Homo sapiens organism (Dec 2013 GRCh38/hg38).

2. Use the EPD selection tool to choose the promoter characteristics one is exploring. In our case, we explore all possible human promoters, which means leaving all the characteristics on their default setting.

3. Choose the locations relative to the TSS for sequences one wants to extract using the Sequence Extraction Tool. This produces a file in FASTA format that can be downloaded to your computer. We chose to extract sequences from locations -249 to +50, to match the literature [100, 93].

4. After obtaining the promoters FASTA file, use Command 7.1 with the file obtained in step 3 replacing `fasta_file.fa`. Also change the organism to an organism of interest in `organism_promoters`. This will create the NBLAST database needed for sequence alignment.

5. Use our Python script `dataset_blast.py` to create the testing dataset. The script is explained below.

All the source code and files we reference can be obtained online through our GitHub repository[7]. The `dataset_blast.py` script consists of Python code that

[4] `https://blast.ncbi.nlm.nih.gov/Blast.cgi?CMD=Web&PAGE_TYPE=BlastDocs&DOC_TYPE=Download`

[5] `https://www.ncbi.nlm.nih.gov/books/NBK279690/pdf/Bookshelf_NBK279690.pdf`

[6] `https://epd.epfl.ch/`

[7] `https://github.com/ivanpmartell/masters-thesis`

searches for an organism's (by default human) chromosomes in FASTA file format. This can be edited to another organism in the file and is clearly marked on the source code. The search is performed under a `data/` folder. The folder structure must follow `data/organism_chrs/chrname.fa`, with the color red specifying the organism and the chromosome name (e.g. $5, 22, X, Y$) being tested. The NBLAST database should also be created in the `data/blast/` folder. The chromosomes can be obtained websites that contain reference genomes. It is important to note that the reference genome assembly you download is the same as the one obtained in step 1 of the database creation above. We obtain it from the UCSC Genome browser using the provided script: `data/human_chrs/download.sh`.

```
makeblastdb -in fasta_file.fa -title ``Human Promoter
BLAST Database (EPDnew)" -dbtype nucl -out organism_promoters
```

Command 7.1: BLAST+ command to create NBLAST database from FASTA file

The `dataset_blast.py` script does the following:

1. Requires `organism` and `chromosome_name`: the organism, same as the one for which the reference genome was obtained, that will be used to create the testing dataset. Also, the chromosome name that will specify the sequences you are interested on using for testing.

2. Optionally requires `threshold`: the threshold that is used to label sequences. It is a measure of tolerance for the amount of contiguous nucleotides equal to a promoter needed for a sequence to be classified as a promoter. This threshold is used to label sequences as promoters when the amount of identical nucleotides relative to a promoter surpasses the threshold value. If `threshold` is not given, the `promoter_upstream_length` and `promoter_downstream_length` are used to calculate the threshold.

3. Optionally requires `promoter_upstream_length` and `promoter_downstream_length`: the promoter upstream and downstream lengths from the TSS for all promoters sequences. These lengths are used to create a threshold for the amount of identical nucleotides to a promoter, a sequence has to contain in order to be labeled as a promoter. They can be be less than or equal to the values from step 3 of the NBLAST+ database creation. If not given, a threshold should be given.

4. Requires `input_length`: the length of the sequences to create for the testing dataset. This is used to split the reference genome into substrings of the given length. These substring sequences will be used as input to the alignment algorithm.

5. Requires `stride`: the amount of nucleotides shifted after the creation of a substring. The algorithm starts from the beginning of the reference sequence, and in a rolling window manner will splice a sequence of length `input_length`, to obtain a substring that will be part of the testing dataset. The algorithm will move the rolling window by the stride value to splice another sequence and continue doing this until the end of the reference sequence is reached. Therefore, some overlap can occur between the substrings. The total number N of testing sequences will the be given by $N = \frac{L-l}{S} + 1$, where L is the length of the chromosome sequence, l is the `input_length`, and S is the stride value.

6. Calls NBLAST to do ungapped sequence alignment and checks the number of identical nucleotides between two sequences. The first sequence is a substring of the reference genome, and the second sequence is a promoter sequence from the promoter annotations file. This second sequence is obtained by BLAST through the NBLAST database created previously. The alignments will decide the label (promoter or non-promoter) of the input sequence.

To create our testing dataset, we chose human chromosomes as described by the training experiments in Table 7.4 through Table 7.6. For the upstream and downstream lengths, we used the same values from step 3 of the NBLAST+ database creation, thus 249 for upstream length, and 50 for the downstream length, although not required as the threshold was also given. For input length, we chose 300 as per the literature [100, 93]. We set a threshold of 250, meaning that 83% (250/300) of contiguous nucleotides from a testing sequence should be identical to a promoter sequence in order to be labeled as a promoter. As for the stride value, we set it to 300, the same as `input_length` to avoid overlaps.

An advantage of this method is that one can use any promoter database, including gene upstream sequences that can be obtained from the UCSC Genome browser[8]. This method can also be extended to any other DNA regions that want to be studied like other regulatory regions (e.g. enhancers), intragenic regions (introns), and

[8] `http://hgdownload.soe.ucsc.edu/goldenPath/hg38/bigZips/`

intergenic regions. It also expands to promoters with similar DNA that have not yet been experimentally validated by biologists. Occasions where this method is helpful include when the sequences that want to be studied are similar to already known sequences, and the specific locations where they happen in a genome are unknown.

7.2 Annotation database method

When there exists annotation databases with locations of where our sequences of interest happen in the genome, this method can be used to exploit this additional location information to produce the testing dataset in a more accurate and less computationally complex procedure than the previously described sequence alignment method. This is because this method works directly with the locations on the reference genome, skipping the alignment process.

This method is exemplified with the use of the Mass Genome Annotation (MGA) database [32], which was developed by the same people who created EPDnew. Apart from being a repository of multiple DNA assay sample results, the database contains supplementary information for multiple promoter databases in the form of TSS locations. which help us create a testing dataset for promoter recognition. By knowing the TSS that match perfectly to the promoters from EPDnew we are interested in, we can acquire the complete sequence position of a promoter by their upstream and downstream length. Just as with the sequence alignment method, the program requires an organism and chromosome name, the promoter upstream and downstream length, the sequences length, a threshold, and a stride. The annotation database method differs from the sequence alignment method in its use of an annotation file, instead of a file containing promoter sequences that is used to setup the BLAST database. Another difference is accentuated in how the sequences are labeled. Our process for creating a testing dataset using the annotation database method process is the following:

1. Browse the Mass Genome Annotation Data Repository website[9] to obtain the annotation file with the TSS of the promoters of the desired model organism. The data can be obtained in the 'MGA Data Overview' page under Genome Annotation for the desired organism. In our case we chose the homo sapiens

[9]https://ccg.epfl.ch/mga

(Dec 2013 GRCh38/hg38) EPDnew genome annotation relative to version 006. The downloaded file will be a file in SGA format.

2. After obtaining the SGA file, we use our Python script `dataset_parallel.py`, located in the `OURS`, folder to create the testing dataset. This script makes use of parallel computing to increase the algorithms performance and provide results in a short amount of time. A non-parallel version of the script (`dataset.py`) is also available in the same folder. The source code for the script is explained below.

The `dataset_parallel.py` code is written in Python and takes the SGA file from step 1, searching the TSS from the file in previously downloaded chromosomes from a reference genome assembly such as human's hg38. The organism must be supplied as in the sequence alignment method. The reference genome in the form of chromosomes should also follow the same file structure. The chromosomes can be downloaded with the same `data/human_chrs/download.sh` file from before. A promoter upstream and downstream length, used to extract the promoter sequences from the reference genome supplied, should be provided. The `input_length` value that must be given to the algorithm will be used to create the rolling window length for the splicing of the chromosome sequence as done in the sequence alignment method. The stride also follows the sequence alignment method for the rolling window process of creating input sequences. In this method, the threshold t is optional, because it will be equal to $t = u + d + 1$ if not provided, where u is the promoter upstream length and d is the promoter downstream length. The threshold is used as before, in the sequence alignment method, to provide the minimum amount of contiguous nucleotides in a sequence to be considered a promoter.

The algorithm works by creating an array that spans the chromosome's sequence length. When a nucleotide is part of a promoter, the array value at that position will be set to one and otherwise zero. This makes it so that promoter regions within a sequence are covered by ones and non-promoter regions are covered by zeros. The algorithm then takes the sum of the locations of the array that correspond to the nucleotides of the window being processed, deciding whether the sequence is a promoter if the sum is greater than the threshold. An important caveat for this method is that since promoter regions can occur in both the sense and antisense directions, the algorithm can take an overlapping promoter region in both directions to be a single promoter. In some cases, this might be desired as just finding the promoters might

be important, but when directionality is needed, having two separate arrays for each direction might be more suitable. An additional option that could be implemented can make use of the location of the input sequence (rolling window) to set its label. A threshold will determine the amount of overlap that needs to happen between the sequence positions in order for the input sequence to be considered a promoter. The advantages of this method, as stated before, come from being more accurate and less computationally complex while allowing for more flexibility on the algorithm aspect to determine promoter sequences as seen above.

A technique to transition from the sequence alignment method to the annotation database method exists. This involves creating a BLAST database with the reference genome, and using the promoters file as the input query to NBLAST. NBLAST would then align the sequences and return a list of possible candidates, which must be filtered and saved into a file. An example of this process with human chromosome 1 promoters is given in Command 7.2. For more information about the previous command, please refer to the NCBI BLAST manual[10]. The output from NBLAST is passed onto `awk`, which is a text processing scripting language. Here, we use `awk` to filter sequences that contain 300 nucleotides 100% identical to promoters. The numbers can be altered to fit your needs. The result is a promoter annotation file which can then be used on our script `data/blast/transition.py`. This script takes the output from Command 7.2 and transforms it to the same format as the SGA file that the annotation database method requires. In order for the script to perform the transformation, the `downstream_length` for all sequences must be provided. The script's output can then be used as an annotation file for the `dataset_parallel.py` script, with the `downstream_length` following the previously provided value of the transition script.

```
blastn -query human_promoters_chr1.fa -db human_chr1
-outfmt "7 qseqid pident nident sstart send sstrand"
-num_threads 4 -word_size 30 -perc_identity 100
-ungapped | awk '/100.000\t300/' > annotations.out
```

Command 7.2: Command to create a transition promoter annotation file

[10] `https://www.ncbi.nlm.nih.gov/books/NBK279690/pdf/Bookshelf_NBK279690.pdf`

7.2.1 Results on testing database

To compare the models from the literature more thoroughly, we create a testing dataset using the annotation database method. This gives us a promoter recognition problem that mimics a realistic promoter recognition scenario. This scenario occurs when one tries to find promoter sequences in a novel genome. For validation purposes, we need to know what the genome's promoters are. Thus, we acquired part of the human reference genome containing chromosome 1 and 2 from UCSC genome browser. We used the script to create the testing dataset from the reference chromosomes with promoter and non-promoter labels, according to the promoter knowledge from the MGA database. After this, we used the resulting dataset to test the three models from the literature previously discussed: CNNProm, ICNN, and DeePromoter. The summarized results can be seen in Table 7.1, Table 7.2, and Table 7.3. For the complete results on the testing dataset, refer to the figures in Appendix A.

CNNProm model				
Balanced Accuracy	Precision	Recall	F1 Score	MCC
76%	1%	58%	3%	0.085

Table 7.1: CNNProm model by Umarov and Solovyev cross-validated using our testing dataset

ICNN model				
Balanced Accuracy	Precision	Recall	F1 Score	MCC
71%	1%	47%	2%	0.068

Table 7.2: ICNN model by Qian et al. cross-validated using our testing dataset

DeePromoter model				
Balanced Accuracy	Precision	Recall	F1 Score	MCC
48%	0.1%	93%	0.3%	-0.011

Table 7.3: DeePromoter model by Oubounyt et al. cross-validated using our testing dataset

From the above tables, it can be noted that the three models have poor precision in promoter recognition. This means that these models recognize non-promoter sequences as promoters, creating large numbers of false positives. It is also important to note that some models have acceptable Balanced Accuracy values above 70%, but this can be deceiving if other metrics are not taken into account for a complete performance evaluation of the model. By checking all other metrics, we can see that the performances of these three models are inadequate. As noted in chapter 4, it is always preferable to have multiple evaluation metrics that synergize with each other to get a complete view of the model's performance.

7.3 Experiments

Here, we follow up on the results from the previous section with experiments that test different models using the sequence alignment method for obtaining a testing dataset. In this case, only the DeePromoter CLSTM type architecture is tested for computational limitation reasons. This architecture was used for its combination of CNN and RNN layers that proved effective in chapter 5. Note that only the architecture was taken from DeePromoter [93], not the trained model. The models were trained using various datasets, including some testing datasets from the sequence alignment method. The experiments were evaluated using 10-fold cross-validation that takes weeks to complete on our desktop with 8 cores and a mid-range graphics processing unit (GPU) to accelerate the process. These experiments exhibit the same pattern of low precision as previously shown, reiterating the difficulty of the promoter recognition task and demonstrating that the dataset is a more representative sample for promoter recognition. This is demonstrated with the model being trained and evaluated on the same overall data using cross-validation, while still resulting in low precision values. All experiments use 8 folds as the training dataset and 2 folds as the validation dataset. Once trained, the models are tested on different datasets to evaluate their promoter recognition capabilities.

We created 26 separate experiments to comprehensibly test different training and testing criteria. The training criteria are:

- Different promoter annotations; one from EPDnew, and one from UCSC upstream data
- Similar species comparisons; human and mouse promoters

- Different tolerance threshold levels for the sequence alignment method
- Different loss functions involving the output layer; softmax and sigmoid outputs
- Baseline dataset from literature for direct comparison
- Different sampling techniques for training with an imbalanced dataset
- Different types of promoter sequences by motifs; TATA and non-TATA promoters
- Synthetic data nucleotide substitution methods; uniform and original distribution

The testing criteria are:

- Different testing chromosomes
- Different species; human, mouse, and rat
- Baseline dataset from literature

The experiments are numbered by a grouping of similar training or testing criteria. The difference between all experiments lie in the datasets used as input for the models. The numbering system follows a three digit combination. The first digit denotes the type of data used, including chromosomes, literature data, and a mixture of both. The second digit denotes the type of species the data comes from. The third digit denotes the experiment number for differentiation purposes. Experiments with two digits can be viewed as a three digit number where the first digit is hidden, but act as a 1. Experiments are separated into two categories: *training criteria experiments* (TrainEs), and *testing criteria experiments* (TestEs). TrainEs use the datasets as training input for creating a PRM, while TestEs use the datasets as testing input for evaluating a PRM.

TrainEs 01 through 04 contain data from human chromosomes, while TrainEs 03 and 04 make use of oversampling for human promoters to help with the imbalanced promoter dataset. TrainEs 11 through 14 contain data from mouse chromosomes. TrainEs 13 and 14 also make use of oversampling, but for mouse promoters. TrainE 201 contains data on a sample of curated promoters and non-promoter sequences from the literature, specifically the ICNN dataset. TrainEs 301 to 306 contain data on a

curated subset of all known human promoters with synthetic non-promoter sequences from the literature, more specifically the DeePromoter dataset. TrainEs 311 to 316 are the counterparts of the previously mentioned $301-306$ TrainEs containing mouse data. TrainEs 321 and 322 follow the previous TrainEs by using the complete set of human and mouse promoters together. TrainEs 401 and 402 are a mix of human chromosome data and curated human promoter data. Lastly, TrainE 411 is a mix of mouse chromosome data and curated mouse promoter data. All $4xx$ TrainEs make use of oversampling to prevent models from overfitting on the highly larger non-promoter data. In contrast, TrainE 201 can be considered undersampling, as the majority of the genome data accounting for non-promoters has been removed.

TestEs 01 and 02 contain data on human chromosomes 1 and 2 respectively, while TestEs 11 and 12 contain data on mouse chromosomes 1 and 2. TestEs 201, 211, and 221 contain data on human, mouse, and rat promoter data from EPDnew. Rats were used as a testing species because of their related phylogeny to both humans and mice, and to assess how the different PRMs would perform when tested on a related but unknown genome. These species TestEs only contain promoter sequences without non-promoter sequences. Therefore, accuracy, balanced accuracy, Jaccard similarity, and recall all provide the same results, while precision, F_1 score, and MCC do not provide useful results. Finally, TestE 301 contain promoters and non-promoters from the literature, specifically from the ICNN dataset as a baseline in PRM.

7.3.1 Results

The results for the 26 experiments are categorized in the following sections according to the different training and testing criteria set previously. In section 7.3.1.2, we compare the two promoter databases. In section 7.3.1.3, we compare different types of promoters and synthetic non-promoters using DeePromoter data. In section 7.3.1.6, we compare the different loss functions involving the output layer. In section 7.3.1.4, we compare three different sampling techniques used for working with imbalanced dataset. In section 7.3.1.5, we compare models trained on promoters from different species. In section 7.3.1.7, we compare three tolerance threshold levels when obtaining data from the sequence alignment method. The testing criteria are present in all previous sections with the exception of section 7.3.1.6 and section 7.3.1.7, as the results for those criteria were not significant. When testing on the different chromosomes, the data involves using TestEs 01, 02, 11, and 12. When testing on different

species, the data involves using TestEs 201, 211, and 221. As for the baseline testing set, the data involved is TestEs 301.

7.3.1.1 Experiment details

The specific details on each experiment are presented in this section as a reference for the summarized visual results for the experiments in the following sections. Sequences used as input to models for recognition purposes followed a predetermined length of 300 bp. The complete genomic data shown in the experiments were split into sequences of this specified length and subsequently fed into the model. The visuals were created using Power BI[11], an analytics tool that provides interactive visualizations in the way of online dashboards and reports. The complete set of results can be accessed in our Power BI report[12]. The criteria for all experiments follow the description from section 7.3 and are specified in Tables 7.4, 7.5, 7.6, and 7.7.

Training Criteria Experiments	
01	All hg38 chromosomes (1-23 & X and Y) and DNA sequences of length 1000 bp located upstream of hg38 genes
02	hg38 chromosome 2 and EPDnew unique promoters from location -999 to +100
03	hg38 chromosomes (1 & 2), their promoter sequences using the sequence alignment method with five times oversampling, and DNA sequences of length 1000 bp located upstream of hg38 genes
04	hg38 chromosome 1 and promoter sequences from all hg38 chromosomes using the sequence alignment method with ten times oversampling
11	All mm10 chromosomes (1-20 & X and Y) and DNA sequences of length 1000 bp located upstream of mm10 genes
12	mm10 chromosome 2 and EPDnew unique promoters from location -999 to +100
13	mm10 chromosomes (1 & 2), their promoter sequences using the sequence alignment method with five times oversampling, and DNA sequences of length 1000 bp located upstream of mm10 genes
14	mm10 chromosome 1 and promoter sequences from all mm10 chromosomes using the sequence alignment method with ten times oversampling
201	All human promoter and non-promoter data from ICNN [100]

Table 7.4: First part describing the details of training criteria experiments

[11] `https://powerbi.microsoft.com/en-us/`

[12] `https://github.com/ivanpmartell/masters-thesis/blob/master/results.pbix`

[13] Original sequence distribution refers to the ACTG percentage of the promoter sequence

Training Criteria Experiments (continued)	
301	Human TATA-box promoter data from DeePromoter with original sequence distribution[13] for synthetic non-promoter data
302	Human TATA-box promoter and synthetic non-promoter data from DeePromoter. This synthetic data is created using a uniform distribution for replaced nucleotides
303	Human **non TATA-box** promoter data from DeePromoter with original sequence distribution[13] for synthetic non-promoter data
304	Human **non TATA-box** promoter and synthetic non-promoter data from DeePromoter. This synthetic data is created using a uniform distribution for replaced nucleotides
305	All human promoter data from DeePromoter with original sequence distribution[13] for synthetic non-promoter data. This experiment combines both 301 and 303 data together
306	All human promoter and synthetic non-promoter data from DeePromoter. This synthetic data is created using a uniform distribution for replaced nucleotides. This experiment combines both 302 and 304 data together
311	Mouse TATA-box promoter data from DeePromoter with original sequence distribution[13] for synthetic non-promoter data
312	Mouse TATA-box promoter and synthetic non-promoter data from DeePromoter. This synthetic data is created using a uniform distribution for replaced nucleotides
313	Mouse **non TATA-box** promoter data from DeePromoter with original sequence distribution[13] for synthetic non-promoter data
314	Mouse **non TATA-box** promoter and synthetic non-promoter data from DeePromoter. This synthetic data is created using a uniform distribution for replaced nucleotides
315	All mouse promoter data from DeePromoter with original sequence distribution[13] for synthetic non-promoter data. This experiment combines both 311 and 313 data together
316	All mouse promoter and synthetic non-promoter data from DeePromoter. This synthetic data is created using a uniform distribution for replaced nucleotides. This experiment combines data from experiments 312 and 314

Table 7.5: Second part describing the details of training criteria experiments

Training Criteria Experiments (continued)	
321	All human promoters and non-promoters using both methods of synthetic data creation. This experiment combines data from experiments 305 and 306
322	All human and mouse promoter and non-promoter data from DeePromoter. This synthetic data is created using a uniform distribution for replaced nucleotides
401	hg38 chromosome 1 with all human promoters and non-promoters from DeePromoter, and promoter sequences from all hg38 chromosomes using the sequence alignment method with ten times oversampling
402	hg38 chromosome 1 with all human promoters and non-promoters from ICNN, and promoter sequences from all hg38 chromosomes using the sequence alignment method with ten times oversampling
411	mm10 chromosome 1 with all mouse promoters and non-promoters from DeePromoter, and promoter sequences from all mm10 chromosomes using the sequence alignment method with ten times oversampling

Table 7.6: Third part describing the details of training criteria experiments

Testing Criteria Experiments	
01	Complete hg38 chromosome 1 with its promoter and non-promoter sequences using the sequence alignment method
02	Complete hg38 chromosome 2 with its promoter and non-promoter sequences using the sequence alignment method
11	Complete mm10 chromosome 1 with its promoter and non-promoter sequences using the sequence alignment method
12	Complete mm10 chromosome 2 with its promoter and non-promoter sequences using the sequence alignment method
201	Dataset of only **human** promoters from EPDnew database. This means non-promoter sequences are not part of this testing dataset
211	Dataset of only **mouse** promoters from EPDnew database. This means non-promoter sequences are not part of this testing dataset
221	Dataset of only **rat** promoters from EPDnew database. This means non-promoter sequences are not part of this testing dataset
301	ICNN dataset containing promoter and non-promoter sequences from the literature

Table 7.7: Details of testing criteria experiments

7.3.1.2 Comparison of promoter annotation datasets

Promoter annotation datasets contain promoter sequences that have been biologically verified. EPDnew database has been used in most works from the literature, as can be seen in chapter 3. EPDnew collects data from multiple sources, including ENCODE and FANTOM. UCSC offers a genome browser, which contains upstream sequences for every biologically verified gene in multiple organisms. These upstream sequences can be used as promoter sequences since they are adjacent to the TSS.

The TrainEs are grouped into two disjointed sets according to the promoter annotation dataset they are based on. TrainEs 02 and 12 are grouped into the EPDnew set. TrainEs 03 and 13 are also grouped into EPDnew, but using oversampling. TrainEs 01 and 11 are grouped into the UCSC upstream set. Finally, TrainE 201 is used as a baseline for comparison with the literature as the ICNN dataset.

Figure 7.1 shows the results from different promoter annotation datasets being tested on the baseline ICNN dataset. As expected, the set trained on the baseline ICNN data does the best in all categories, since the model was trained on the same dataset it is being tested on. It is interesting to note that out of the other sets, precision is best for UCSC upstream set, while recall is best for EPDnew.

Figure 7.2 shows the results from different promoter annotation datasets being tested on the full human and mouse chromosome data. Here, the baseline ICNN data does the worst, while the other three sets follow the same pattern as in Figure 7.1. Overall, the UCSC upstream set seems to do best when tested on full chromosome data.

Figure 7.3 shows the recall from different promoter annotation datasets being tested on different species. It is interesting that the values are highest for the rat dataset, which means that the results translate well in related species for all promoter annotation types.

Figure 7.1: Results from different promoter annotation datasets being tested on the baseline ICNN dataset

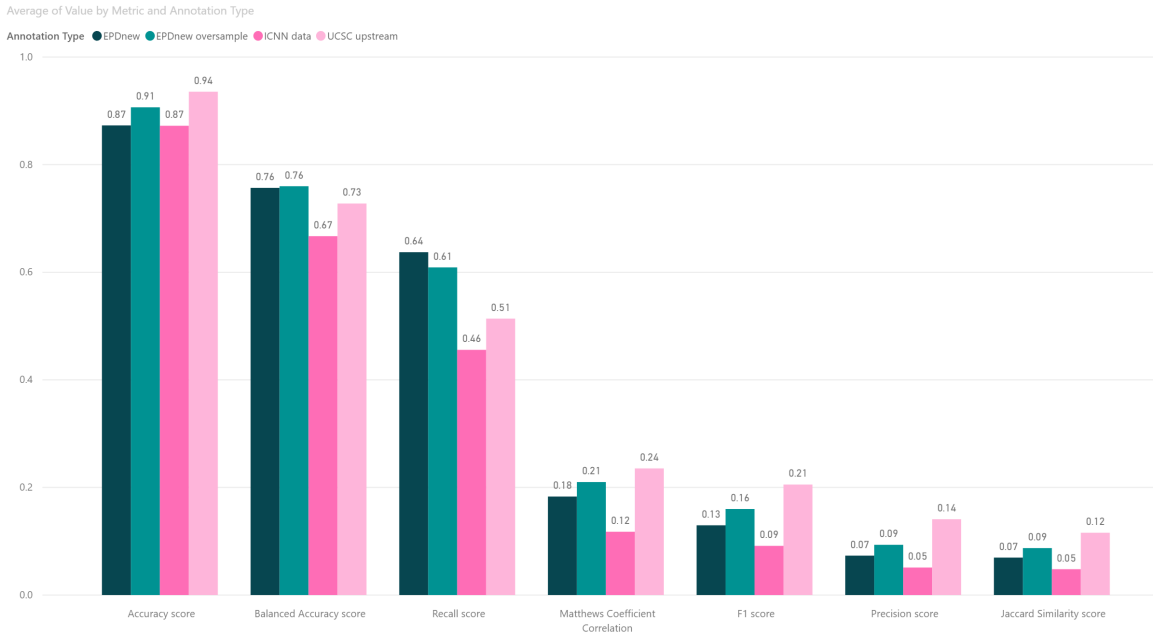


Figure 7.2: Results from different promoter annotation datasets being tested on human and mouse chromosome data

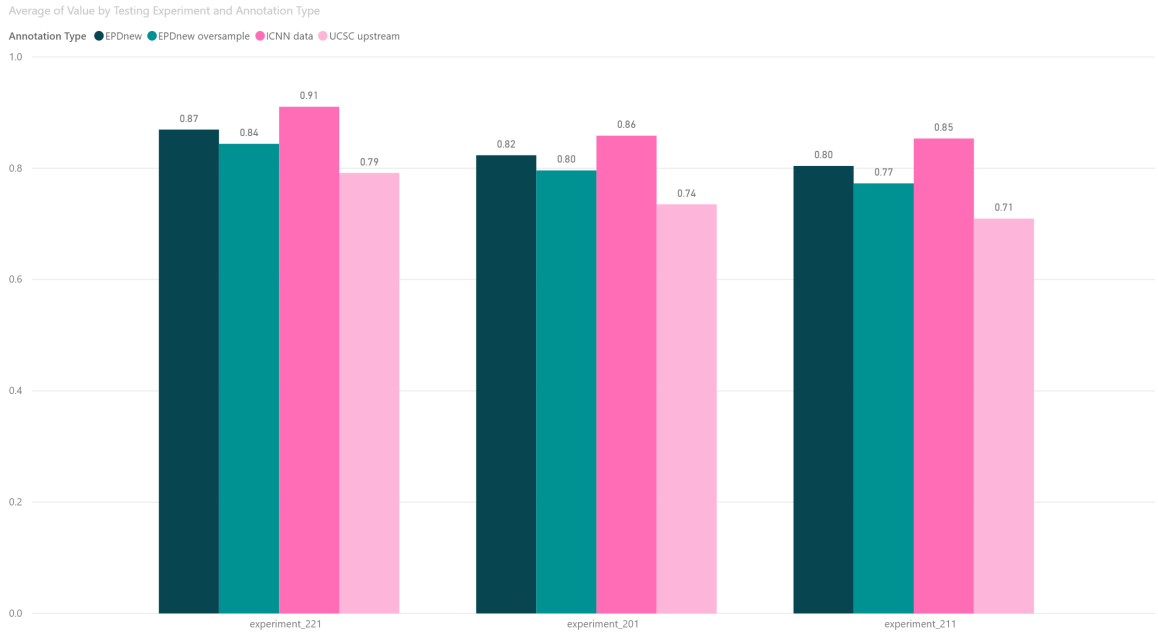


Figure 7.3: Results from different promoter annotation datasets being tested on different species

7.3.1.3 Comparison of DeePromoter's data

This section focuses on testing trained models with DeePromoter's data to gain insights into models using specific promoter types and synthetic data for training purposes. Promoter types include promoters containing a TATA-box motif and promoters without a TATA-box motif. Synthetic data includes the substitution method by Oubounyt et al. [93] using uniformly distributed nucleotides, and the same substitution method using the original sequence's nucleotide distribution. This latter distribution has the same properties of shuffling the nucleotides instead of replacing them with completely independent DNA sequences.

Figure 7.4, Figure 7.5, and Figure 7.6 show results comparing the different promoter types. The 'all promoters' set contains TrainEs 305, 306, 315, and 316. The 'non TATA-box promoters' set contains TrainEs 303, 304, 315, and 316. Finally, the 'TATA-box promoters' set contains TrainEs 301, 302, 311, and 312. All results display a trend where the models trained on TATA-box promoters only are insufficient in recognizing promoters, although Figure 7.5 show significantly better accuracy for models trained on TATA-box promoter when tested on entire chromosomes. This is because non-promoter sequences are more accurately classified with models trained on TATA-box promoters. The precision of all models is still very low, which might imply that the models focus on the non-promoter class.

Figure 7.7, Figure 7.8, and Figure 7.9 show results comparing the different synthetic data strategies. The 'original distribution' set contains TrainEs 301, 303, 305, 311, 313, and 315, while the 'uniform distribution' set contains TrainEs 302, 304, 306, 312, 314, and 316. Here, the 'original distribution' substitution trends to lower values as opposed to the 'uniform distribution' substitution methods. We believe this happens because the generated non-promoters from the 'original distribution' method are indistinguishable from promoters to the machine learning model.

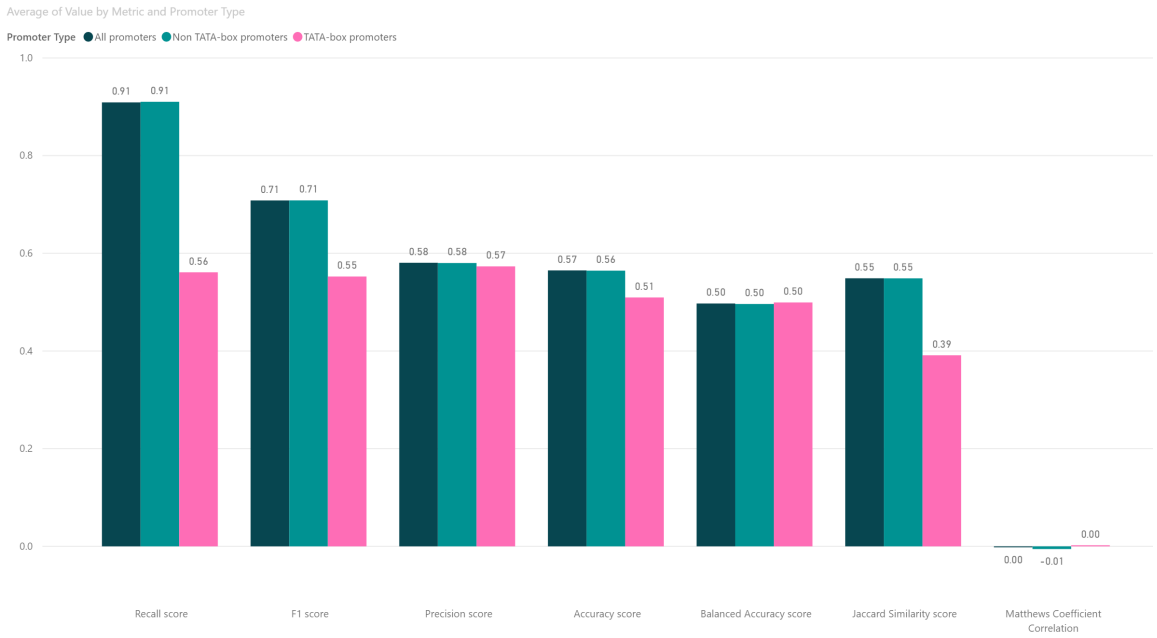


Figure 7.4: Promoter type experiments tested on baseline dataset

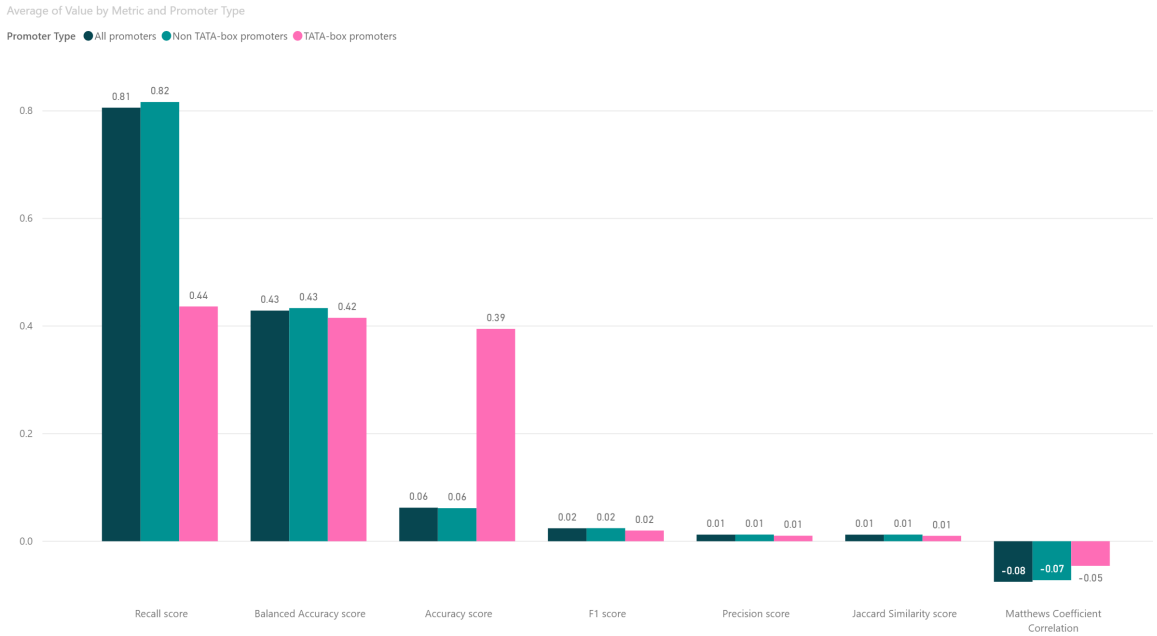


Figure 7.5: Promoter type experiments tested on chromosome datasets

Figure 7.6: Promoter type experiments tested on species datasets

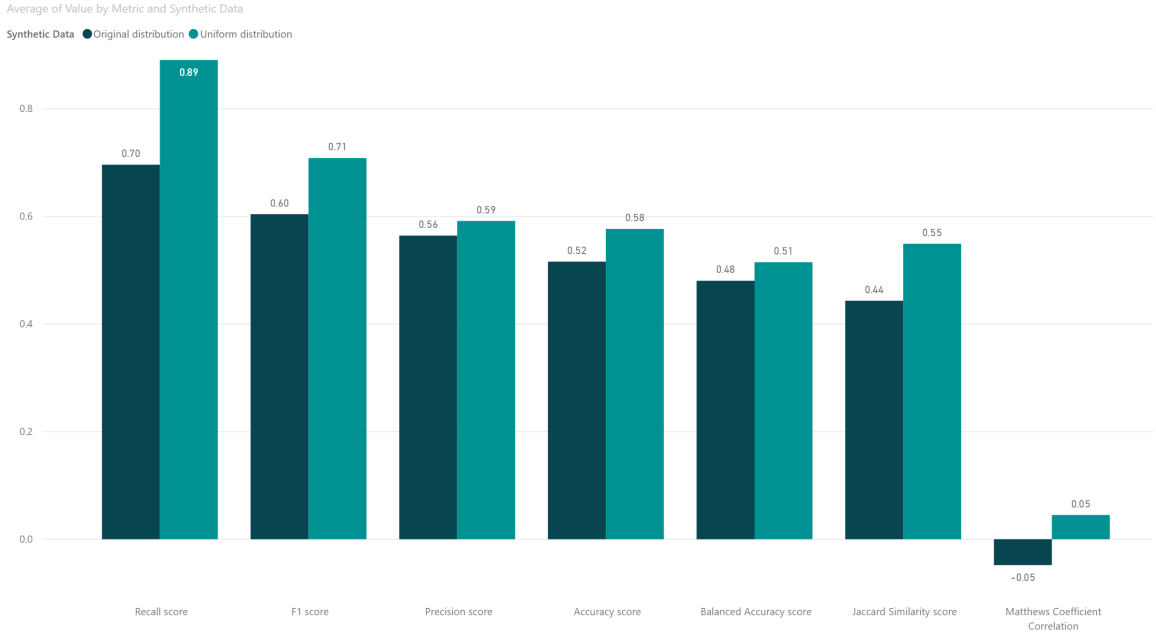


Figure 7.7: Results comparing the different synthetic data strategies on baseline ICNN data

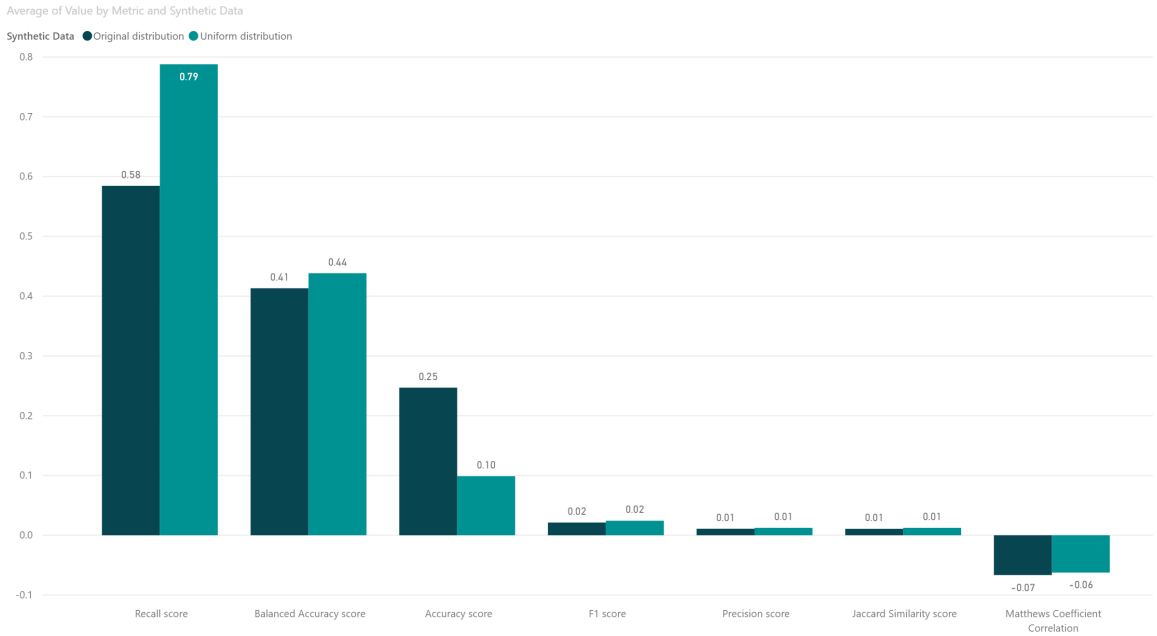


Figure 7.8: Results comparing the different synthetic data strategies on human and mouse chromosome data

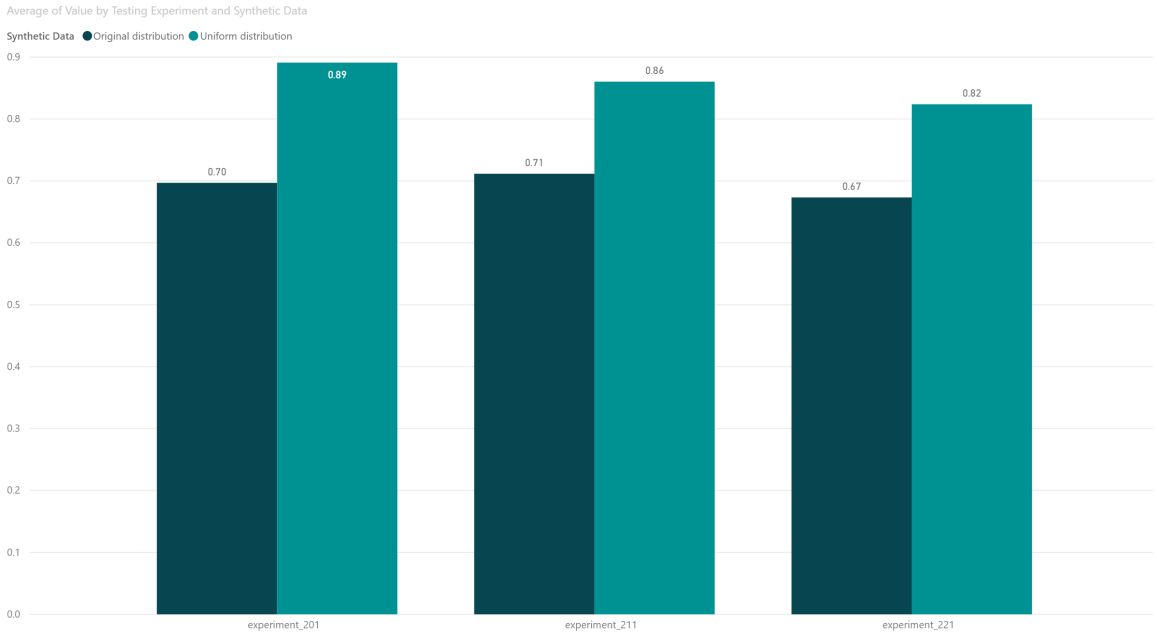


Figure 7.9: Results comparing the different synthetic data strategies on different species

7.3.1.4 Comparison of sampling methods

Sampling methods are used to balance the number of promoter and non-promoter sequences within a dataset. In our case, we created datasets containing a ratio of at least one promoter for every 10 non-promoters, contrasting from the naturally occurring ratio of one promoter sequence for every 100 non-promoter sequences. The ideal ratio is perfectly balanced; in practice, however, this is not possible for promoter recognition. In the training process, a balanced dataset is beneficial for machine learning models to avoid the model from focusing on the majority non-promoter class. We verify the performance claims using a balanced dataset by comparing models with oversampling, undersampling, and a 'normal' baseline. The results are shown in Figures 7.10, 7.11, 7.12

The oversampling set contains TrainEs 03, 04, 13, and 14. The undersampling set only contains the dataset from the literature (ICNN) in TrainE 201. The 'normal' set includes TrainEs 01, 02, 11, and 12. Results show that undersampling did best in the baseline dataset. However, this does not mean anything significant, as in this case, the dataset for training is the same as the testing dataset. It is interesting that on most metrics, the 'normal' set does marginally better than oversampling. The exception is for recall, where oversampling performs significantly better. As for the recall on the species testing dataset, undersampling seems to be the better option.

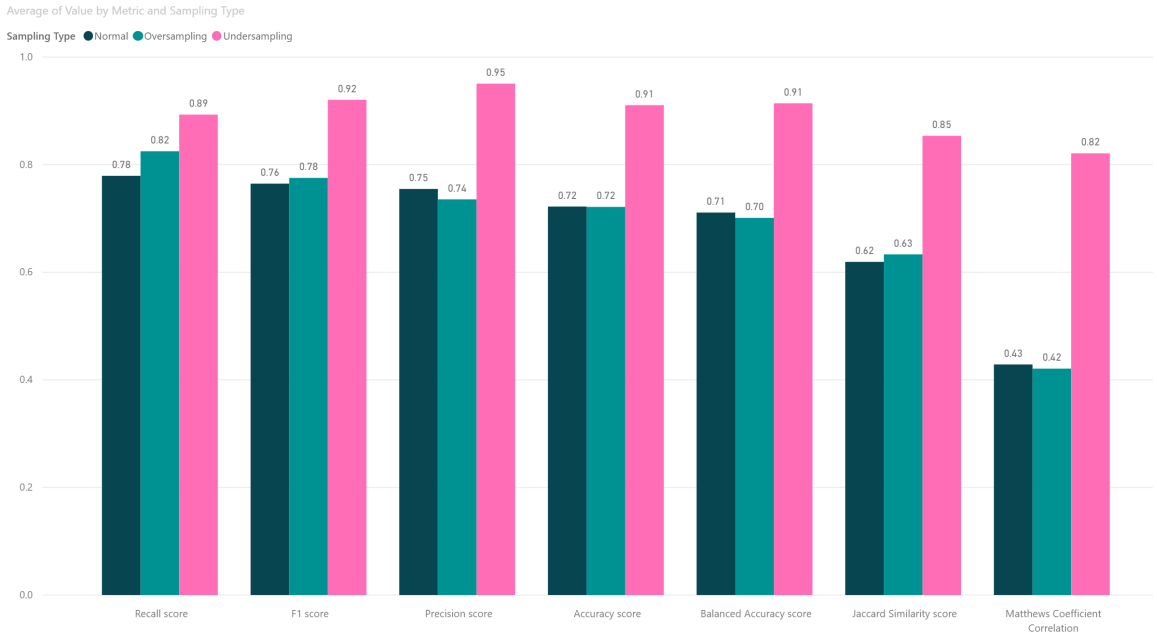


Figure 7.10: Results comparing sampling methods for imbalanced data on baseline ICNN data

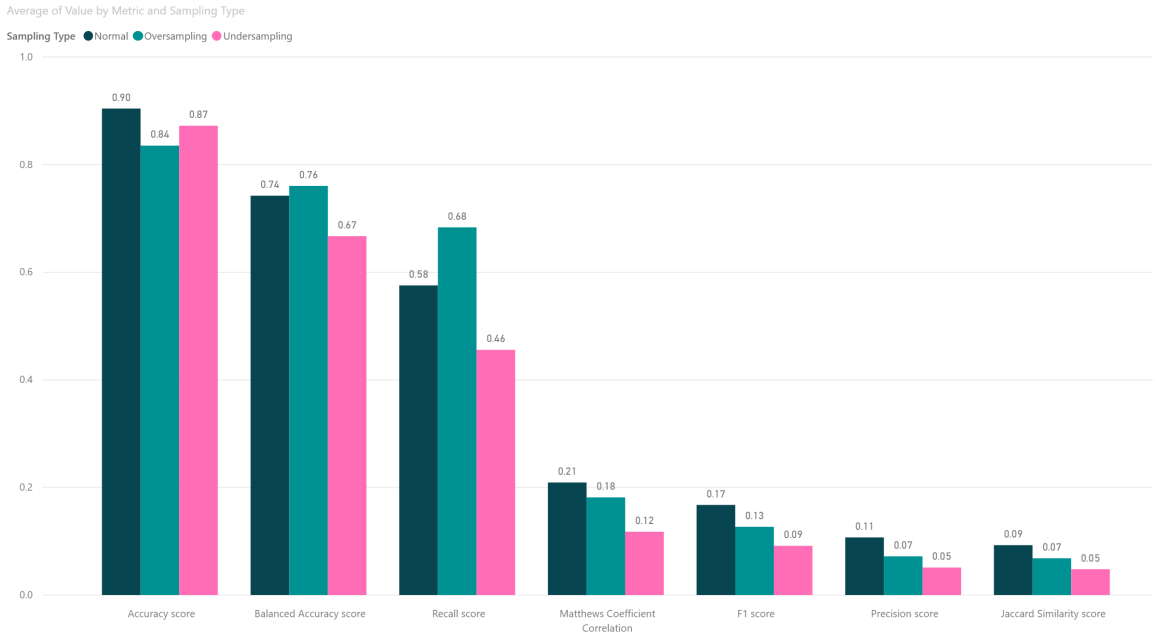


Figure 7.11: Results comparing sampling methods for imbalanced data on human and mouse chromosome data

Figure 7.12: Results comparing sampling methods for imbalanced data on different species

7.3.1.5 Comparison between species

Models have been trained using both human and mouse datasets to test how well they can perform on promoter recognition for related species. These results can help us understand how a trained model translates to an unknown, but related genome. It can also provide us with information on how the model might perform if trained on species that are more closely related to the tested species.

Models have been grouped into 7 categories depending on which specific species dataset they were trained on. As before, the 'ICNN data' category includes data from TrainEs 201. The 'human curated' category includes data from TrainEs $301 - 306$, while 'mouse curate' includes data from TrainEs $311-316$. The 'human chromosomes' category includes data from TrainEs $01 - 04$, whereas 'mouse chromosomes' includes data from TrainEs $11 - 14$. Lastly, the 'human both' category includes data from TrainE 401 and 'mouse both' from TrainE 411.

When tested on the baseline dataset (Figure 7.13), the human categories perform better, but it is notable that their mouse counterparts perform comparably. When tested on human chromosomes (Figure 7.14), the model trained on mouse chromosomes perform very similarly to the models train on human chromosomes. When tested on mouse chromosomes (Figure 7.15), the human models performed only slightly worse than mouse models. Note that on recall for both human and mouse tests, the 'mouse both' category came out on top. When comparing all categories using the promoter-only dataset for species (Figure 7.16), it is interesting that all categories did better with rat, with the exception of curated datasets, which are obtained from DeePromoter data. This suggests that the models can transfer to related species well, and that the more related the species, the better the results might be.

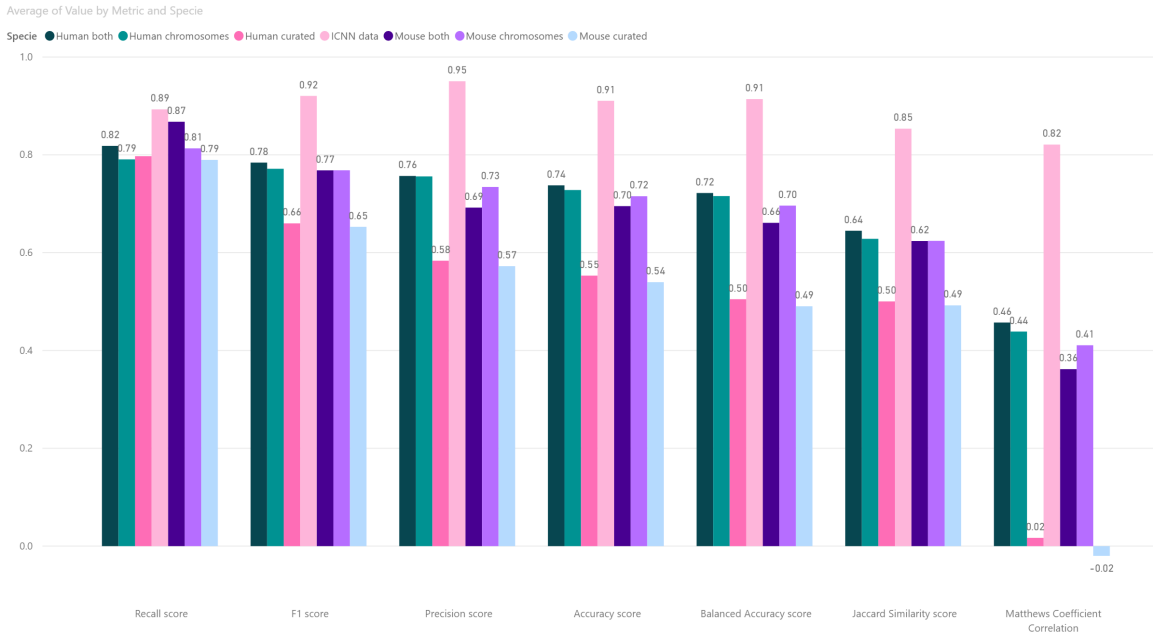


Figure 7.13: Results comparing models trained on human and mouse data by testing on baseline ICNN data

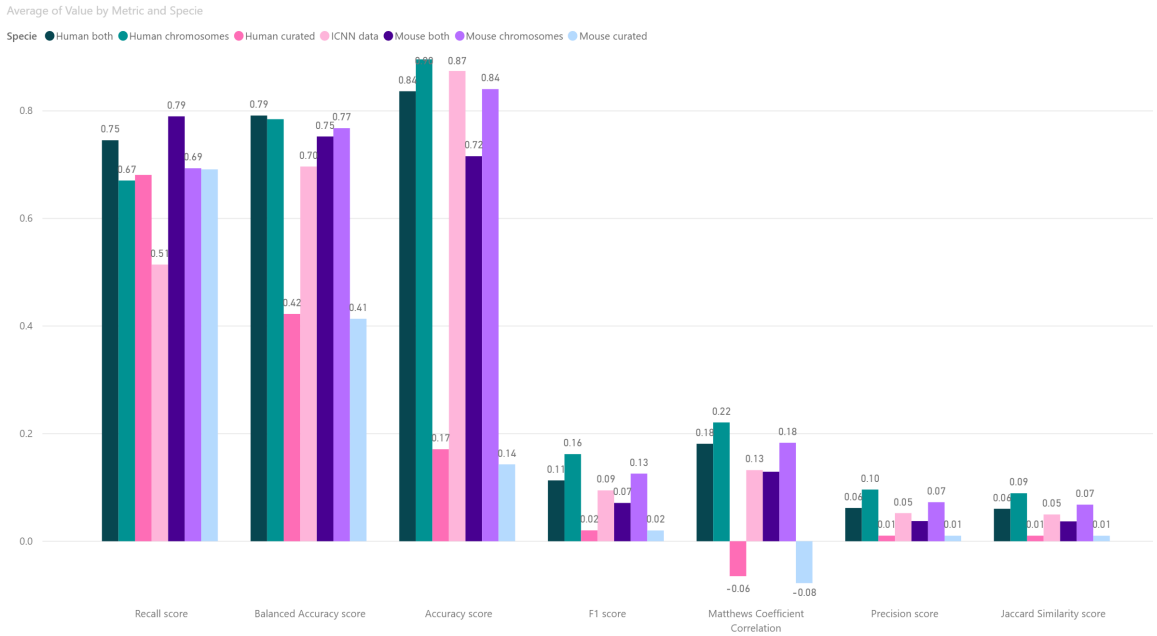


Figure 7.14: Results comparing models trained on human and mouse data by testing on human chromosome data

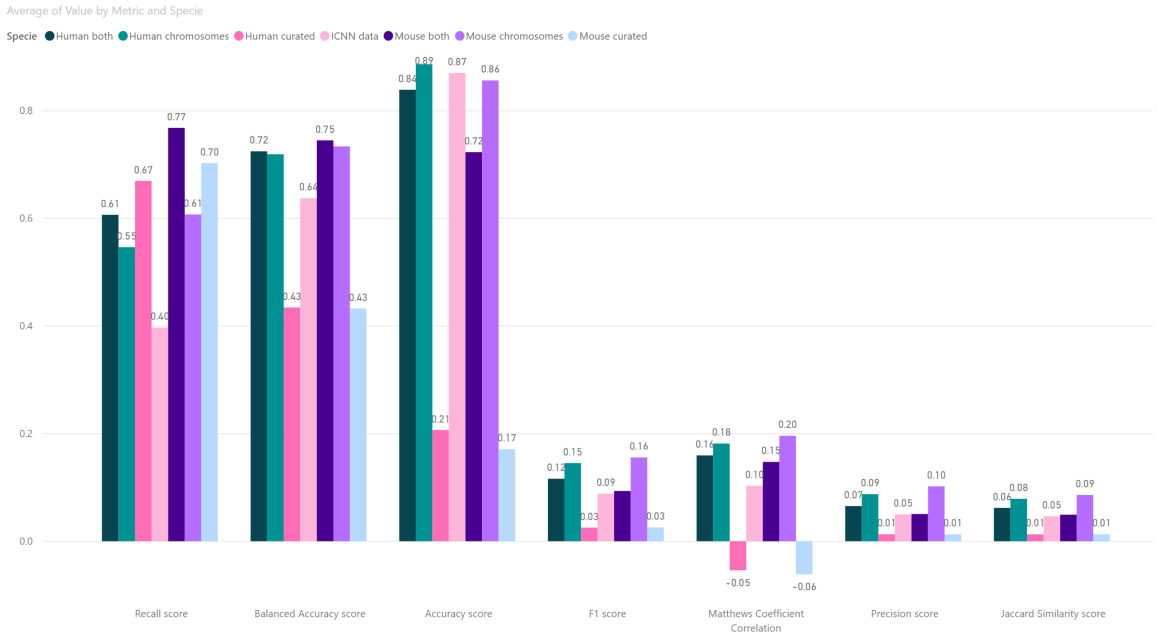


Figure 7.15: Results comparing models trained on human and mouse data by testing on mouse chromosome data

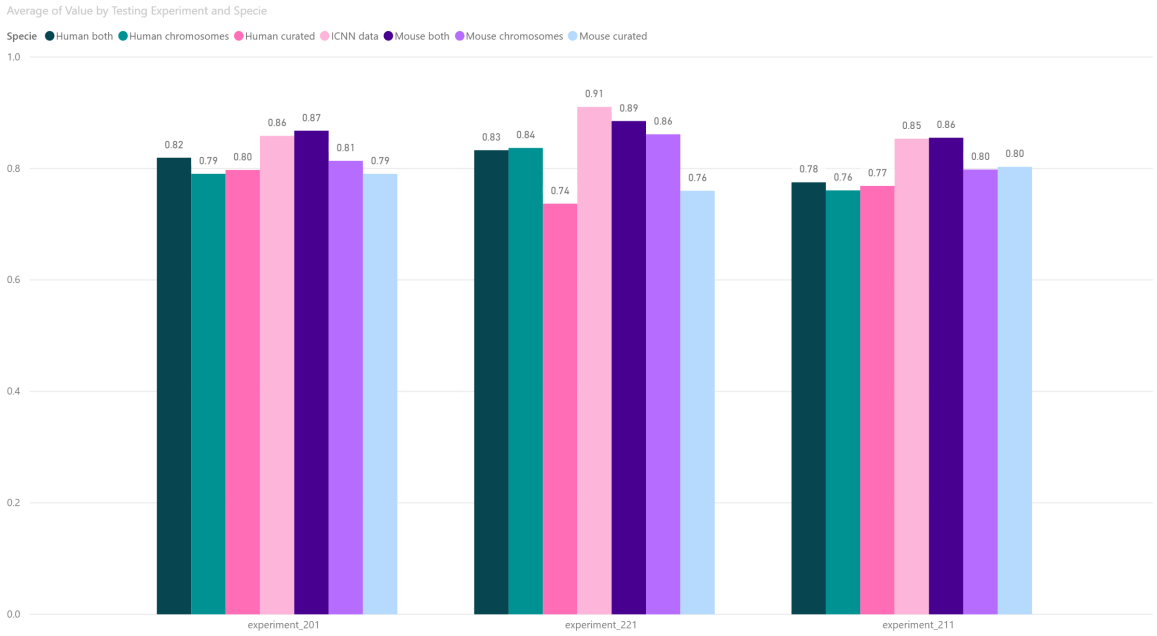


Figure 7.16: Results comparing models trained on human and mouse data by testing on different species

7.3.1.6 Comparison between output functions

We explore how the activation function in the output layer of a neural network can affect the performance of the model. The activation function makes a difference in the classification of the model, as no threshold is needed when using softmax as opposed to sigmoid. This is because softmax requires two output neurons, where one gives a probability of the input being a promoter, and the other gives the probability of being a non-promoter.

Figure 7.17 contains four figures. The two figures on the left compare experiments tested on mouse chromosomes, while the two figures on the right compare experiments tested on human chromosomes. The top row displays values regarding precision, while the bottom row contains values pertaining recall. For most models trained on chromosomes, their precision is better when having a sigmoid activation function. On the other hand, the recall is better for models trained on chromosomes when having a softmax activation function.

Figure 7.18 shows how the metrics compare between sigmoid and softmax activation function when trained on the baseline ICNN data (201). The values are similar for the activation functions. Notably, Figure 7.19 depicts a completely reversed trend. The models used to create Figure 7.19 were trained on chromosomes sequences, and were selected for their high precision in comparison to the rest of the models. These high-precision models can be seen at the top row of Figure 7.17 as experiments 01 and 11. These previous figures, which show models trained on chromosomes (01 and 11) and the baseline data (201) that were tested on the same chromosome data, show how the training data can also affect the activation function's output. Finally, Figure 7.20 displays the results from the models with the highest recall (14 and 411) on Figure 7.17, just as Figure 7.19 did for high precision models.

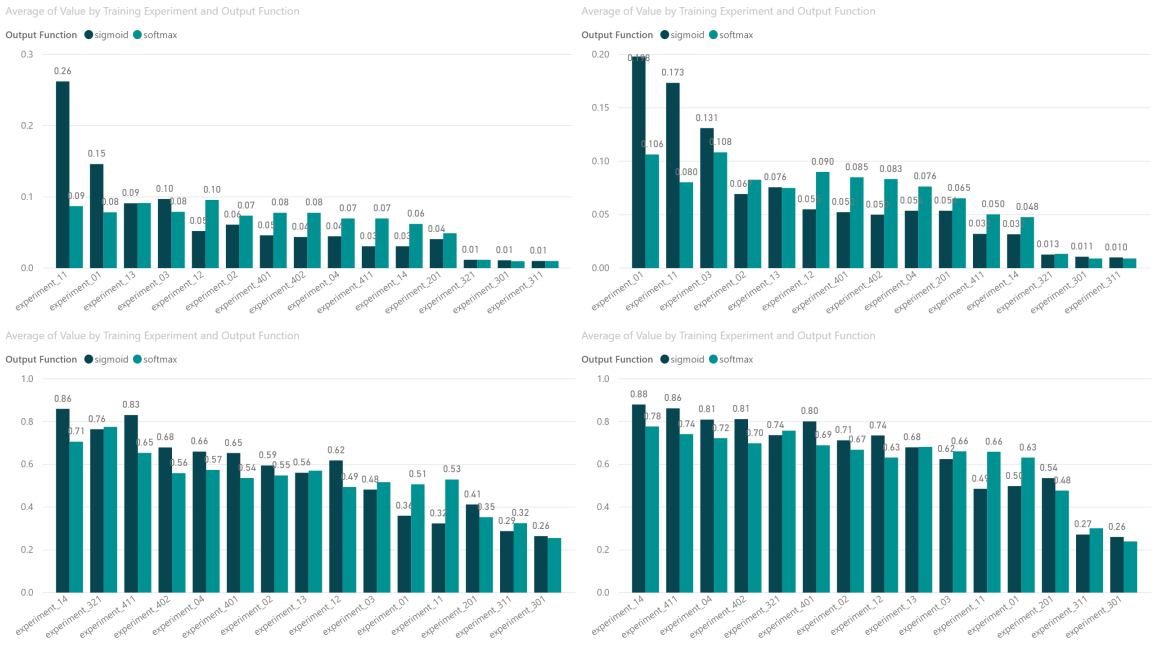


Figure 7.17: Recall and precision comparison between output functions from most experiments' models on human and mouse chromosome data

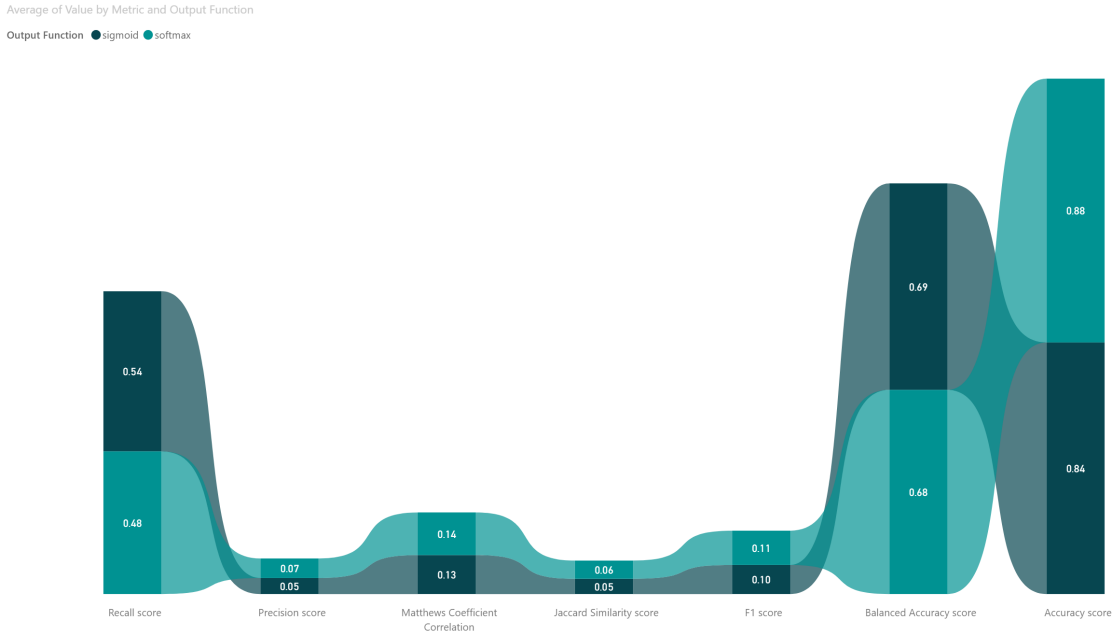


Figure 7.18: Metrics comparison between output functions using baseline ICNN data

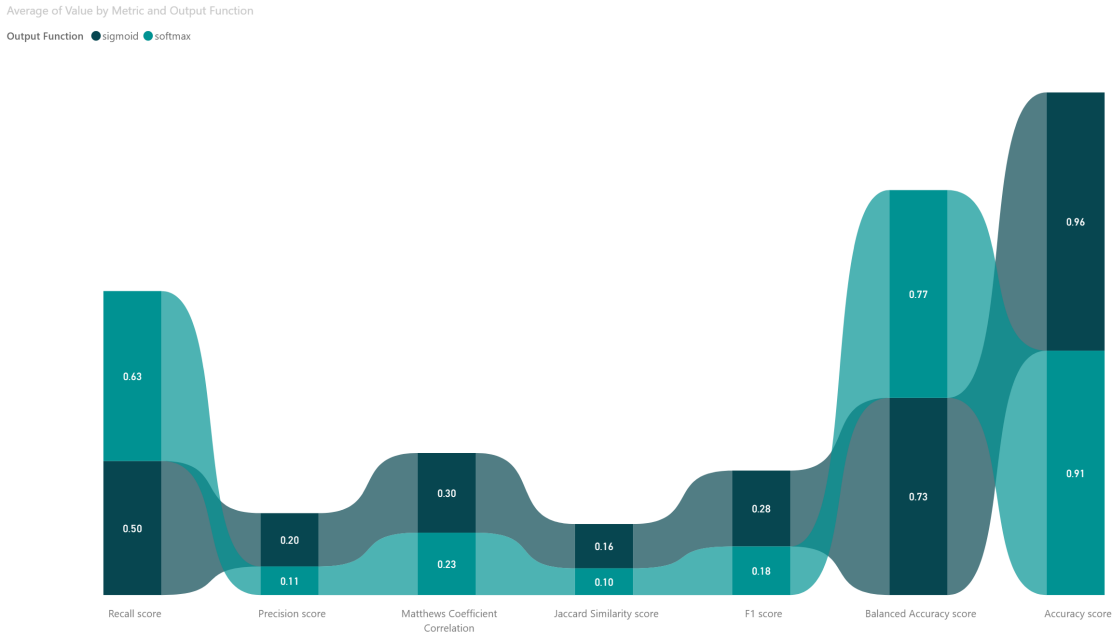


Figure 7.19: Metrics comparison between output functions using high-precision models trained on chromosome data

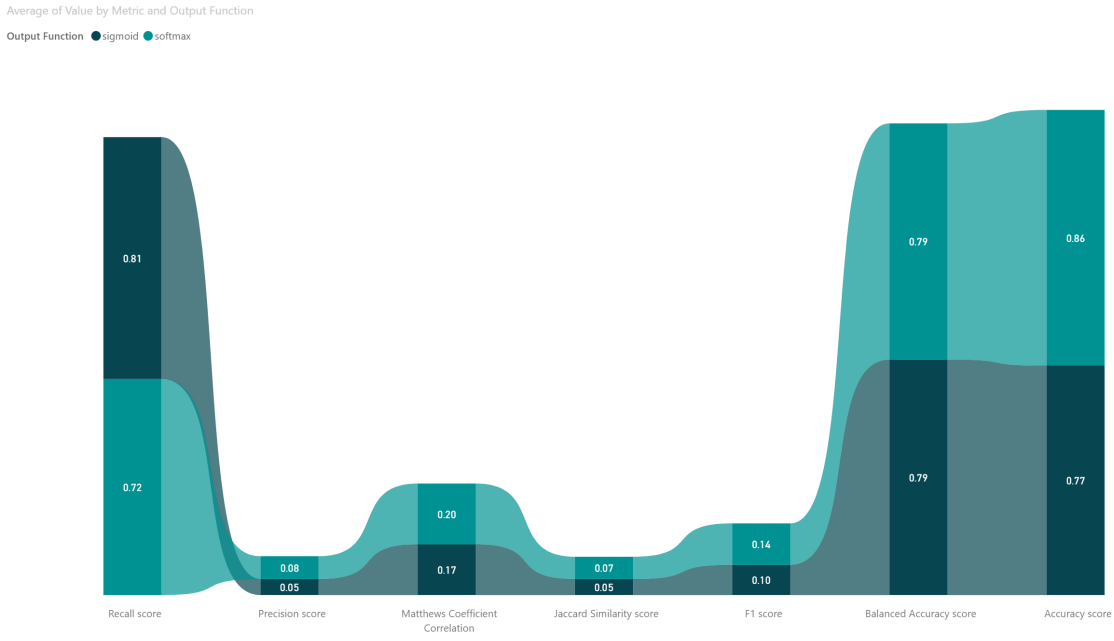


Figure 7.20: Metrics comparison between output functions using high-recall models trained on chromosome data

7.3.1.7 Comparison of tolerance threshold levels

The results for different tolerance thresholds do not vary much. Figures in this section compare MCC results from experiments, since it contains some variability. Figure 7.21 shows experiments being tested on the baseline ICNN data. Figure 7.22 shows experiments being tested on chromosome data. The variability is most noticeable with tolerance 50 in Figure 7.22, as the selection process becomes more lenient when choosing promoter sequences. This leniency can affect the models' performances when testing on the baseline dataset, as shown in Figure 7.21. Interestingly, the trend on models (excluding $3xx$) performing progressively worse when tested on chromosomes, whereas the models tested on the baseline dataset have a mostly stable performance around 0.45.

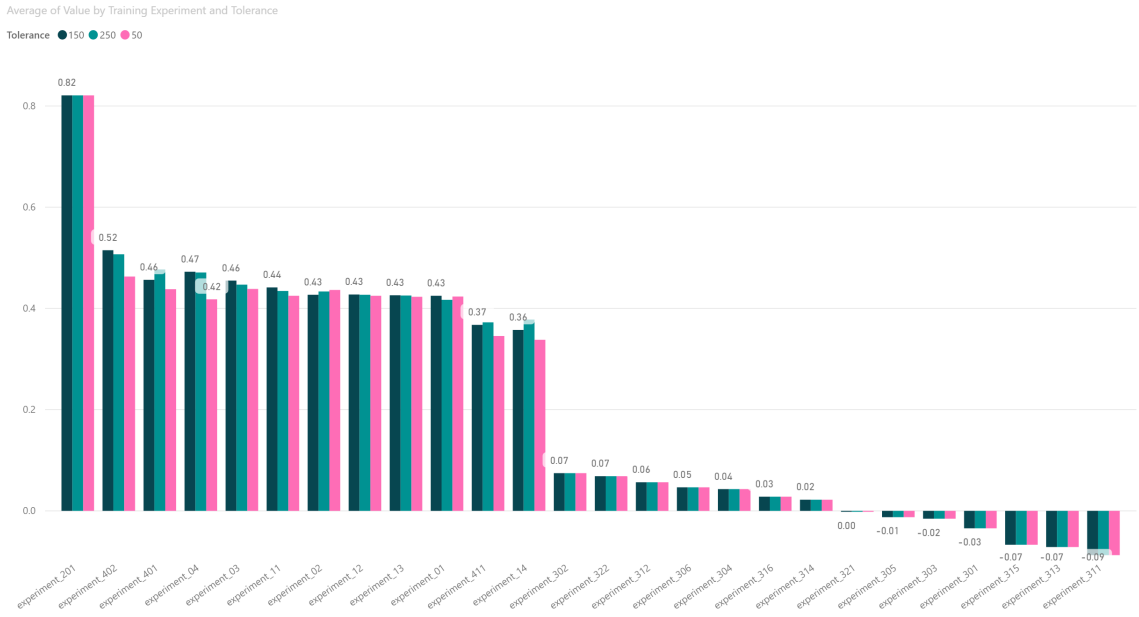


Figure 7.21: Comparison of tolerance threshold levels on baseline ICNN data

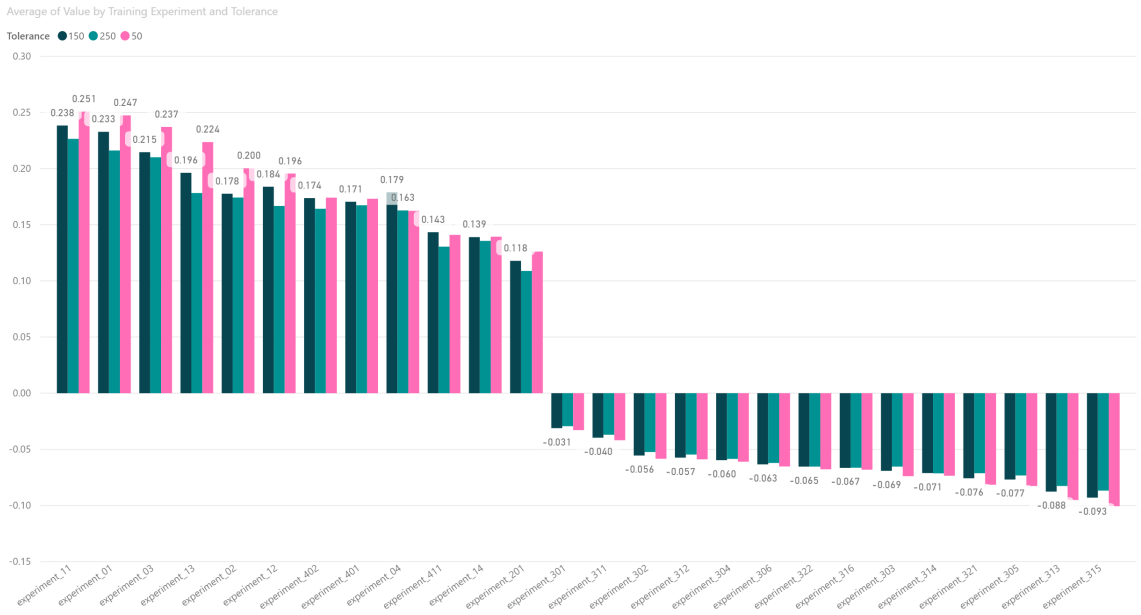


Figure 7.22: Comparison of tolerance threshold levels on human and mouse chromosome data

Chapter 8

Discussion and Future Work

This chapter discusses the findings from our main chapters and offers insights into how the process of ab initio promoter recognition can be improved. We also describe ways to expand this work.

8.1 DNA as a natural language

In chapter 5, we look into modelling DNA as a language by utilizing NLP architectures for creating trained models that recognize promoters. We learned that most of our tested models perform slightly differently when evaluating their results. Considering all metrics measured between models with the same embeddings, they had a maximum difference of about 5%. This difference was larger when models with different embeddings and k-mers were compared. The largest difference between metrics for all models amounted to approximately $12\% - 15\%$, which is substantial.

We examined a limited amount of k-mer lengths to determine which of these would work best. We found that larger k values tend to decrease the accuracy of the model, although larger k-mers should be tested to produce a definitive conclusion. Embeddings are created to find similarities between the different kmers or terms in a language. The embeddings explored did not provide an improvement for all models. The models that benefitted from embeddings have architectures that do not contain an NN layer for processing patterns before the sequence learning RNN layer. A CNN or a hierarchical network structure can process the input for patterns and thus eliminate the need for embeddings in our case. The similarities between DNA k-mers found using embeddings might not have a purpose in DNA sequences, as its language

structure is poorly understood. In our findings, we noticed that the embeddings served as a limited type of pattern recognition layer between k-mers. It also appears that a simple architecture combining a CNN and an RNN will model the data with similar performance to complex architectures with hierarchical structures. An advantage of hierarchical architectures seems to be their ability to learn the language structure within a sequence requiring a k-mer value. In comparison, convolutions in CNNs require a filter size, which acts as the k-mer length, making CNNs less flexible than hierarchical architectures.

We suggest that k-mers with length $k \in prime$ should be tested, since their length cannot be formed as a combination of previously explored k-mers. Convolution strides should also be fully explored, as the tests we performed contained a constant stride value. Testing larger k-mers creates computationally restrictive amounts of terms, and thus an approximation could be made using convolutional filter sizes because of the similar behaviour between k-mers and convolutions in section 5.4. Other ideas that should be further explored include the creation of more efficient methods than k-mer encoding to avoid the k-mer combinatorial explosion problem. One such idea might be to create very similar k-mers, sequences with few nucleotide polymorphisms, to use the same embedding. Additionally, nucleotide base pair encodings could also be simplified by using their chemical classifications including: strong vs. weak, purine vs. pyrimidine, and keto versus amino. Alternative representations should also be explored to provide a comprehensive view of all possibilities. These representations are included in the standard IUB/IUPAC nucleic acid codes[1].

Regarding the training performance, most models had started to decrease in validation performance by 30 epochs. This means that the models were starting to overfit to the training data. Therefore, the training of most models considered could be shortened from 100 epochs to around 20 to 30 epochs. We managed to resolve this in later experiments by utilizing early stopping regularization. The only models that did not overfit over 100 epochs were models possessing an LSTM architecture. Although the LSTM models' validation loss was still decreasing by 100 epochs, their performance was not on par with other tested models. With more training, the LSTM models might have reached the same performance as the other models. However, this would be improbable since even the LSTM models with embeddings could not reach that performance level before starting to overfit. It is notable that some LSTM models reached the validation accuracy levels of other models with different

[1] `https://emunix.emich.edu/~evett/BioinformaticsTools/IUB%20Codes.htm`

architectures, but in the testing stage, these LSTM models could not perform as well. Another interesting discovery in the training data is that all architectures seem sufficiently able to model the data, as the training loss could read values very close to zero. This does not mean the models would be acceptable at promoter recognition, but rather at memorizing the sequences. This indicates that the model can still be improved if more data is available and further validated by NLP models for human languages that perform well in tasks containing a vast amount of training data. An impediment to consider is that promoter recognition is part of a partially understood language (DNA). However, as the acquisition of more differing DNA sequences from each species is obtained, the amount of data used will approach the amount other NLP tasks require.

In terms of interpreting the models' results, attention is an appropriate candidate to visualize and explore what the model deems as important when recognizing a sequence. It is important to note that our results provided low levels of attention throughout the sequence. This might have happened because of the limited number of training data utilized, or because the attention mechanism used was not intricate enough. Since our implemented attention mechanism was developed in 2016, newer attention mechanisms might offer better results. Additional scrutiny to the models should also be provided by using attention as a visualization technique, in order to validate the models and interpret them. This is because models might be recognizing non-promoters instead of recognizing promoters as desired. There are methods, such as out of distribution generalization [103], that try to alleviate this problem. These should be explored to potentially increase the validity of the model.

There are many other methods for interpreting results from deep learning models, including sample-based methods like LIME and gradient-based methods like the work by Ross et al. [109]. These and more methods can be further studied in machine learning interpretability surveys [87, 15, 3, 166, 17]. All of these interpretability methods should be attempted in promoter recognition models to gain more insights into how the models recognize promoters. We anticipate that this will contribute to the creation of better models, as well as increase the understanding of promoter sequences.

8.2 Comparing approaches from the literature

In chapter 6, the results obtained from comparing the models from the literature were unexpected, as the models performed very well in their original evaluations presented in their respective papers. We believe that this is important for people who want to use a machine learning model to solve a task. It is imperative that the models be reevaluated with one's own data to ensure that a model will be able to perform its task accordingly. In case this is not possible, it is important to note the amount of data used to train a machine learning model. This is more evident in deep learning models, as these tend to overfit the data. If the data is limited and does not appear to be a representative sample of the task, there are two possibilities. One possibility is to obtain the amount of data necessary to make the training dataset a representative sample, although it might be difficult to judge. The other possibility is to explore other methods for approximating a solution to the problem.

From Qian et al.'s comparison results in Figure 3.6, we can assume that their model was better than CNNProm, although it can be argued that ICNN requires positional knowledge that was given by the encoding schemes of the model. The encoding worked in that case because the position was known beforehand, as all the data was located in the correct position relative to the TSS. In a promoter recognition problem, we would not have that information, explaining why CNNProm produced better results than ICNN in section 6.2. DeePromoter is inferred to perform the worst out of the three because of their use of synthetic non-promoter data, which ends up being unrepresentative of natural DNA sequences.

We also inferred from surveys from the literature (chapter 3 that most early recognition methods have the purpose of analysing motifs that are part of promoters. For promoter recognition, there are many methods being developed in slightly varying ways to solve this task. The definition of the task has not been formally defined, and as such, it has been labeled using different terminology. This terminology includes promoter recognition, classification, detection, and prediction. This makes it difficult to know whether a method has already been discovered or attempted by others in the past. A formal name for each task, and a naming convention for methods solving the task, should be developed to improve the progress of the research for this area.

8.3 Testing methodology

The shortcomings obtained from our comparison of models from the literature highlighted the need to create a testing methodology that provides robust and extensive promoter recognition testing datasets. In chapter 7, we provide the details of this testing methodology; in this section, we explain how our methodology can transfer to different types of genomic tasks. Firstly, the testing dataset resulting from our methodology can be created using other types of genomic sequences. This requires some source-code editing, but the same methodology should still apply. Such a method always requires a reference genome, while the promoter file changes to the specific type of sequence one is interested in. This can be further expanded to accept multiple types of files along with the reference genome. This means that our methodology allows us to derive a testing dataset for any type of DNA sequence. The algorithm in the script for the sequence alignment method does not perform very efficiently, but it allows the flexibility to perform additional operations to each aligned sequence by NBLAST. If efficiency is a priority, a transition method is also provided to create an annotation file from the aligned sequences result produced by NBLAST. The annotation database method is the most efficient method among all the provided methods, but requires an annotation file to start the process. The methods can transition only in one direction: from sequence alignment method and using the transition method to the annotations database method. The other direction, from the annotations database method to the sequence alignment method, is not provided.

The sequence alignment method can benefit from other alignment algorithms. When the BLAST database is made from promoter sequences, a global alignment algorithm could be used on the data, as the alignments occur between two sequences of the same size. In contrast, when the reference genome is used to create a BLAST database, a local alignment algorithm must be employed. This is because the alignment happens between a small sequence (the promoter) and a much larger sequence (the reference genome). Additional alignment algorithms that can be used include EMBOSS tools called 'needle' and 'water' for global and local alignment respectively.

Another area of improvement for the testing methodology includes adding more methods for addressing the imbalanced data problem in promoter recognition. We only explored basic methods that involve sampling, but other methods created to alleviate the imbalanced data problem that should be explored are described by Johnson and Khoshgoft [65]. These methods should improve a training algorithm's perfor-

mance by guiding the model to learn the desired features in DNA sequences that focus on promoter recognition. A PRM by Umarov et al. [134] that focused on alleviating the imbalanced data problem was recently published. This paper further validates the need for such methods to be utilized in solving this task.

Although biological assay data is not directly utilized in ab initio PRMs, they can be used to validate the results from these methods. The data can also be combined to create hybrid PRMs in order to create more comprehensive and precise recognition methods. All the databases mentioned in section 2.1.2 and data from section 2.1.3 can be utilized towards this end. This can also be achieved by creating multiple models for each type of data and making an ensemble of the models that can potentially recognize promoters with high precision and accuracy. Thus, if some data is unavailable, only the models that utilize the provided data will be used towards the promoter recognition estimate. Multiple studies [155, 169, 160] show the efficacy of several biological assay data for promoter recognition.

8.4 Additional future work

The advantages of ab initio deep learning promoter recognition models reside in finding short motif sequences specific to the promoters. As these motifs appear regularly throughout the genome, it is apparent that promoter recognition using only DNA sequences can prove ineffective. The ineffectiveness of the models has been validated using our testing methodology in chapter 7. One suggestion to improve ab initio promoter recognition involves using the DNA structure from a sequence. This can be estimated using only the DNA sequence by analyzing its free energy [158]. Another suggestion involves combining the model with chromatin information [64] to provide the DNA structure, although it involves using additional data, making it a hybrid PRM.

The models we explored are made to distinguish promoters from non-promoters using relatively short ($< 500 bps$) DNA sequences. Currently, models need to split a large DNA sequence into small subsequences to be able to process the data. Future work includes recognizing promoter sequences using larger segments of DNA. A minimum of 500 bp length sequences should be explored, as that is a representative size for a proximal promoter [158]. Another method that is noteworthy to explore involves recognizing a promoter sequence within a larger DNA sequence without the need to split the larger genomic sequence into subsequences. If research continues

following the methods of recent literature, which use relatively short DNA sequences to recognize promoters, a second model that can estimate TSS in a genome can be utilized. This second model can then be used as a first step to help the PRM focus on important areas of the genome that might be promoters, and limit the PRMs false positive results.

Another interesting technique worth exploring to combat the imbalance problem in promoters includes the creation synthetic data for positive sequences by examining the nucleotides of promoter sequences. Deep learning algorithms that are noteworthy for modelling and generating new promoter sequences are generative adversarial networks [142] and autoencoders [133]. Techniques involving the creation of synthetic data without deep learning could follow a similar pattern to the process from DeePromoter, combined with the motif knowledge of promoters from ICNN. An example for such a technique can be made by shuffling the 'Non-element' sequences from promoters with a tool, such as uShuffle [63], while allowing the 'Element' sequences to remain. This new sequence can then be regarded as a promoter, as the important 'Element' sequences that make up a promoter are still present.

We conclude by suggesting an ensemble between all three levels of PRMs [157], which might be able to provide a better overall model, and eliminate the false positives that are exceedingly present in ab initio PRM output. Ab initio PRMs can work as the most general level, where the promoters recognized by these methods are the best promoter candidates for the next level PRM. At this point, we can apply the hybrid approach described in section 2.1.4.1. The hybrid approach works on specific cell types, as it uses the specific information for that cell type in their recognition process. This can further discard promoters that are not used in the cell type. Homology-based PRMs can be used in both previous levels as a way of recognizing promoters in closely related organisms that have no genomic characterization. The homology PRMs can then focus on the TFBS of predicted promoters of the previous levels instead of all orthologous gene information. Homology PRMs rely on regulatory regions being conserved in closely related species. This might still be useful as a recent study [19] revealed that gene expression patterns and transcription factor binding preferences are broadly conserved across animals. They also demonstrated these findings for short sequence motifs in distal promoter sequence architectures between mammals.

Bibliography

[1] Bruce Alberts. *Molecular Biology of the Cell*. Garland Science, August 2017. Google-Books-ID: jK6UBQAAQBAJ.

[2] Robin Andersson and Albin Sandelin. Determinants of enhancer and promoter activities of regulatory elements. *Nature Reviews Genetics*, October 2019.

[3] Vijay Arya, Rachel K. E. Bellamy, Pin-Yu Chen, Amit Dhurandhar, Michael Hind, Samuel C. Hoffman, Stephanie Houde, Q. Vera Liao, Ronny Luss, Aleksandra Mojsilović, Sami Mourad, Pablo Pedemonte, Ramya Raghavendra, John Richards, Prasanna Sattigeri, Karthikeyan Shanmugam, Moninder Singh, Kush R. Varshney, Dennis Wei, and Yunfeng Zhang. One Explanation Does Not Fit All: A Toolkit and Taxonomy of AI Explainability Techniques. *arXiv:1909.03012 [cs, stat]*, September 2019. arXiv: 1909.03012.

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv:1409.0473 [cs, stat]*, May 2016. arXiv: 1409.0473.

[5] Vladimir B. Bajic, Michael R. Brent, Randall H. Brown, Adam Frankish, Jennifer Harrow, Uwe Ohler, Victor V. Solovyev, and Sin Lam Tan. Performance assessment of promoter predictions on ENCODE regions in the EGASP experiment. *Genome Biology*, 7(1):S3, August 2006.

[6] Vladimir B. Bajic, Sin Lam Tan, Yutaka Suzuki, and Sumio Sugano. Promoter prediction analysis on the whole human genome. *Nature Biotechnology*, 22(11):1467–1473, November 2004.

[7] Keith R. Benson. T. H. Morgan's resistance to the chromosome theory. *Nature Reviews Genetics*, 2(6):469–474, June 2001.

[8] Michael F. Berger and Martha L. Bulyk. Universal protein binding microarrays for the comprehensive characterization of the DNA binding specificities of transcription factors. *Nature protocols*, 4(3):393–411, 2009.

[9] Elizabeth M. Blackwood and James T. Kadonaga. Going the Distance: A Current View of Enhancer Action. *Science*, 281(5373):60–63, July 1998.

[10] Matthias Blohm, Glorianna Jagfeld, Ekta Sood, Xiang Yu, and Ngoc Thang Vu. Comparing Attention-Based Convolutional and Recurrent Neural Networks: Success and Limitations in Machine Reading Comprehension. In *Proceedings of the 22nd Conference on Computational Natural Language Learning*, pages 108–118, Brussels, Belgium, 2018. Association for Computational Linguistics.

[11] Thomas Boulin. Reporter gene fusions. *WormBook*, 2006.

[12] R. R. Burgess. Sigma Factors. In Sydney Brenner and Jefferey H. Miller, editors, *Encyclopedia of Genetics*, pages 1831–1834. Academic Press, New York, January 2001.

[13] Prabir Burman. Estimation of Optimal Transformations Using v-Fold Cross Validation and Repeated Learning-Testing Methods. *Sankhyā: The Indian Journal of Statistics, Series A (1961-2002)*, 52(3):314–345, 1990.

[14] Steve Busby and Richard H. Ebright. Promoter structure, promoter recognition, and transcription activation in prokaryotes. *Cell*, 79(5):743–746, December 1994.

[15] Diogo V. Carvalho, Eduardo M. Pereira, and Jaime S. Cardoso. Machine Learning Interpretability: A Survey on Methods and Metrics. *Electronics*, 8(8):832, August 2019.

[16] C. Thomas Caskey and Philip Leder. The RNA code: Nature's Rosetta Stone. *Proceedings of the National Academy of Sciences*, 111(16):5758–5759, April 2014.

[17] Supriyo Chakraborty, Richard Tomsett, Ramya Raghavendra, Daniel Harborne, Moustafa Alzantot, Federico Cerutti, Mani Srivastava, Alun Preece, Simon Julier, Raghuveer M. Rao, Troy D. Kelley, Dave Braines, Murat Sensoy, Christopher J. Willis, and Prudhvi Gurram. Interpretability of deep learning models:

A survey of results. In *2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, pages 1–6, San Francisco, CA, August 2017. IEEE.

[18] Sneha Chaudhari, Gungor Polatkan, Rohan Ramanath, and Varun Mithal. An Attentive Survey of Attention Models. *arXiv:1904.02874 [cs, stat]*, April 2019. arXiv: 1904.02874.

[19] Ling Chen, Alexandra E. Fish, and John A. Capra. Prediction of gene regulatory enhancers across species reveals evolutionarily conserved sequence properties. *PLoS Computational Biology*, 14(10), October 2018.

[20] Yifei Chen, Yi Li, Rajiv Narayan, Aravind Subramanian, and Xiaohui Xie. Gene expression inference with deep learning. *Bioinformatics*, 32(12):1832–1839, June 2016.

[21] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. June 2014.

[22] Onuma Chumsakul, Kensuke Nakamura, Shu Ishikawa, and Taku Oshima. GeF-seq: A Simple Procedure for Base Pair Resolution ChIP-seq. *Methods in Molecular Biology (Clifton, N.J.)*, 1837:33–47, 2018.

[23] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. December 2014.

[24] Jeanne Chèneby, Marius Gheorghe, Marie Artufel, Anthony Mathelier, and Benoit Ballester. ReMap 2018: an updated atlas of regulatory regions from an integrative analysis of DNA-binding ChIP-seq experiments. *Nucleic Acids Research*, 46(D1):D267–D275, 2018.

[25] Geoffrey M. Cooper. The Sequences of Complete Genomes. *The Cell: A Molecular Approach. 2nd edition*, 2000.

[26] NCBI Resource Coordinators. Database Resources of the National Center for Biotechnology Information. *Nucleic Acids Research*, 45(Database issue):D12–D17, January 2017.

[27] Jean-Baptiste Cordonnier, Andreas Loukas, and Martin Jaggi. On the Relationship between Self-Attention and Convolutional Layers. *arXiv:1911.03584 [cs, stat]*, January 2020. arXiv: 1911.03584.

[28] Arnold Cosmas D. A functional assay for promoters. *Nature Methods*, 14(2):109–109, February 2017. Number: 2 Publisher: Nature Publishing Group.

[29] Nevena Cvetesic and Boris Lenhard. Core promoters across the genome. *Nature Biotechnology*, 35(2):123–124, February 2017. Number: 2 Publisher: Nature Publishing Group.

[30] Qi Dong, Shaogang Gong, and Xiatian Zhu. Imbalanced Deep Learning by Minority Class Incremental Rectification. *arXiv:1804.10851 [cs]*, April 2018. arXiv: 1804.10851.

[31] René Dreos, Giovanna Ambrosini, Rouayda Perier, and Philipp Bucher. The Eukaryotic Promoter Database: expansion of EPDnew and new promoter analysis tools. January 2015.

[32] René Dréos, Giovanna Ambrosini, Romain Groux, Rouayda Cavin Périer, and Philipp Bucher. MGA repository: a curated data resource for ChIP-seq and other genome annotated data. *Nucleic Acids Research*, 46(D1):D175–D180, January 2018.

[33] Sascha H. Duttke, Max W. Chang, Sven Heinz, and Christopher Benner. Identification and dynamic quantification of regulatory elements using total RNA. *Genome Research*, October 2019.

[34] S. Döhr, A. Klingenhoff, H. Maier, M. Hrabé de Angelis, T. Werner, and R. Schneider. Linking disease-associated genes to regulatory networks via promoter organization. *Nucleic Acids Research*, 33(3):864–872, 2005.

[35] Gerda Egger, Gangning Liang, Ana Aparicio, and Peter A. Jones. Epigenetics in human disease and prospects for epigenetic therapy. *Nature*, 429(6990):457–463, May 2004.

[36] ENCODE Project Consortium. An integrated encyclopedia of DNA elements in the human genome. *Nature*, 489(7414):57–74, September 2012.

[37] Richard Evans, John Jumper, James Kirkpatrick, Laurent Sifre, Tim Green, Chongli Qin, Augustin Zidek, Sandy Nelson, Alex Bridgland, Hugo Penedones, Stig Petersen, Karen Simonyan, Steve Crossan, David Jones, David Silver, Koray Kavukcuoglu, Demis Hassabis, and Andrew Senior. De novo structure prediction with deeplearning based scoring. *Thirteenth Critical Assessment of Techniques for Protein Structure Prediction*, December 2018.

[38] Tom Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8):861–874, June 2006.

[39] Oriol Fornes, Jaime A. Castro-Mondragon, Aziz Khan, Robin van der Lee, Xi Zhang, Phillip A. Richmond, Bhavi P. Modi, Solenne Correard, Marius Gheorghe, Damir Baranašić, Walter Santana-Garcia, Ge Tan, Jeanne Chèneby, Benoit Ballester, François Parcy, Albin Sandelin, Boris Lenhard, Wyeth W. Wasserman, and Anthony Mathelier. JASPAR 2020: update of the open-access database of transcription factor binding profiles. *Nucleic Acids Research*, 2019.

[40] Oriol Fornes, Marius Gheorghe, Phillip A. Richmond, David J. Arenillas, Wyeth W. Wasserman, and Anthony Mathelier. MANTA2, update of the Mongo database for the analysis of transcription factor binding site alterations. *Scientific Data*, 5(1):1–7, July 2018.

[41] Rosalind E. Franklin and R. G. Gosling. Molecular Configuration in Sodium Thymonucleate. *Nature*, 171(4356):740–741, April 1953.

[42] Terrence S. Furey. ChIP-seq and Beyond: new and improved methodologies to detect and characterize protein-DNA interactions. *Nature reviews. Genetics*, 13(12):840–852, December 2012.

[43] D J Galas and A Schmitz. DNAse footprinting: a simple method for the detection of protein-DNA binding specificity. *Nucleic Acids Research*, 5(9):3157–3170, September 1978.

[44] Andrea Galassi, Marco Lippi, and Paolo Torroni. Attention, please! A Critical Review of Neural Attention Models in Natural Language Processing. *arXiv:1902.02181 [cs, stat]*, February 2019. arXiv: 1902.02181.

[45] Austen R. D. Ganley and Takehiko Kobayashi. Phylogenetic footprinting to find functional DNA elements. *Methods in Molecular Biology (Clifton, N.J.)*, 395:367–380, 2007.

[46] Nico Geisel. Constitutive versus Responsive Gene Expression Strategies for Growth in Changing Environments. *PLoS ONE*, 6(11), November 2011.

[47] J. Geurts, O. J. Arntz, M. B. Bennink, L. a. B. Joosten, W. B. van den Berg, and F. A. J. van de Loo. Application of a disease-regulated promoter is a safer mode of local IL-4 gene therapy for arthritis. *Gene Therapy*, 14(23):1632–1638, December 2007.

[48] Leilani H. Gilpin, David Bau, Ben Z. Yuan, Ayesha Bajwa, Michael Specter, and Lalana Kagal. Explaining Explanations: An Overview of Interpretability of Machine Learning. In *2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)*, pages 80–89, October 2018.

[49] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[50] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. LSTM: A Search Space Odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 28(10):2222–2232, October 2017. arXiv: 1503.04069.

[51] Zhi-Jian Han, Yan-Hu Feng, Bao-Hong Gu, Yu-Min Li, and Hao Chen. The post-translational modification, SUMOylation, and cancer (Review). *International Journal of Oncology*, 52(4):1081–1094, April 2018. Publisher: Spandidos Publications.

[52] David J Hand. A Simple Generalisation of the Area Under the ROC Curve for Multiple Class Classification Problems. page 16, 2001.

[53] Nathan Harmston, Wendy Filsell, and Michael Stumpf. *What the papers say: Text mining for genomics and systems biology*, volume 5. October 2010.

[54] Edith Heard and Robert A. Martienssen. Transgenerational Epigenetic Inheritance: Myths and Mechanisms. *Cell*, 157(1):95–109, March 2014.

[55] Moritz Hess, Stefan Lenz, Tamara J. Blätte, Lars Bullinger, and Harald Binder. Partitioned learning of deep Boltzmann machines for SNP data. *Bioinformatics*, 33(20):3173–3180, October 2017.

[56] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, November 1997.

[57] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, January 1991.

[58] Hui Hu, Ya-Ru Miao, Long-Hao Jia, Qing-Yang Yu, Qiong Zhang, and An-Yuan Guo. AnimalTFDB 3.0: a comprehensive resource for annotation and prediction of animal transcription factors. *Nucleic Acids Research*, 47(D1):D33–D38, January 2019.

[59] D. H. Hubel and T. N. Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of Physiology*, 148(3):574–591, October 1959.

[60] Maxwell A. Hume, Luis A. Barrera, Stephen S. Gisselbrecht, and Martha L. Bulyk. UniPROBE, update 2015: new tools and content for the online database of protein-binding microarray data on protein-DNA interactions. *Nucleic Acids Research*, 43(Database issue):D117–122, January 2015.

[61] Institute of Medicine (US) Committee on Accelerating Rare Diseases Research and Orphan Product Development, Marilyn J. Field, and Thomas F. Boat. *Profile of Rare Diseases*. National Academies Press (US), 2010. Publication Title: Rare Diseases and Orphan Products: Accelerating Research and Development.

[62] Henry Jakubowski. *Biochemistry Online*. Biology LibreTexts, June 2016. Library Catalog: bio.libretexts.org.

[63] Minghui Jiang, James Anderson, Joel Gillespie, and Martin Mayne. uShuffle: A useful tool for shuffling biological sequences while preserving the k-let counts. *BMC Bioinformatics*, 9:192, April 2008.

[64] Shan Jiang and Ali Mortazavi. Integrating ChIP-seq with other functional genomics data. *Briefings in Functional Genomics*, 17(2):104–115, March 2018.

[65] Justin M. Johnson and Taghi M. Khoshgoftaar. Survey on deep learning with class imbalance. *Journal of Big Data*, 6(1):27, March 2019.

[66] Tamar Juven-Gershon and James T. Kadonaga. Regulation of Gene Expression via the Core Promoter and the Basal Transcriptional Machinery. *Developmental biology*, 339(2):225–229, March 2010.

[67] Aditi Kanhere and Manju Bansal. Structural properties of promoters: similarities and differences between prokaryotes and eukaryotes. *Nucleic Acids Research*, 33(10):3165–3175, 2005.

[68] Andrej Karpathy, Justin Johnson, and Li Fei-Fei. Visualizing and Understanding Recurrent Networks. June 2015.

[69] Kiyoshi Kawaguchi. A multithreaded software model for backpropagation neural network applications, 2000.

[70] Alexander Kel, Ulyana Boyarskikh, Philip Stegmaier, Leonid S. Leskov, Andrey V. Sokolov, Ivan Yevshin, Nikita Mandrik, Daria Stelmashenko, Jeannette Koschmann, Olga Kel-Margoulis, Mathias Krull, Anna Martínez-Cardús, Sebastian Moran, Manel Esteller, Fedor Kolpakov, Maxim Filipenko, and Edgar Wingender. Walking pathways with positive feedback loops reveal DNA methylation biomarkers of colorectal cancer. *BMC Bioinformatics*, 20(4):119, April 2019.

[71] Tae-Kyung Kim, Martin Hemberg, and Jesse M. Gray. Enhancer RNAs: A Class of Long Noncoding RNAs Synthesized at Enhancers. *Cold Spring Harbor Perspectives in Biology*, 7(1):a018622, January 2015. Company: Cold Spring Harbor Laboratory Press Distributor: Cold Spring Harbor Laboratory Press Institution: Cold Spring Harbor Laboratory Press Label: Cold Spring Harbor Laboratory Press Publisher: Cold Spring Harbor Lab.

[72] Jan Kuka&#x10D, ka, Vladimir Golkov, and Daniel Cremers. Regularization for Deep Learning: A Taxonomy. February 2018.

[73] Ivan V. Kulakovskiy, Ilya E. Vorontsov, Ivan S. Yevshin, Ruslan N. Sharipov, Alla D. Fedorova, Eugene I. Rumynskiy, Yulia A. Medvedeva, Arturo Magana-Mora, Vladimir B. Bajic, Dmitry A. Papatsenko, Fedor A. Kolpakov, and Vsevolod J. Makeev. HOCOMOCO: towards a complete collection of transcription factor binding models for human and mouse via large-scale ChIP-Seq analysis. *Nucleic Acids Research*, 46(D1):D252–D259, January 2018.

[74] Mensah Kwabena Patrick, Adebayo Felix Adekoya, Ayidzoe Abra Mighty, and Baagyire Y. Edward. Capsule Networks – A survey. *Journal of King Saud University - Computer and Information Sciences*, September 2019.

[75] Hong-Yan Lai, Zhao-Yue Zhang, Zhen-Dong Su, Wei Su, Hui Ding, Wei Chen, and Hao Lin. iProEP: A Computational Predictor for Predicting Promoter. *Molecular Therapy - Nucleic Acids*, 17:337–346, September 2019.

[76] Samuel A. Lambert, Arttu Jolma, Laura F. Campitelli, Pratyush K. Das, Yimeng Yin, Mihai Albu, Xiaoting Chen, Jussi Taipale, Timothy R. Hughes, and Matthew T. Weirauch. The Human Transcription Factors. *Cell*, 172(4):650–665, February 2018.

[77] Nguyen Quoc Khanh Le, Edward Kien Yee Yapp, N. Nagasundaram, and Hui-Yuan Yeh. Classifying Promoters by Interpreting the Hidden Information of DNA Sequences via Deep Learning and Combination of Continuous FastText N-Grams. *Frontiers in Bioengineering and Biotechnology*, 7, 2019.

[78] Qun Li, Iain J. Clarke, and A. Ian Smith. Acetylation. In *Handbook of Biologically Active Peptides*, pages 1711–1714. Elsevier, 2013.

[79] Yifeng Li, Chih-yu Chen, Alice M. Kaye, and Wyeth W. Wasserman. The identification of cis-regulatory elements: A review from a machine learning perspective. *Biosystems*, 138:6–17, December 2015.

[80] Hao Lin, Zhi-Yong Liang, Hua Tang, and Wei Chen. Identifying Sigma70 Promoters with Novel Pseudo Nucleotide Composition. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 16(4):1316–1321, July 2019.

[81] Bin Liu, Fan Yang, De-Shuang Huang, and Kuo-Chen Chou. iPromoter-2L: a two-layer predictor for identifying promoters and their types by multi-window-based PseKNC. *Bioinformatics*, 34(1):33–40, January 2018.

[82] Christopher D. Manning, Christopher D. Manning, and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999. Google-Books-ID: YiFDxbEX3SUC.

[83] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, December 1943.

[84] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. *arXiv:1301.3781 [cs]*, September 2013. arXiv: 1301.3781.

[85] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997. Google-Books-ID: EoYBngEACAAJ.

[86] Czuee Morey, Sushmita Mookherjee, Ganesan Rajasekaran, and Manju Bansal. DNA Free Energy-Based Promoter Prediction and Comparative Analysis of Arabidopsis and Rice Genomes. *Plant Physiology*, 156(3):1300–1315, July 2011.

[87] W. James Murdoch, Chandan Singh, Karl Kumbier, Reza Abbasi-Asl, and Bin Yu. Definitions, methods, and applications in interpretable machine learning. *Proceedings of the National Academy of Sciences*, 116(44):22071–22080, October 2019.

[88] Marwa Naili, Anja Habacha Chaibi, and Henda Hajjami Ben Ghezala. Comparative study of word embedding methods in topic segmentation. *Procedia Computer Science*, 112:340–349, 2017.

[89] Saul B. Needleman and Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, March 1970.

[90] Patrick Ng. dna2vec: Consistent vector representations of variable-length k-mers. *arXiv:1701.06279 [cs, q-bio, stat]*, January 2017. arXiv: 1701.06279.

[91] Nobuo Ogawa and Mark D. Biggin. High-throughput SELEX determination of DNA sequences bound by transcription factors in vitro. *Methods in Molecular Biology (Clifton, N.J.)*, 786:51–63, 2012.

[92] Uwe Ohler and Heinrich Niemann. Identification and analysis of eukaryotic promoters: recent computational approaches. *Trends in Genetics*, 17(2):56–60, February 2001.

[93] Mhaned Oubounyt, Zakaria Louadi, Hilal Tayara, and Kil To Chong. DeePromoter: Robust Promoter Predictor Using Deep Learning. *Frontiers in genetics*, 10:286–286, April 2019.

[94] Mrutyunjaya Panda. Elephant Search with Deep Learning for Microarray Data Analysis. *arXiv:1707.03604 [cs, q-bio]*, July 2017. arXiv: 1707.03604.

[95] William R. Pearson. An Introduction to Sequence Similarity ("Homology") Searching. *Current protocols in bioinformatics / editoral board, Andreas D. Baxevanis ... [et al.]*, 0 3, June 2013.

[96] Anders Gorm Pedersen, Pierre Baldi, Yves Chauvin, and Søren Brunak. The biology of eukaryotic promoter prediction—a review. *Computers & Chemistry*, 23(3):191–207, June 1999.

[97] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, 2014. Association for Computational Linguistics.

[98] Christopher L. Plaisier, Sofie O'Brien, Brady Bernard, Sheila Reynolds, Zac Simon, Chad M. Toledo, Yu Ding, David J. Reiss, Patrick J. Paddison, and Nitin S. Baliga. Causal Mechanistic Regulatory Network for Glioblastoma Deciphered Using Systems Genetics Network Analysis. *Cell Systems*, 3(2):172–186, August 2016.

[99] Chris P. Ponting and Leo Goodstadt. Separating derived from ancestral features of mouse and human genomes. *Biochemical Society Transactions*, 37(4):734–739, August 2009.

[100] Ying Qian, Yu Zhang, Bingyu Guo, Shasha Ye, Yuzhu Wu, and Jiongmin Zhang. An Improved Promoter Recognition Model Using Convolutional Neural Network. In *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, pages 471–476, Tokyo, Japan, July 2018. IEEE.

[101] Brian J. Raney, Timothy R. Dreszer, Galt P. Barber, Hiram Clawson, Pauline A. Fujita, Ting Wang, Ngan Nguyen, Benedict Paten, Ann S. Zweig, Donna Karolchik, and W. James Kent. Track data hubs enable visualization of user-defined genome-wide annotations on the UCSC Genome Browser. *Bioinformatics*, 30(7):1003–1005, April 2014.

[102] Vetriselvi Rangannan and Manju Bansal. Relative stability of DNA as a generic criterion for promoter prediction: whole genome annotation of micro-

bial genomes with varying nucleotide base composition. *Molecular bioSystems*, 5(12):1758–1769, December 2009.

[103] Jie Ren, Peter J Liu, Emily Fertig, Jasper Snoek, Ryan Poplin, Mark Depristo, Joshua Dillon, and Balaji Lakshminarayanan. Likelihood Ratios for Out-of-Distribution Detection. page 12.

[104] Jason A. Reuter, Damek Spacek, and Michael P. Snyder. High-Throughput Sequencing Technologies. *Molecular cell*, 58(4):586–597, May 2015.

[105] Keith D Robertson. DNA methylation and chromatin – unraveling the tangled web. *Oncogene*, 21(35):5361–5379, August 2002.

[106] Pau Rodríguez, Miguel A. Bautista, Jordi Gonzàlez, and Sergio Escalera. Beyond one-hot encoding: Lower dimensional target embedding. *Image and Vision Computing*, 75:21–31, July 2018.

[107] Stephane Rombauts, Kobe Florquin, Magali Lescot, Kathleen Marchal, Pierre Rouzé, and Yves Van de Peer. Computational Approaches to Identify Promoters and cis-Regulatory Elements in Plant Genomes. *Plant Physiology*, 132(3):1162–1176, July 2003.

[108] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[109] Andrew Slavin Ross, Michael C. Hughes, and Finale Doshi-Velez. Right for the Right Reasons: Training Differentiable Models by Constraining their Explanations. pages 2662–2670, 2017.

[110] Ananda L. Roy and Dinah S. Singer. Core Promoters in Transcription: Old Problem, New Insights. *Trends in biochemical sciences*, 40(3):165–171, March 2015.

[111] David E. Rumelhart and James L. McClelland. Learning Internal Representations by Error Propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*, pages 318–362. MITP, 1987. ISSN: null.

[112] Tara N. Sainath, Oriol Vinyals, Andrew Senior, and Haşim Sak. Convolutional, long short-term memory, fully connected deep neural networks. In *2015 IEEE*

International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 4580–4584. IEEE, 2015.

[113] Masakazu Sato, Koji Horie, Aki Hara, Yuichiro Miyamoto, Kazuko Kurihara, Kensuke Tomio, and Harushige Yokota. Application of deep learning to the classification of images from colposcopy. *Oncology Letters*, 15(3):3518–3523, March 2018. Publisher: Spandidos Publications.

[114] M. Schuster and K.K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, November 1997.

[115] Eran Segal, Yoseph Barash, Itamar Simon, Nir Friedman, and Daphne Koller. From promoter sequence to expression: a probabilistic framework. In *Proceedings of the sixth annual international conference on Computational biology - RECOMB '02*, pages 263–272, Washington, DC, USA, 2002. ACM Press.

[116] Roded Sharan, Shaul Karni, and Yifat Felder. Analysis of Biological Networks: Transcriptional Networks - Promoter Sequence Analysis, January 2007.

[117] Roded Sharan, Ivan Ovcharenko, Asa Ben-Hur, and Richard M. Karp. CREME: a framework for identifying cis-regulatory modules in human-mouse conserved segments. *Bioinformatics*, 19(suppl_1):i283–i291, July 2003.

[118] Alex Sherstinsky. Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network. August 2018.

[119] Chance Simmons and Mark A. Holliday. A Comparison of Two Popular Machine Learning Frameworks. *The Journal of Computing Sciences in Colleges*, page 20, 2019.

[120] Jeremy M. Simon, Paul G. Giresi, Ian J. Davis, and Jason D. Lieb. Using FAIRE (Formaldehyde-Assisted Isolation of Regulatory Elements) to isolate active regulatory DNA. *Nature protocols*, 7(2):256–267, January 2012.

[121] Anna Sloutskin, Yehuda M. Danino, Yaron Orenstein, Yonathan Zehavi, Tirza Doniger, Ron Shamir, and Tamar Juven-Gershon. ElemeNT: a computational tool for detecting core promoter elements. *Transcription*, 6(3):41–50, May 2015.

[122] Shannon Soucy, Lorraine Olendzenski, and J. Peter Gogarten. Orthologues, Paralogues and Horizontal Gene Transfer in the Human Holobiont. In *eLS*. American Cancer Society, 2013.

[123] Ralf C. Staudemeyer and Eric Rothstein Morris. Understanding LSTM – a tutorial into Long Short-Term Memory Recurrent Neural Networks. September 2019.

[124] Derek Martin Stein. *Ion beam sculpting molecular scale devices*. PhD thesis, 2003. ISSN: 9780493976495 Journal Abbreviation: Ph.D. Thesis Publication Title: Ph.D. Thesis.

[125] Alexander J. Stewart, Sridhar Hannenhalli, and Joshua B. Plotkin. Why Transcription Factor Binding Sites Are Ten Nucleotides Long. *Genetics*, 192(3):973–985, November 2012.

[126] Alessandra M. Sullivan, Kerry L. Bubb, Richard Sandstrom, John A. Stamatoyannopoulos, and Christine Queitsch. DNase I hypersensitivity mapping, genomic footprinting, and transcription factor networks in plants. *Current Plant Biology*, 3-4:40–47, September 2015.

[127] Tanlin Sun, Bo Zhou, Luhua Lai, and Jianfeng Pei. Sequence-based prediction of protein protein interaction using a deep-learning algorithm. *BMC Bioinformatics*, 18(1):277–277, May 2017.

[128] Ayako Suzuki, Shin Kawano, Toutai Mitsuyama, Mikita Suyama, Yae Kanai, Katsuhiko Shirahige, Hiroyuki Sasaki, Katsushi Tokunaga, Katsuya Tsuchihara, Sumio Sugano, Kenta Nakai, and Yutaka Suzuki. DBTSS/DBKERO for integrated analysis of transcriptional regulation. *Nucleic Acids Research*, 46(D1):D229–D238, 2018.

[129] Hazuki Takahashi, Sachiko Kato, Mitsuyoshi Murata, and Piero Carninci. CAGE- Cap Analysis Gene Expression: a protocol for the detection of promoter and transcriptional networks. *Methods in molecular biology (Clifton, N.J.)*, 786:181–200, 2012.

[130] Hilal Tayara, Muhammad Tahir, and Kil To Chong. Identification of prokaryotic promoters and their strength by integrating heterogeneous features. *Genomics*, August 2019.

[131] The FANTOM Consortium and the RIKEN PMI and CLST (DGT). A promoter-level mammalian expression atlas. *Nature*, 507(7493):462–470, March 2014.

[132] Lindsey S. Treviño, Quan Wang, and Cheryl L. Walker. Phosphorylation of Epigenetic "Readers, Writers and Erasers": Implications for Developmental Reprogramming and the Epigenetic Basis for Health and Disease. *Progress in biophysics and molecular biology*, 118(0):8–13, July 2015.

[133] Michael Tschannen, Olivier Bachem, and Mario Lucic. Recent Advances in Autoencoder-Based Representation Learning. *arXiv:1812.05069 [cs, stat]*, December 2018. arXiv: 1812.05069.

[134] Ramzan Umarov, Hiroyuki Kuwahara, Yu Li, Xin Gao, and Victor Solovyev. Promoter analysis and prediction in the human genome using sequence-based deep learning models. *Bioinformatics*, 35(16):2730–2737, August 2019.

[135] Ramzan Kh. Umarov and Victor V. Solovyev. Recognition of prokaryotic and eukaryotic promoters using convolutional deep learning neural networks. *PLOS ONE*, 12(2):e0171410, February 2017.

[136] Gilles Vandewiele, Isabelle Dehaene, György Kovács, Lucas Sterckx, Olivier Janssens, Femke Ongenae, Femke De Backere, Filip De Turck, Kristien Roelens, Johan Decruyenaere, Sofie Van Hoecke, and Thomas Demeester. Overly Optimistic Prediction Results on Imbalanced Data: Flaws and Benefits of Applying Over-sampling. *arXiv:2001.06296 [cs, eess, stat]*, January 2020. arXiv: 2001.06296.

[137] Anne Vanet, Laurent Marsan, and Marie-France Sagot. Promoter sequences and algorithmical methods for identifying them. *Research in Microbiology*, 150(9-10):779–799, November 1999.

[138] Vincent Vedel and Ivan Scotti. Promoting the promoter. *Plant Science*, 180(2):182–189, February 2011.

[139] Fjodor van Veen. The Neural Network Zoo, September 2016.

[140] Fjodor van Veen. Neural Network Zoo Prequel: Cells and Layers, March 2017.

[141] Bin Wang, Angela Wang, Fenxiao Chen, Yuncheng Wang, and C.-C. Jay Kuo. Evaluating Word Embedding Models: Methods and Experimental Results. *APSIPA Transactions on Signal and Information Processing*, 8:e19, 2019. arXiv: 1901.09785.

[142] Zhengwei Wang, Qi She, and Tomas E. Ward. Generative Adversarial Networks in Computer Vision: A Survey and Taxonomy. *arXiv:1906.01529 [cs]*, February 2020. arXiv: 1906.01529.

[143] Zhong Wang, Mark Gerstein, and Michael Snyder. RNA-Seq: a revolutionary tool for transcriptomics. *Nature reviews. Genetics*, 10(1):57–63, January 2009.

[144] Wyeth W. Wasserman and William Krivan. In silico identification of metazoan transcriptional regulatory regions. *Naturwissenschaften*, 90(4):156–166, April 2003.

[145] Wyeth W. Wasserman and Albin Sandelin. Applied bioinformatics for the identification of regulatory elements. *Nature Reviews Genetics*, 5(4):276–287, April 2004. Number: 4 Publisher: Nature Publishing Group.

[146] J. D. Watson and F. H. C. Crick. Molecular Structure of Nucleic Acids: A Structure for Deoxyribose Nucleic Acid. *Nature*, 171(4356):737–738, April 1953.

[147] Bob Weinhold. Epigenetics: The Science of Change. *Environmental Health Perspectives*, 114(3):A160–A167, March 2006.

[148] Gary M. Weiss. Mining with rarity: a unifying framework. *ACM SIGKDD Explorations Newsletter*, 6(1):7–19, June 2004.

[149] Lilian Weng. Attention? Attention!, June 2018.

[150] Bernard Widrow. Adaptive" adaline" Neuron Using Chemical" memistors.". Technical report, 1960.

[151] M. H. F. Wilkins, A. R. Stokes, and H. R. Wilson. Molecular Structure of Nucleic Acids: Molecular Structure of Deoxypentose Nucleic Acids. *Nature*, 171(4356):738–740, April 1953.

[152] Xuan Xiao, Zhao-Chun Xu, Wang-Ren Qiu, Peng Wang, Hui-Ting Ge, and Kuo-Chen Chou. iPSW(2L)-PseKNC: A two-layer predictor for identifying

promoters and their strength by hybrid features via pseudo K-tuple nucleotide composition. *Genomics*, 111(6):1785–1793, December 2019.

[153] Wenxuan Xu, Li Zhang, and Yaping Lu. SD-MSAEs: Promoter recognition in human genome based on deep feature extraction. *Journal of Biomedical Informatics*, 61:55–62, June 2016.

[154] Wenxuan Xu, Lin Zhu, and De-Shuang Huang. DCDE: An Efficient Deep Convolutional Divergence Encoding Method for Human Promoter Recognition. *IEEE Transactions on NanoBioscience*, 18(2):136–145, April 2019.

[155] Xinyi Yang and Annalisa Marsico. In Silico Promoter Recognition from deepCAGE Data. *Methods in Molecular Biology (Clifton, N.J.)*, 1468:171–199, 2017.

[156] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical Attention Networks for Document Classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489, San Diego, California, June 2016. Association for Computational Linguistics.

[157] Venkata Rajesh Yella and Manju Bansal. In silico Identification of Eukaryotic Promoters. December 2014.

[158] Venkata Rajesh Yella, Aditya Kumar, and Manju Bansal. Identification of putative promoters in 48 eukaryotic genomes on the basis of DNA free energy. *Scientific Reports*, 8, March 2018.

[159] Ivan Yevshin, Ruslan Sharipov, Semyon Kolmykov, Yury Kondrakhin, and Fedor Kolpakov. GTRD: a database on gene transcription regulation—2019 update. *Nucleic Acids Research*, 47(D1):D100–D105, January 2019.

[160] Benedikt Zacher, Margaux Michel, Björn Schwalb, Patrick Cramer, Achim Tresch, and Julien Gagneur. Accurate Promoter and Enhancer Identification in 127 ENCODE and Roadmap Epigenomics Cell Types and Tissues by GenoSTAN. *PLOS ONE*, 12(1):e0169249, January 2017. Publisher: Public Library of Science.

[161] Jia Zeng, Xiao-Qin Cao, and Hong Yan. Human Promoter Recognition using Kullback-Leibler Divergence. In *2007 International Conference on Machine Learning and Cybernetics*, volume 6, pages 3319–3325, August 2007. ISSN: 2160-1348.

[162] Jia Zeng, Xiao-Yu Zhao, Xiao-Qin Cao, and Hong Yan. SCS: Signal, Context, and Structure Features for Genome-Wide Human Promoter Recognition. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 7(3):550–562, July 2010.

[163] Jia Zeng, Shanfeng Zhu, and Hong Yan. Towards accurate human promoter recognition: a review of currently used sequence features and classification methods. *Briefings in Bioinformatics*, 10(5):498–508, September 2009.

[164] Daniel R. Zerbino, Premanand Achuthan, Wasiu Akanni, M. Ridwan Amode, Daniel Barrell, Jyothish Bhai, Konstantinos Billis, Carla Cummins, Astrid Gall, Carlos García Girón, Laurent Gil, Leo Gordon, Leanne Haggerty, Erin Haskell, Thibaut Hourlier, Osagie G. Izuogu, Sophie H. Janacek, Thomas Juettemann, Jimmy Kiang To, Matthew R. Laird, Ilias Lavidas, Zhicheng Liu, Jane E. Loveland, Thomas Maurel, William McLaren, Benjamin Moore, Jonathan Mudge, Daniel N. Murphy, Victoria Newman, Michael Nuhn, Denye Ogeh, Chuang Kee Ong, Anne Parker, Mateus Patricio, Harpreet Singh Riat, Helen Schuilenburg, Dan Sheppard, Helen Sparrow, Kieron Taylor, Anja Thormann, Alessandro Vullo, Brandon Walts, Amonida Zadissa, Adam Frankish, Sarah E. Hunt, Myrto Kostadima, Nicholas Langridge, Fergal J. Martin, Matthieu Muffato, Emily Perry, Magali Ruffier, Dan M. Staines, Stephen J. Trevanion, Bronwen L. Aken, Fiona Cunningham, Andrew Yates, and Paul Flicek. Ensembl 2018. *Nucleic Acids Research*, 46(D1):D754–D761, January 2018.

[165] David W. Zhang, Juan B. Rodríguez-Molina, Joshua R. Tietjen, Corey M. Nemec, and Aseem Z. Ansari. Emerging Views on the CTD Code, 2012.

[166] Quan-shi Zhang and Song-chun Zhu. Visual interpretability for deep learning: a survey. *Frontiers of Information Technology & Electronic Engineering*, 19(1):27–39, January 2018.

[167] Xiujun Zhang. Position Weight Matrices. In Werner Dubitzky, Olaf Wolkenhauer, Kwang-Hyun Cho, and Hiroki Yokota, editors, *Encyclopedia of Systems Biology*, pages 1721–1722. Springer, New York, NY, 2013.

[168] Xiao-yu Zhao, Jin Zhang, Yuan-yuan Chen, Qiang Li, Tao Yang, Cong Pian, and Liang-yun Zhang. Promoter recognition based on the maximum entropy hidden Markov model. *Computers in Biology and Medicine*, 51:73–81, August 2014.

[169] Yuming Zhao, Fang Wang, Su Chen, Jun Wan, and Guohua Wang. Methods of MicroRNA Promoter Prediction and Transcription Factor Mediated Regulatory Network, 2017. ISSN: 2314-6133 Library Catalog: www.hindawi.com Pages: e7049406 Publisher: Hindawi Volume: 2017.

[170] Peng Zhou, Wei Shi, Jun Tian, Zhenyu Qi, Bingchen Li, Hongwei Hao, and Bo Xu. Attention-Based Bidirectional Long Short-Term Memory Networks for Relation Classification. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 207–212, Berlin, Germany, 2016. Association for Computational Linguistics.

Appendix A

Reports on comparisons of approaches from the literature

A.1 Implemented CNNProm model results

```
Metrics report for trained model (evaluation/cnnprom/deepromoter.csv) .
Threshold set to 0.50
              precision    recall  f1-score   support

non-promoter      0.722     0.304     0.428     29597
    promoter      0.559     0.883     0.685     29597

    accuracy                          0.594     59194
   macro avg      0.641     0.594     0.556     59194
weighted avg      0.641     0.594     0.556     59194

Accuracy score: 0.594
Balanced Accuracy score: 0.594
F1 score: 0.685
Matthews Coefficient Correlation: 0.229
Precision score: 0.559
Recall score: 0.883
F-beta score: 0.685
Jaccard Similarity score: 0.521
```

Figure A.1: Report with results from our reproduced CNNProm model tested on DeePromoter data

```
Metrics report for trained model (evaluation/cnnprom/icnn.csv) .
Threshold set to 0.50
              precision    recall  f1-score   support

non-promoter      0.925     0.882     0.903      5131
    promoter      0.918     0.949     0.933      7156

    accuracy                          0.921     12287
   macro avg      0.921     0.915     0.918     12287
weighted avg      0.921     0.921     0.920     12287

Accuracy score: 0.921
Balanced Accuracy score: 0.915
F1 score: 0.933
Matthews Coefficient Correlation: 0.836
Precision score: 0.918
Recall score: 0.949
F-beta score: 0.933
Jaccard Similarity score: 0.874
```

Figure A.2: Report with results from our reproduced CNNProm model tested on ICNN data

A.2 Implemented ICNN model results

```
Metrics report for trained model (evaluation/icnn/deepromoter.csv) .
Threshold set to 0.50
              precision    recall  f1-score   support

non-promoter      0.706     0.476     0.569     29597
    promoter      0.605     0.802     0.689     29597

    accuracy                          0.639     59194
   macro avg      0.655     0.639     0.629     59194
weighted avg      0.655     0.639     0.629     59194

Accuracy score: 0.639
Balanced Accuracy score: 0.639
F1 score: 0.689
Matthews Coefficient Correlation: 0.294
Precision score: 0.605
Recall score: 0.802
F-beta score: 0.689
Jaccard Similarity score: 0.526
```

Figure A.3: Report with results from our reproduced ICNN model tested on DeePromoter data

```
Metrics report for trained model (evaluation/icnn/cnnprom.csv) .
Threshold set to 0.50
              precision    recall  f1-score   support

non-promoter      0.849     0.917     0.882     35987
    promoter      0.888     0.802     0.843     29597

    accuracy                          0.865     65584
   macro avg      0.868     0.859     0.862     65584
weighted avg      0.867     0.865     0.864     65584

Accuracy score: 0.865
Balanced Accuracy score: 0.859
F1 score: 0.843
Matthews Coefficient Correlation: 0.728
Precision score: 0.888
Recall score: 0.802
F-beta score: 0.843
Jaccard Similarity score: 0.728
```

Figure A.4: Report with results from our reproduced ICNN model tested on CNNProm data

A.3 Implemented DeePromoter model results

```
Metrics report for trained model (evaluation/deepromoter/icnn.csv) .
Threshold set to 0.50
              precision    recall  f1-score   support

non-promoter      0.585     0.064     0.115      5131
    promoter      0.590     0.968     0.733      7156

    accuracy                          0.590     12287
   macro avg      0.588     0.516     0.424     12287
weighted avg      0.588     0.590     0.475     12287

Accuracy score: 0.590
Balanced Accuracy score: 0.516
F1 score: 0.733
Matthews Coefficient Correlation: 0.074
Precision score: 0.590
Recall score: 0.968
F-beta score: 0.733
Jaccard Similarity score: 0.579
```

Figure A.5: Report with results from our reproduced DeePromoter model tested on ICNN data

```
Metrics report for trained model (evaluation/deepromoter/cnnprom.csv) .
Threshold set to 0.50
              precision    recall  f1-score   support

non-promoter      0.536     0.035     0.067     35987
    promoter      0.451     0.963     0.614     29597

    accuracy                          0.454     65584
   macro avg      0.493     0.499     0.340     65584
weighted avg      0.498     0.454     0.314     65584

Accuracy score: 0.454
Balanced Accuracy score: 0.499
F1 score: 0.614
Matthews Coefficient Correlation: -0.005
Precision score: 0.451
Recall score: 0.963
F-beta score: 0.614
Jaccard Similarity score: 0.443
```

Figure A.6: Report with results from our reproduced DeePromoter model tested on CNNProm data

Appendix B

Complete results on testing dataset

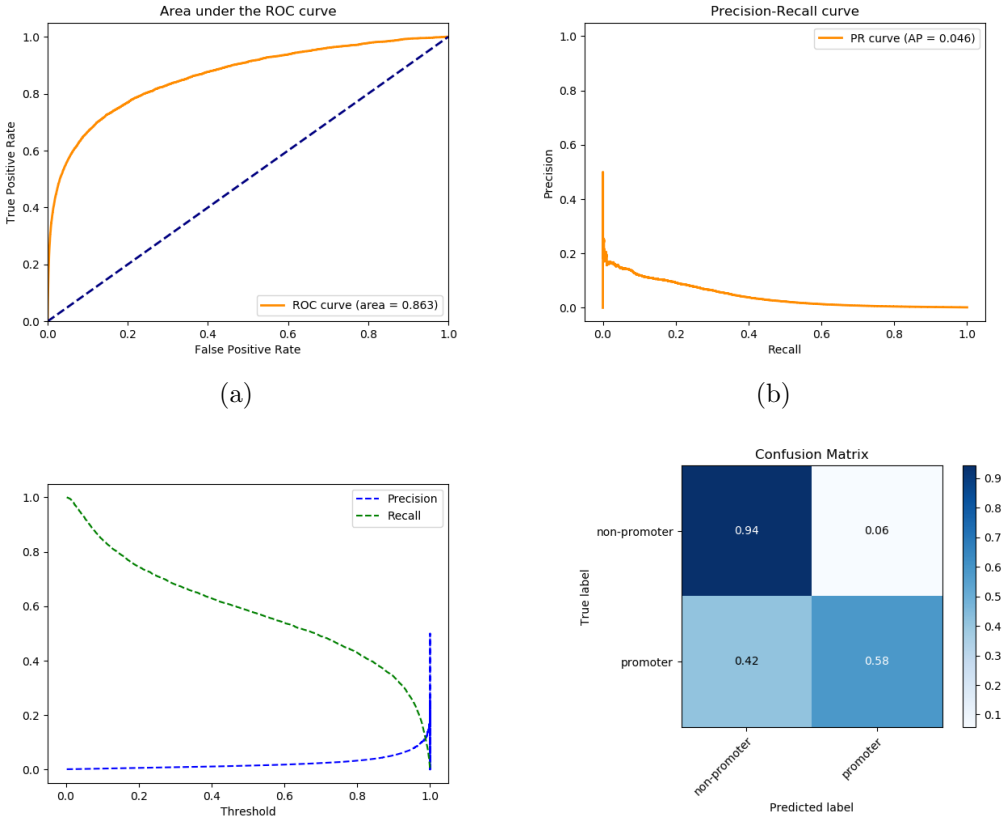


Figure B.1: Results from the reproduced CNNProm model tested on **our** data: (a) shows the AUC; (b) shows the PR curve; (c) shows precision and recall by threshold; (d) shows the confusion matrix.

```
Metrics report for trained model (evaluation/cnnprom/ours.csv) .
Threshold set to 0.50
              precision    recall  f1-score   support

non-promoter      0.999     0.943     0.970   4602408
    promoter      0.014     0.584     0.028      6599

    accuracy                          0.942   4609007
   macro avg      0.507     0.764     0.499   4609007
weighted avg      0.998     0.942     0.969   4609007

Accuracy score: 0.942
Balanced Accuracy score: 0.764
F1 score: 0.028
Matthews Coefficient Correlation: 0.085
Precision score: 0.014
Recall score: 0.584
F-beta score: 0.028
Jaccard Similarity score: 0.014
```

Figure B.2: Report with results from our reproduced CNNProm model tested on **our** data

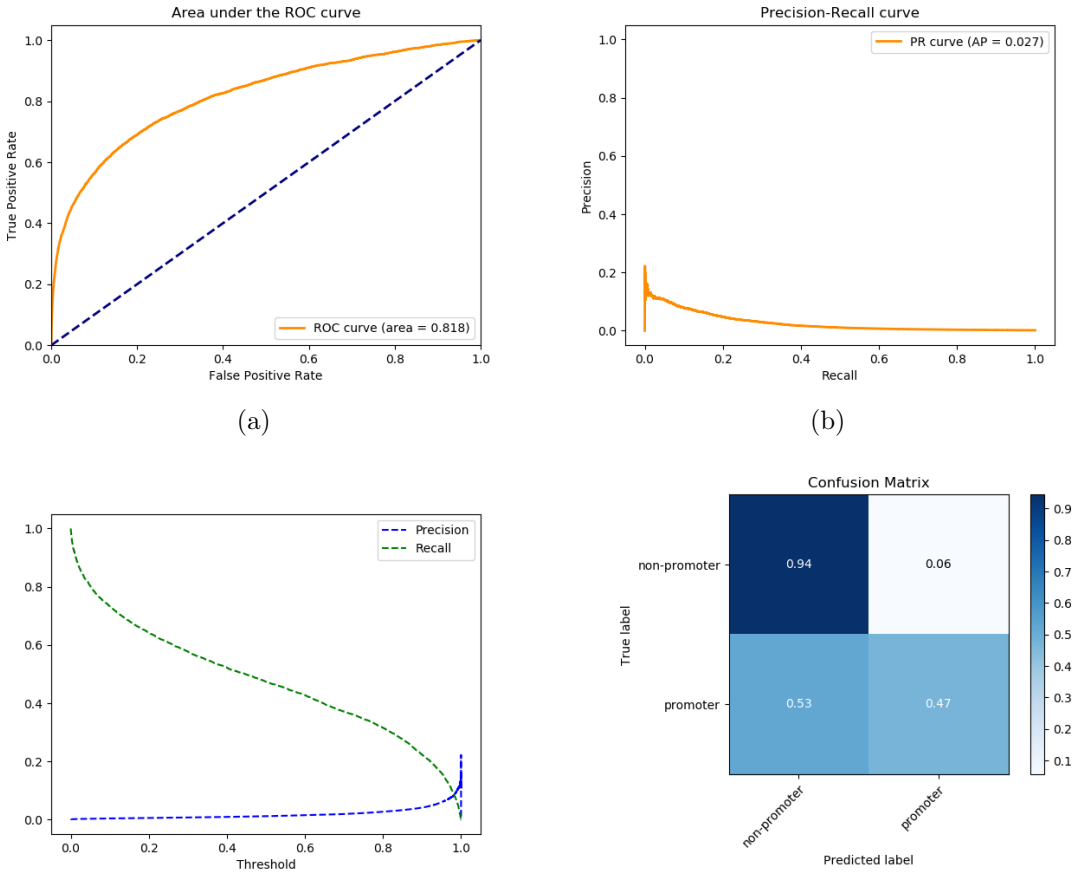


Figure B.3: Results from the reproduced ICNN model tested on **our** data: (a) shows the AUC; (b) shows the PR curve; (c) shows precision and recall by threshold; (d) shows the confusion matrix.

```
Metrics report for trained model (evaluation/icnn/ours.csv) .
Threshold set to 0.50
              precision    recall  f1-score   support

non-promoter      0.999     0.944     0.971   4602408
    promoter      0.012     0.474     0.023      6599

    accuracy                          0.943   4609007
   macro avg      0.506     0.709     0.497   4609007
weighted avg      0.998     0.943     0.969   4609007

Accuracy score: 0.943
Balanced Accuracy score: 0.709
F1 score: 0.023
Matthews Coefficient Correlation: 0.068
Precision score: 0.012
Recall score: 0.474
F-beta score: 0.023
Jaccard Similarity score: 0.012
```

Figure B.4: Report with results from our reproduced ICNN model tested on **our** data

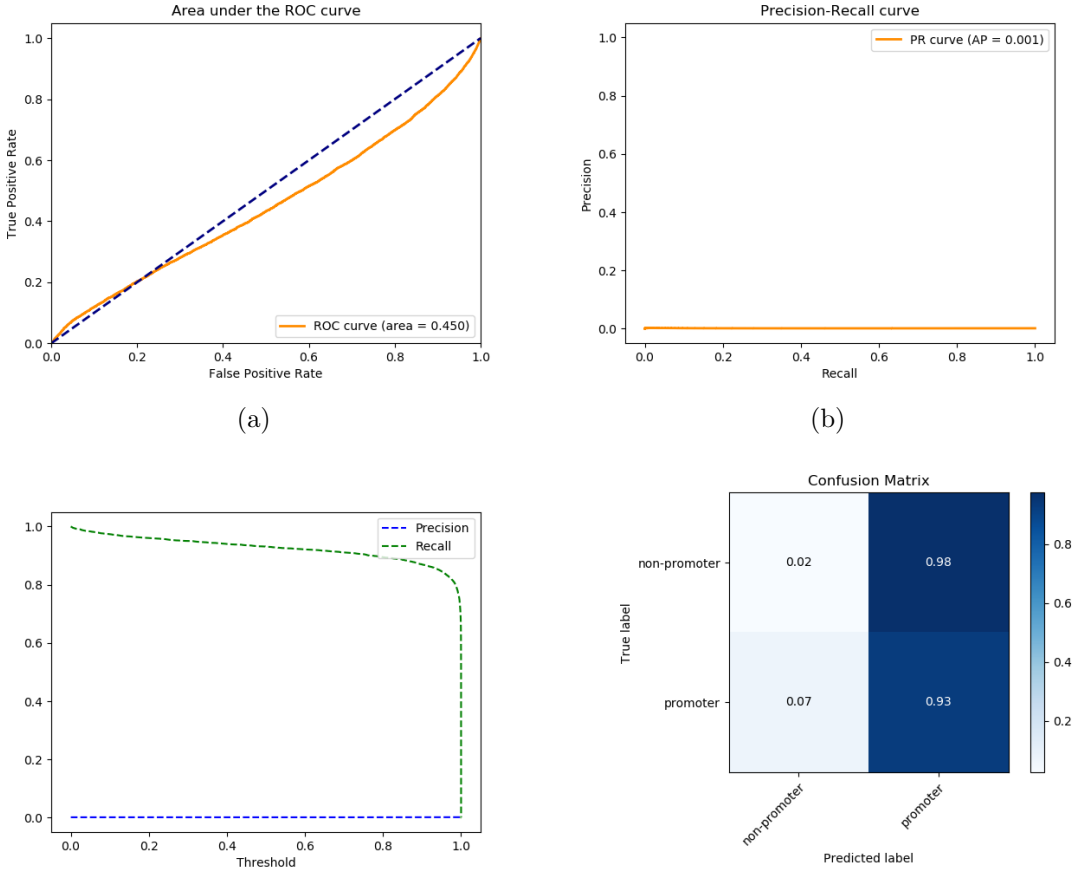


Figure B.5: Results from the reproduced DeePromoter model tested on **our** data: (a) shows the AUC; (b) shows the PR curve; (c) shows precision and recall by threshold; (d) shows the confusion matrix.

```
Metrics report for trained model (evaluation/deepromoter/ours.csv) .
Threshold set to 0.50
              precision    recall  f1-score   support

non-promoter      0.996     0.025     0.048   4602408
    promoter      0.001     0.931     0.003      6599

    accuracy                          0.026   4609007
   macro avg      0.499     0.478     0.025   4609007
weighted avg      0.995     0.026     0.048   4609007

Accuracy score: 0.026
Balanced Accuracy score: 0.478
F1 score: 0.003
Matthews Coefficient Correlation: -0.011
Precision score: 0.001
Recall score: 0.931
F-beta score: 0.003
Jaccard Similarity score: 0.001
```

Figure B.6: Report with results from our reproduced DeePromoter model tested on **our** data