FACULTY OF SCIENCES

FACULTY OF BIOSCIENCE ENGINEERING

FACULTY OF ENGINEERING AND ARCHITECTURE

DEEP LEARNING FOR CLASSIFICATION OF DNA FUNCTIONAL SEQUENCES

Word count: 29363

Griet De Clercq

Student number: 01103351

Promoters: Prof. Dr. Ir. Joni Dambre
Prof. Dr. Wesley De Neve
Prof. Dr. Willem Waegeman

Supervisor: Jasper Zuallaert

A dissertation submitted to Ghent University in partial fulfilment of the requirements for the degree of Master of Science in Bioinformatics: Bioscience Engineering

Academic Year: 2018 - 2019

GHENT UNIVERSITY

Preface

This master thesis was a long time coming. Three years to be exact, due to suffering a depression at the end of 2016, which caused me to temporally put my studies on hold. I managed to overcome my condition, and am immensely proud to finally present the finished result of these past three years.

I firstly would like to thank my promotors Prof. Dr. Wesley De Neve and Prof. Dr. Willem Waegeman for providing me with guidance during my thesis years. I especially want to thank my head promotor Prof. Dr. Ir. Joni Dambre, for giving me the opportunity to finish my chosen subject, her never-ending contribution of her expert knowledge, her patience, and for taking on the role of both promotor and supervisor. I would also like to thank Prof. Dr. Yvan Saeys for supplying the subject for this thesis, and Lionel Pigou and Jasper Zuallaert for answering every possible question I ever shot their way.

A round of applause is in order for my loving parents, who gave me the opportunity of pursuing an education at Ghent University, who believed in me year after year, and who allowed me the freedom I needed to find myself again. Another series of thanks go out to my friends and family; Melissa and Sofie for the consistent encouragement and the tummy-aching laugh sessions, Bram for always being his level-headed and calm self, Michiel and Francis for reaching out to me when times were difficult, and my brother and sister for repeatedly providing a loving and welcome home to visit whenever I wanted. Above all, I would like to thank my boyfriend Mathieu for tolerating all of my shenanigans throughout the years, and for being the most handsome rubber duck I could ever think of during my hours-long coding sessions.
A very special thanks goes out to my therapists Nele Arbyn and Liselot Vangronsvelt. Even while I can already hear them say 'in the end, you did all of this work yourself', I cannot express my gratitude enough for all the support they provided. You guys truly worked miracles.

And although science hasn't found a way to communicate with animals yet, I would still like to thank my cat Ellie, for being my ever-present fluffy companion during my deep learning endeavours, and my dog Mona, for always being her overly enthusiastic self when I needed hugs the most.

Griet De Clercq
Ghent, 26 August 2019

Table of Contents

List of Figures

List of Tables

List of Abbreviations

2D two-dimensional
3' three prime
3D three-dimensional
4D four-dimensional
5' five prime

A adenine

BAC bacterial artificial chromosome
BDGP Berkeley Drosophila Genome Project Reese (2001)
bp base pairs
BRE^{d} downstream B recognition element
BRE^{u} upstream B recognition element

C cytosine
CAGE cap analysis gene expression
CNN convolutional neural network
CV cross-validation

DCE downstream core element
DNA desoxyribonucleic acid
DNN deep neural network
DPE downstream promoter element

EP3 Easy Promoter Prediction Program Abeel *et al.* (2008)
EPD Eukaryotic Promoter Database Cavin Perier *et al.* (1998)

FDR false discovery rate
FN false negative
FP false positive
FPR false positive rate

G guanine
GAN generative adversarial network Goodfellow *et al.* (2014)
GPU graphics processing unit

HMM hidden Markov model

inr initiator element
IUPAC International Union of Pure and Applied Chemistry

kbp kilo base pairs

Mbp mega base pairs

MDT maximal dependency tree
MLP multilayer perceptron
mRNA messenger ribonucleic acid
MSE mean squared error
MTE motif ten element

NaN not-a-number
NCBI National Center for Biotechnology Information www.ncbi.nlm.nih.gov
NGS next-generation sequencing
NN neural network

ORF open reading frame

Pol I polymerase I
Pol II polymerase II
Pol III polymerase III
PPP promoter prediction program
PWM position weight matrix

RAM random-access memory
RAMPAGE RNA Annotation and Mapping of Promoters for the Analysis of Gene Expression
ReLu rectified linear unit
RMSE root mean squared error
RNA ribonucleic acid
RNN recurrent neural network
rRNA ribosomal ribonucleic acid

SD standard deviation
SVM support vector machine

T thymine
tanh hyperbolic tangent
TF transcription factor
TP true positive
tRNA transfer ribonucleic acid
TSS transcription start site

U uracil
UCSC University of California Santa Cruz https://genome.ucsc.edu/
UTR untranslated region

VIB Flemish Institute for Biotechnology

WAM weight array model
WMM weight matrix model

XCPE1 x gene core promoter element 1
XCPE2 x gene core promoter element 2

List of Symbols

- downstream of a position within the DNA
gap within a DNA or RNA sequence

+ upstream of a position within the DNA

B one of following nucleobases: CGTU

D one of following nucleobases: AGTU

H one of following nucleobases: ACTU

K one of following nucleobases: GTU

M one of following nucleobases: AC

N one of following nucleobases: ACGTU

R one of following nucleobases: AG

S one of following nucleobases: CG

V one of following nucleobases: ACG

W one of following nucleobases: ATU

Y one of following nucleobases: CTU

Abstract

English

Deep learning is currently one of the most popular machine learning methods due to its ability to autonomously extract features from enormous amounts of data and automatically learn meaningful representations from them. It is applied in various scientific fields, such as image and speech recognition, and robotics. Due to the recent explosion in biological 'omics' data, deep learning has also found its application within the biology field, with it being used in problems such as early cancer recognition and protein-protein interactions. In this dissertation, it is applied in the automatic annotation of splice sites and promoter sequences in *A. thaliana* and *H. sapiens*.

Splice site prediction is a mainly solved problem where prediction results as high as 95% are obtained for both precision and recall. However, state of the art deep learning models represent the data using a one-hot encoding, which has the limitation that it requires all samples to be of the same length if no padding or cropping is to be applied. A new data representation strategy called k-mer encoding is therefore proposed which can work on datasets consisting of samples with varying lengths. Another problem frequently encountered in splice site prediction is the large class imbalance in favour of the negative samples. This results in high false discovery rates as deep learning algorithms need large amounts of data in order to be able to generalise to unseen data. To circumvent this, a new augmentation method is proposed that takes the reverse complement of the positive samples. As opposed to splice site prediction, automated promoter annotation suffers from mediocre prediction results due to the lack of easily identifiable consensus sequences and a straight-forward negative sample construction method. A new negative sample construction scheme is introduced in this dissertation, which starts from the positive samples and which preserves the most conserved core promoter elements present in them.

The state of the art splice site prediction method by Zuallaert *et al.* (2018) is used as a starting point in order to develop and obtain a detailed understanding on how deep neural networks work. Their convolutional neural network called SpliceRover is rebuilt for use on an *A. thaliana* splice site dataset. This replica model failed to reproduce the results reported by Zuallaert *et al.*, stalling at 90% precision and recall values for the positive donor and acceptor classes. If the data is augmented using the reverse complement strategy, significant improvements are seen in the false discovery rate and precision values. This indicates that the proposed augmentation method can help in solving the problems encountered in splice site prediction. When k-mer encoding is applied, deep learning fails to capture the details within the data and does not produce meaningful results. The networks used for promoter annotation were able to produce state of the art prediction results with one-hot encoded data. If k-mer encoded data is used, a significant worse performance is encountered for values of $k \in [3, 4]$. A k-mer value of 5 can however compete with the results obtained with one-hot encoded *A. thaliana* data, although this is most likely due to the large differences in features between the negative and positive promoter class. The proposed negative construction scheme has a neutral effect towards the prediction results and does not yield significant improvements.

Futher works consists of altering the negative construction method in such a way that it contains negative samples from all sorts of negative sample construction approaches. When the negative samples are constructed using a (semi)-random construction scheme, the bases should be chosen in such a way that the species-specific DNA base composition is respected. This way, a more robust deep learning network can be developed that still achieves high prediction results when employed outside of test settings.

Nederlands

Deep learning is momenteel één van de populairste gebruikte methoden in het domein van machinaal leren. Dit komt doordat het in staat is volledig autonoom eigenschappen te herkennen in grote hoeveelheden data, en via deze automatisch te leren hoe de data voor te stellen en te voorspellen. Vandaag wordt het gebruikt in een waaier aan wetenschappelijke disciplines zoals bijvoorbeeld beeld- en spraakherkenning en robotica. Door de recente explosieve groei in het aantal 'omics' datasets heeft deep learning ondertussen ook zijn plaats gevonden binnen de biologie. Zo wordt het onder meer gebruikt om kanker op te sporen in een vroeg stadium en om eiwit-eiwit interacties te voorspellen. In deze masterproef wordt deep learning gebruikt voor de automatische herkenning van splice sites en promotoren in *A. thaliana* en *H. sapiens*.

Splice site predictie is een probleem dat reeds grotendeels is opgelost, en waar precisie en sensitiveit waarden van 95% makkelijk worden behaald. De state of the art deep learning modellen gebruiken echter een one-hot encodering om hun data voor te stellen, welke als nadeel heeft dat een dataset enkel voorbeelden mag bevatten die dezelfde lengte hebben. Dit kan worden omzeild met de k-mer encodering, welke geen problemen heeft met voorbeelden van variable lengte zonder deze te moeten aanpassen. Een ander frequent voorkomend probleem is het lage aantal positieve voorbeelden aanwezig in datasets, wat aanleiding geeft tot een verhoogde false discovery rate. Dit komt doordat deep learning algoritmen nood hebben aan een groot aantal voorbeelden uit de verschillende klassen om optimaal te kunnen leren. Het kan worden omzeild door de data te augmenteren. Hiervoor wordt een nieuwe data augmentatie strategie geïntroduceerd die het omgekeerde complement neemt van elk positieve splice site DNA sequentie. In tegenstelling tot splice site predictie leidt het gebruik van machinaal leren in automatische promoter annotatie nog steeds tot tegenvallende resultaten. Dit komt omdat er geen duidelijke consensus sequenties aanwezig zijn in promotoren, en omdat de constructie van een negatieve dataset niet voor de hand ligt. In deze masterproef wordt daarom een nieuwe manier voorgesteld om de negatieve voorbeelden te construeren. Hierbij worden deze opgesteld vanuit de positieve voorbeelden, en worden de meest geconserveerde core promotor sequenties behouden.

De state of the art splice site detectie methode van Zuallaert *et al.* (2018) wordt gebruikt als startpunt om deep learning onder de knie te krijgen. Hun convolutioneel neuraal network SpliceRover werd nagemaakt voor gebruik en haalde een maxmimum waarde van 90% voor precision en recall van de positive splice site klassen. Het slaagt er dus niet in om de resultaten van Zuallaert *et al.* te reproduceren. Bij toepassing van de k-mer encodering werd duidelijk dat deze de onderliggende splice site patronen niet kan weergeven. Wanneer de aangereikte data augmentatie strategie wordt gebruikt zijn er significante verbeteringen te merken in de false discovery rate en precisie waarden. Dit geeft aan dat de omgekeerde complement aanpak een geldige methode is die ook in de praktijk kan gebruikt worden om de resultaten van automatische splice site detectie te verbeteren. De netwerken ontworpen voor promoter predictie met one-hot encodering bereikten resultaten compatibel met de huidige state of the art. Wanneer een k-mer encodering werd gebruikt met k gelijk aan 3 of 4, daalden deze resultaten significant. Een k-mer encodering met k gelijk aan 5 kan echter wél concurreren met de resultaten behaald met one-hot geëncodeerde *A. thaliana* data. Dit is hoogstwaarschijnlijk te wijten aan de grote verschillen tussen de negatieve en positieve samples in de datasets. De nieuw geïntroducteerde methode om negatieve samples aan te maken levert geen significante verbetering op.

Toekomstig werk bestaat eruit om de negatieve voorbeelden constructie methode zo aan te passen dat voorbeelden gemaakt worden via allerhande verschillende technieken. Wanneer hierin een techniek gebruikt die willekeurig DNA basen kiest, dienen ze gekozen te worden zodat de specifieke base compositie eigen aan de species wordt behouden. Hierdoor kan een deep learning netwerk worden ontwikkeld dat ook hoge predictie resultaten behaald in meer realistische omstandigheden.

1. Introduction

Every organism on this planet is able to live and function due to a minuscule molecule found within all of its cells called DNA. Ever since the resolution of its structure in 1953 by Watson and Crick, major breakthroughs have been made in the field, eventually leading to our recently acquired ability to change an individual's DNA at will (Cong *et al*., 2013; NPR, 2019). This genome editing would be impossible without the invention of genome sequencing by Sanger *et al.* (1977), which allowed scientists to read DNA as one would read a book. While Sanger sequencing was the sequencing standard for many years to come, the technique was slow, expensive and labour-intensive (Heather and Chain, 2016). An answer to these problems was given in the form of next-generation and third-generation sequencing methods such as Illumina (Canard and Sarfati, 1994) and nanopore sequencing (Jain *et al.*, 2016). These new methods allowed for automated sequencing that is faster and cheaper than Sanger sequencing with an added surplus of being able to read out a high amount of DNA sequences at once, and led to an explosion in the amount of DNA data that is freely available in genetic databases.
To make sense of and bring structure into this enormous amount of data, several universal structural blocks are determined according to their fulfilled function, their position within the genome and their exact sequence, among other criteria (Stein, 2001). The assignment of a certain DNA section to such a universal structural block or genomic element is called structural annotation. Identifying what role each of the different elements plays within an organism is referred to as functional annotation. Lacking the computational resources for parsing the tremendous amount of data accompanied with sequencing a species' DNA, annotation was mostly done by hand in the years following the first sequenced genomes. As this requires trained professionals and is labour-intensive and time consuming, automatic detection of genomic elements rapidly sparked an extensive interest in the scientific community. The earliest developed algorithms focused on recognising smaller genetic regions such as ribosome binding sites, promoters and splice sites (Staden, 1984), which were later combined to search for genes in whole genomes (Wang *et al.*, 2004) and even functionally annotate them (Ferrão *et al.*, 2019). Research on automatic structural annotation of genomes is still ongoing, and especially functional annotation is still in its infancy.

Of special importance in this master dissertation is the structural annotation of exon-intron junctions and promoters. Exons and introns are the main building blocks of genes and the transitions between them - referred to as splice sites - are important for a successful gene expression chain reaction. They have an influence on the final gene product and mutations in splice patterns can lead to a variety of pathologies (Ars *et al.*, 2000; Daguenet *et al.*, 2015). Promoters are found in the vicinity of the exons and introns, and directly drive the gene expression process. They consist of two smaller blocks, called the core promoter and the proximal promoter. The core promoter is needed to start the expression of a gene, while the proximal promoter helps in further facilitating the process. Recently it has been shown that mutations in promoter regions can lead to cancer and other diseases (Jang *et al.*, 2018; Fredriksson *et al.*, 2014), making them compelling targets for potential new therapies to alleviate or even eradicate these diseases. The annotation of both splice sites and promoters can also facilitate the identification of their accompanying genes, and aid in a better understanding of the molecular processes happening in cells. The first algorithms designed to this end utilised position weight matrices and homology searches for splice site and promoter prediction, respectively (Staden, 1984; Mulligan and Mcclure, 1986). Other popular techniques in the following years were support vector machines (SVMs), decision trees and neural networks (NNs). NNs autonomically extract features from the data and learn meaningful representations from them, resulting in algorithms that can efficiently model complex biological problems. However, to do this, they need a substantial amount of data and computational resources, both of which were not available at the time. These problems were solved with the arrival of the DNA sequencing methods and with the publication of the deep neural network (DNN) AlexNet in 2012 by Krizhevsky *et al.*. In their paper, Krizhevsky *et al.* proposed a new way of training DNNs by utilising graphics processing units (GPUs). GPUs can process large quantities of data in parallel, making them ideal for use with deep learn-

ing approaches. The technique by Krizhevsky *et al.* was quickly picked up by the scientific community, and led to the use of deep learning to address many known hard problems. Today, a variety of state of the art splice site and promoter prediction programs exist that utilise deep learning in their underlying algorithm, such as SpliceRover (Zuallaert *et al.*, 2018) and CNNProm (Umarov and Solovyev, 2017).
The species of interest in this dissertation for splice site and promoter prediction are *Arabidopsis thaliana* and *Homo sapiens*. *A. thaliana*, commonly referred to as thale cress, is an annual flowering plant from the *Brassicaceae* family and grows 20 to 25 cm tall (EnsemblPlants, 2019). It is a multicellular, diploid eukaryotic organism consisting of five chromosomes, which hold the entire genome length of 135 mega base pairs (Mbp). Due to its small physical and genetic size and its rapid life cycle, it is a widely researched plant that is frequently used as a model organism in genetic studies. *H. sapiens* is the only species of the human genus currently living on earth. It is a diploid organism with a genome size containing nearly 3300 Mbp, divided into 22 pairs of autosomal chromosomes and 1 pair of sex chromosomes (NCBI, 2019). This genome contains around 50000 genes, almost 20000 of which are coding genes. This number is frequently revised due to the improvement of genome annotation methods, indicating that there is still a lot left to explore about our own species.

This dissertation is written to provide sufficient background knowledge for both computer scientists who lack a detailed biology education and for biologists who never practised deep learning before. Therefore an extensive summary is given on DNA and the genome in chapter 2, and a high-level overview of NNs and deep learning in chapter 3. A summary of the history of splice site and promoter prediction is given in chapter 4, along with their current state of the art. It is followed by the 'Materials and methods' section in chapter 5, in which the techniques used to tackle the problems in this thesis are presented. Finally, results are shown and discussed in chapter 6, with a final remark and conclusion, and future perspectives in chapter 7.

1.1. Problem statement

As splicing makes up an important step in the gene expression process, the required sites are highly conserved across species. They stretch over only a few base pairs (bp) , thus making it possible to set a fixed length of nucleotides around their consensus sequence that still captures the whole splice site. This fixed length makes it easy to apply frequently used data encodings to the DNA sequences, and incorporate them into already developed algorithms. Because of their highly conserved consensus sequence, algorithms identify the underlying genome patterns that are needed to constitute a splice site with little effort, resulting in methods that can point out the sites in a given sequence in a highly reliable manner. Because of this, splice site detection is mainly a solved problem for most organisms. It however suffers from a large imbalance in the datasets, where the positive samples are heavily outnumbered by the negative ones. This can result in a high number of false positive results, and thus an inflated false discovery rate.

Promoter regions directly drive the initiation of a gene into functional molecules, such as proteins or ribonucleic acids (RNAs). They consist of a proximal promoter, which mostly lacks easily identifiable structures, and a core promoter, which binds the polymerase protein needed to start the gene expression process. Due to the importance of the core promoter in gene expression, it consists of several structural regions which are highly conserved. However, their presence does not lead to easy promoter identification. Three different polymerase molecules exist, leading to three structurally different types of promoters. Within each of these subtypes, still a great variety is found in the structure of their core promoter as not all existing core building blocks are needed for the promoter to function. This leads to various combinations of conserved regions which constitute a valid core promoter. A model designed to recognise promoter sequences will therefore have to learn a great range of different features in order to make correct assumptions about a sequence being a promoter or not.
Another problem is that promoters most commonly stretch over several hundreds of bp, which com-

plicates the choice of a fixed sample length that still captures all present consensus sequences. This is circumvented by setting a large fixed length to the annotated promoter regions. However, this can cut short promoters that are larger than this length, resulting in a loss of nucleotides that can possibly help with correct promoter identification, or surround shorter promoter sequences with non-contributing DNA bases that could introduce meaningless features.
The last issue lies within the construction of negative promoter datasets. While the design of negative splice sites is very straightforward due to their short and highly conserved consensus sequences, this is not the case for promoter data where no single consensus sequence can be assigned that is present within all promoter structures. The negative sample dataset is therefore typically composed out of randomly selected coding sequences of the same species (Liu *et al.*, 2018), promoter sequences from another species (Towell, 1993), or deconstructed positive samples (Oubounyt *et al.*, 2019). These construction methods provide no great challenge for the designed promoter prediction programs (PPPs), as the positive and negative sample datasets have little features in common. State of the art algorithms work well on these artificially constructed datasets, yet fail when applied onto more realistic settings (Bajic *et al.*, 2004).

1.2. Aims

The aims of this master dissertation are twofold. The first goal is to obtain a detailed understanding of how NNs in general, and deep learning in particular, work. To this end, an *A. thaliana* splice site dataset was obtained through the Flemish Institute for Biotechnology (VIB). This dataset has previously been used in other studies conducted by Degroeve *et al.* (2005) and Zuallaert *et al.* (2018), who respectively used an SVM and a DNN to predict the splice sites in the *A. thaliana* dataset. The deep learning method by Zuallaert *et al.*, named SpliceRover, is reproduced. Tweaks are made to the SpliceRover algorithm's architecture in order to observe how different hyperparameters affect the acquired results. A new data augmentation method is proposed where the reverse complement is taken, and this is order to lessen the large class imbalance encountered in the *A. thaliana* splice set. A different train-test-validation split than the more commonly used random and stratified split is also proposed, where the data are split based on the gene the splice sites belong to.

The second goal focuses on tackling the problems that come along with promoter prediction. The first problem encountered is the varying length of promoters which has to be captured in a fixed sample length. The proposed solution to enable variable sample lengths is k-mer encoding, which can take samples of differing length and transform them into vectors with the same fixed length. This is first tested on the splice site data, as results obtained from the new encoding can easily be evaluated according to the already obtained state of the art results by Degroeve *et al.* and Zuallaert *et al.*. The newly proposed encoding is then applied to the promoter datasets of *A. thaliana* and *H. sapiens*, and compared to the more frequently used one-hot encoding to identify k-mer encoded data can be a valid alternative to other more commonly used data encodings. The second issue is found within the construction of the negative samples in the promoter datasets. Two different construction methods are explored in this dissertation, one of which is a newly proposed one. They are compared to the state of the art construction approach by Oubounyt *et al.* (2019) to assess whether their use can result in more realistic testing settings for newly developed PPPs.
Both problems in promoter prediction are addressed through the use of deep learning. Recently, it proved to be a worthy contestant to more conventional machine learning techniques presently used in state of the art PPPs. The aim is to provide another significant boost to the presented potential solutions for the problems commonly manifested in promoter prediction, and to improve the overall performance in a field that is still troubled by mediocre prediction results.

2. Biology of the genome

2.1. DNA and its structure

DNA can be regarded as the source code of all life on earth, containing all the information needed to create and maintain it. Its main building blocks are molecules called nucleotides. They are composed of a nucleobase attached to a sugar called desoxyribose, which in its turn is attached to a phosphate group (see figure 2.1a). There are four bases that can be used in a nucleotide, namely adenine (A), guanine (G), cytosine (C), or thymine (T). These can be regarded as the four letter alphabet with which the DNA source code is written. When it is not known or further specified which exact base is found at a certain place in a DNA sequence, a combination of several bases can be used. Each combination has a unique name and accompanying letter symbol in accordance with the International Union of Pure and Applied Chemistry (IUPAC) nucleotide notation (see table 2.1) (Saenger, 1984; Alberts *et al*., 2014; NIH, 2017).

Several nucleotides are connected to each other by linkage of the sugar group of the first nucleotide to the phosphate group of the next, thus forming a polynucleotide strand with the backbone consisting of alternating sugar and phosphate groups (Saenger, 1984; Ghosh and Bansal, 2003; Alberts *et al.*, 2014). As the phosphate group is attached to the fifth carbon atom of the sugar, this is called the five prime (5') end, while the ending with the exposed sugar group is called the three prime (3') end. Two polynucleotide strands are coiled around each other and joined by hydrogen bonds, creating the DNA double helix structure. The bases are directed towards the center of the helix, and bind to the bases of the opposite strand, following the complementary base pairing rules that A can only bind to T, and G only to C. It is between these specific base pairs that the hydrogen bonds occur. The structure of a double-stranded DNA helix and its composition can be seen in figure 2.1b.
Due to the base pairing, the base composition of one polynucleotide strand should always be complementary to the other strand (Alberts *et al.*, 2014). As each strand should hold the exact same information and thus should consist of the same base sequence, the two strands will have to run in opposite direction to each other in order to still be in line with the base pairing rules. One strand is considered the non-template, sense, or coding strand, and runs from 5' to 3'. The other is considered the template, antisense or non-coding strand, runs from 3' to 5', and is in its 5' to 3' direction the reverse complement of the coding strand (IUPAC-IUB, 1990; Pray, 2008). Positions relative to a certain place or element can be indicated by use of the terms upstream or downstream and their respective symbols - and +, where upstream is towards the 5' end and downstream towards the 3' ending (Lodish, 2008).

In this master dissertation, all sequences are given in 5' to 3' direction unless stated otherwise. Only the coding strand of each DNA sequence is given as the non-coding strand can be derived from it by taking the reverse complement.

2.2. The reference genome

The sequence of the consecutive bases encoding the information that is stored within an organism's DNA is called the genome (Lodish, 2008; Alberts *et al*., 2014; NIH, 2017). It can be visualised as a string of characters using the abbreviations of the four DNA bases, and can range from hundred thousand to 150 billion bp (Ball, 2006; Pellicer *et al.*, 2010). Inside the cell, the genome is neatly stored in compact structures called chromosomes. Not all its contained information codes for life functions or an organism's traits. Tthe majority is even considered to be non-coding (Ponting and Hardison, 2011; Kellis *et al.*, 2014). Coding means that the information held by that certain block of DNA can be expressed as a protein (Twyman, 2003). Although non-coding DNA does not code for proteins, it can still fulfil important functions such as transfer and regulation of the coding DNA, its derived products, or their needed

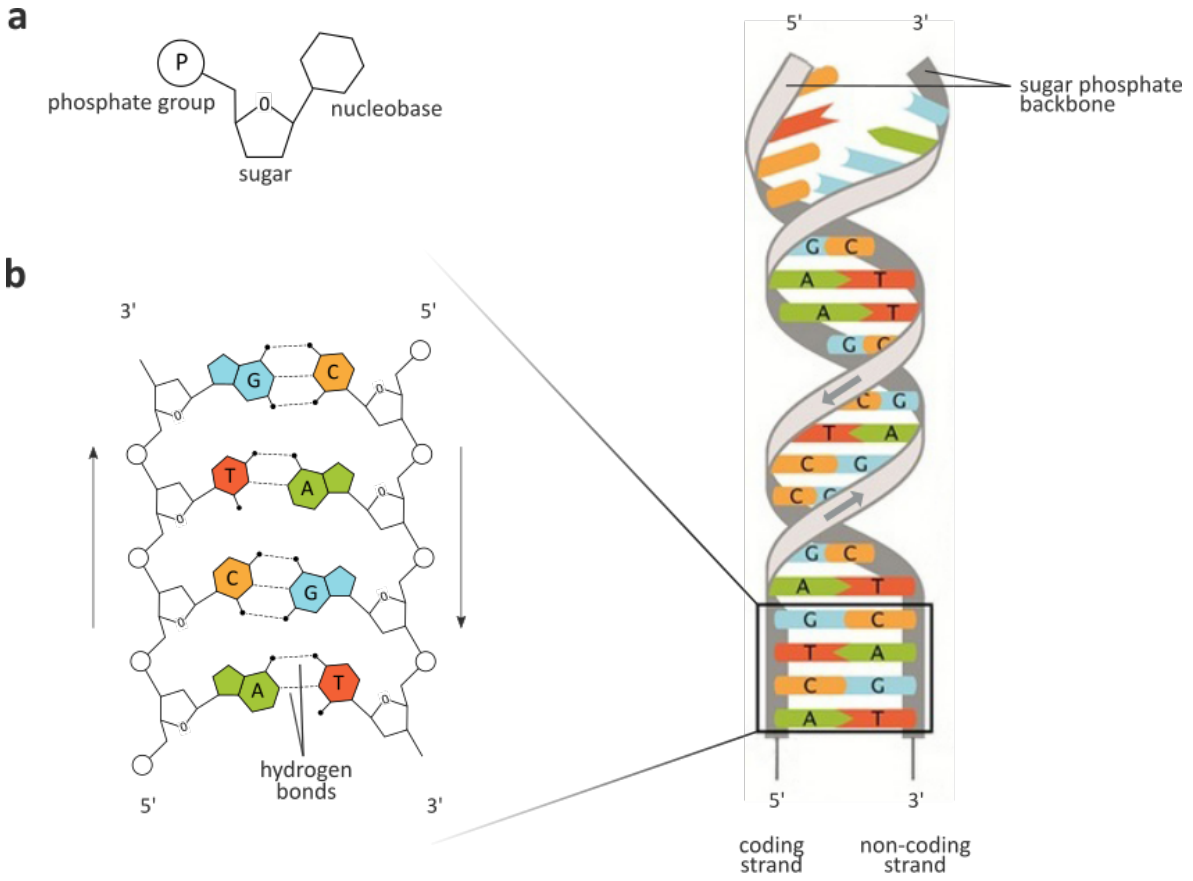


Figure 2.1: **(a)** Simplified structure of a nucleotide with an unspecified base. **(b)** On the right, the three-dimensional (3D) view of a DNA helix is shown. The last four bases are magnified on the left, showing the hydrogen bonds and the sugar phosphate backbones in more detail (adapted from Pray (2008) and Ratcliffe (2015)).

Table 2.1: IUPAC nucleotide notation. The symbols (first column) stand for one or more bases, which are specified in the third column. The full description of the symbol can be found in the second column. The last column indicates the symbol for the complementary bases of the different groups according to the base pairing rules (NC-IUB, 1984).

Symbol	Description	Represented bases	Complement
A	Adenine	A	T or U
C	Cytosine	C	G
G	Guanine	G	C
T	Thymine	T	A
U	Uracil	U	A
R	Purine	AG	Y
Y	Pyrimidine	CT	R
S	Strong interaction	CG	S
W	Weak interaction	AT	W
M	Amino	AC	K
K	Keto	GT	M
B	Not adenine	CGT	V
D	Not cytosine	AGT	H
H	Not guanine	ACT	D
V	Not thymine or uracil	ACG	B
N	Any base	ACGT	N
-	Gap	None	None

building components (Ambros, 2004; de Farias *et al.*, 2014).

Sometimes sequences can be found within a genome that are similar or identical in different organisms across several species. These are conserved sequences, and sequences which are similar in nearly all species are referred to as invariant or highly conserved (NCBI, 1993). The block of most commonly found nucleotides at a specific DNA location is referred to as a consensus sequence (Pierce, 2012), and the IUPAC nucleotide notation (table 2.1) is used to visualise them. If an IUPAC nucleotide symbol is used which contains two or more bases in its group, this means that each represented nucleotide is found with an equal frequency in that exact place.
Different organisms within one species can also have various forms of the same gene, referred to as alleles (Nature Education, 2014; Alberts *et al.*, 2014). An example of this is a gene that gives colour to a plant's flowers. One plant can have yellow flowers, and another plant of the same species can have red ones. The exact same gene codes for colour in both plants, but due to an alternative form in the second plant, the colour of the flowers is changed. This poses a problem for the structural annotation of a genome (NHS, 2017). If an organism's genome is annotated by looking at its DNA sequence, this annotation cannot be easily extrapolated to another organism of the same species as its DNA sequence can differ. This also implies that one cannot speak of *the* genome of a species. In order to still have a representative DNA sequence, scientists use so-called reference genomes to capture the diversity across several members of the same species. Such a reference genome is built by reading the genome of different organisms of a species and mixing them together to obtain one global sequence. In the past, these assemblies have been made for a number of model organisms, and can be freely downloaded from websites such as the National Center for Biotechnology Information (NCBI, www.ncbi.nlm.nih.gov) and the University of California Santa Cruz (UCSC, https://genome.ucsc.edu/).

2.3. Gene expression and RNA

The coding blocks of a genome are all localised in a bigger structural block called a gene, which is a DNA sequence that encodes for a protein or another functional molecule (Conner and Hartl, 2004). In the case of eukaryotes, genes start with non-coding regulatory sequences, followed by an open reading frame (ORF) consisting of coding and non-coding blocks respectively referred to as exons and introns, and end with another sequence of non-coding regulatory regions (Anderson *et al.*, 1981; Lynch, 2006; Shafee *et al.*, 2017). The conversion of the information held within a gene's DNA into a functional molecule is called gene expression (Pierce, 2012; Alberts *et al.*, 2014). In eukaryotes it consists of a transcription, modification, and translation step where an intermediate product called RNA plays the role of information carrier. RNA has a structure similar to DNA, but the sugar used in its nucleotides is ribose, and the base T is replaced by uracil (U). RNA is hardly ever found as a twisted double-strand helix, and more often as a single-stranded molecule folded back upon itself (GSLC, 2016). An overview of the eukaryotic gene expression process is depicted in figure 2.2.

The eukaryotic gene expression process for assembling proteins starts with transcribing the gene's exons and introns. Basal transcription factors (TFs) initiate the transcription by binding to a preceding regulatory sequence named the promoter. The basal TFs control the process's rate and recruit an enzyme called RNA polymerase II (Pol II) which unwinds the double-stranded DNA helix. The RNA Pol II then moves in 3' to 5' direction over the now single-stranded template strand and makes a reverse complementary pre messenger ribonucleic acid (mRNA) molecule. The place where the RNA polymerase will transcribe its first base is called the transcription start site (TSS). When the enzyme reaches a terminator sequence on the template strand, the newly synthesised pre-mRNA strand is released from the RNA polymerase (Clancy, 2008a).

The pre-mRNA molecule has to undergo several modifications, of which only splicing is of further interest for this dissertation. During splicing, a special protein structure called a spliceosome cuts the introns out of the pre-mRNA strand by cleavage at the exon-intron junctions. The coding regions are then joined

together, yielding a mature mRNA strand. It is also possible that exons are cut out of the pre-mRNA sequence, resulting in a different mature mRNA. By this mixing and matching of exons, it is possible for one gene to code for several different proteins. Splicing is necessary as introns cannot be expressed as a protein or another functional molecule (Clancy, 2008b; Alberts *et al.*, 2014).

The third and final step during the expression, called translation, is the conversion of mRNA into a protein. The mRNA is read three nucleotides at a time, and the nucleotide triplets are referred to as codons. Using the RNA alphabet, 64 such codons can be formed, where each nucleotide triplet encodes for a specific amino acid or for a special start or stop signal. A molecule called a ribosome attaches itself to the mRNA start codon carrying the translation start signal. The ribosome reads the signal and with the help of a transfer ribonucleic acid (tRNA) molecule, the corresponding amino acid is collected. Then the ribosome slides over the remaining mRNA-strand reading all the codons, and attaches the associated amino acids provided by the tRNA to its growing protein chain. The translation ends when the ribosome reaches a stop codon carrying the stop signal, and the translated protein is released from the ribosome (Clancy and Brown, 2008).

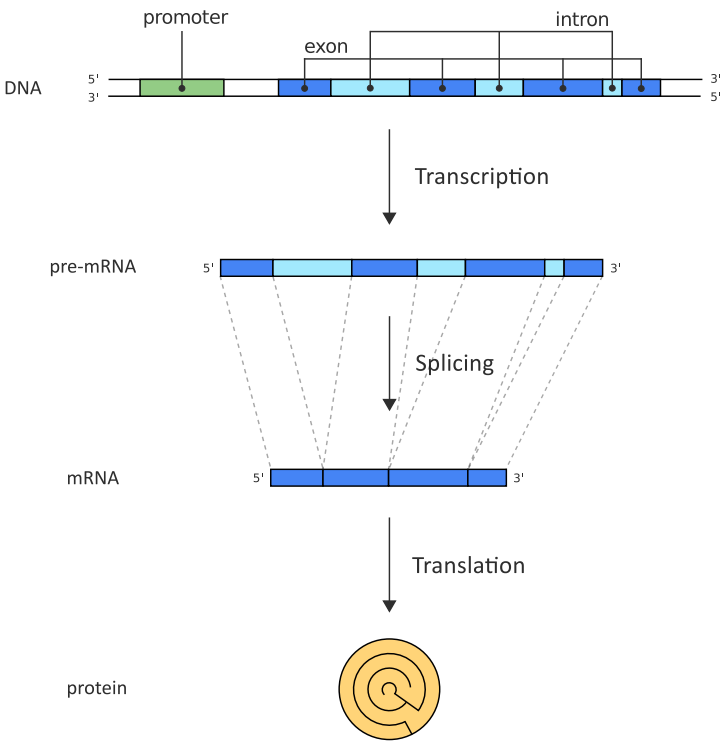


Figure 2.2: Schematic overview of the eukaryotic protein gene expression process. For simplicity, only the promoter region and the exons and introns of the gene are shown.

Apart from protein coding genes, additional genes exist that code for other functional molecules such as ribosomal ribonucleic acid (rRNA) and tRNA. These are respectively transcribed by RNA polymerase I (Pol I) and RNA polymerase III (Pol III) and bind to their own associated promoters. Their gene expression process elapses in a way similar to that of protein coding genes (Paule and White, 2000; Carter and Drouin, 2009).

2.4. Genes and their genomic elements

Several genes lie next to each other in the genome, divided by non-coding DNA blocks. Genes consist of three big structures, namely the ORF and the regulatory sequences before and after this ORF. These three structures consist of several smaller elements themselves, such as exons and introns, promoters, enhancers, and untranslated regions (UTRs). Even within these smaller blocks, other structures can be assigned as well. The overall structure of a eukaryotic gene and its position within the genome is visualised in figure 2.3a.

2.4.1. Splice sites

The ORF is the only part of a gene that carries the coding blocks which can be translated into a functional molecule (Alberts *et al.*, 2014). It starts with an exon, followed by alternating sequences of respectively an intron and an exon. During splicing, introns are cleaved out of the transcribed RNA at the splice sites,

as introns cannot be translated into a protein. Each exon-intron and intron-exon junction respectively falls within a donor and acceptor splice site, which are sometimes also referred to as the 5' and 3' splice sites (Clancy, 2008b). Apart from these two sites, a branch site has to be present as well for splicing to take place, which is a region found directly upstream of the acceptor splice site

Donor splice sites of the most commonly found U2 class introns can be recognised by the presence of a GT consensus sequence at the 5' end (Breathnach *et al.*, 1978; Breathnach and Chambon, 1981). This exact pattern is found in nearly all organisms whose DNA requires splicing. It is followed by a less invariant, but still mostly conserved region with RAGT as consensus sequence (Harris and Senapathyl, 1990; Patel and Steitz, 2003). At the 3' ending of the preceding exon, a MAG consensus sequence is found (Lodish, 2008).
The 3' end of an U2 intron is terminated by an acceptor splice site with NCAG as consensus sequence (Breathnach *et al.*, 1978; Breathnach and Chambon, 1981; Patel and Steitz, 2003), which in turn is directly preceded by a 15 to 20 bp long region, rich in Y nucleotides, called the polypyrimidine tract (Lodish, 2008). The 5' end of the succeeding exon is characterised by a G as most commonly encountered consensus nucleotide (Alberts *et al.*, 2014). Between 20 to 50 nucleotides upstream of the acceptor site lies the branch site, with an invariant A in its otherwise loosely conserved CTVACT consensus sequence (Reed and Maniatis, 1985; Patel and Steitz, 2003; Lodish, 2008).

Other consensus sequences in splicing exist, such as the U12-type introns. These introns are recognised by the consensus sequences of RTATCCTTT and CCTTAAC at respectively the donor and branch sites. The acceptor site has a sequence similar to that of the U2 introns (Lodish, 2008; Turunen *et al.*, 2013). A visualisation of the splice sites of an U2 intron and its surrounding exons can be found in figure 2.3b.

2.4.2. Promoter and enhancer regions

In the non-coding regulatory sequences that precede and succeed the ORF, enhancers and promoters can be found. Both are zones within the DNA that control the transcription process (Clancy, 2008a).

A promoter is a region upstream of a gene's ORF that is needed to initiate transcription (Smale and Kadonaga, 2003; Clancy, 2008a). Each eukaryotic gene has at least one promoter, with some genes being regulated by multiple promoters to enable tissue-specific gene regulation (Kim *et al.*, 2005; Adams *et al.*, 2011). It consists of another two major elements, called the proximal promoter and the core promoter. The core promoter is the sequence within 50 to 100 bp around the TSS (Roeder, 1996). It holds the sequences where the basal TFs can bind in order to recruit the RNA polymerase, and is thus the minimal sequence that is needed to initiate DNA transcription (Butler and Kadonaga, 2002). Because of its importance in the transcription process, most of its elements are conversed across species. The second element is the proximal promoter, which is found immediately upstream of the core promoter and up to 250 bp upstream of the TSS (Lodish *et al.*, 2000). It encodes binding sites for specifically needed TFs such as activators and repressors. As these binding sites are not needed to initiate transcription but only help in further regulation of the gene expression, their sequences differ greatly across genes.
As three different promoter types can be distinguished according to which RNA polymerase they bind, different structures can be distinguished as well. The most thoroughly researched one is the eukaryotic RNA Pol II promoter with one TSS. Several TF binding sequences have been identified and annotated within its core promoter. These elements are listed in table 2.2 along with their location and consensus sequence, and depicted in figure 2.3c. Not all these elements are needed in order for the core promoter to function, and typically only several of these building blocks will be present at once. Because of all these possible combinations, a variety of core promoter structures exist (Breathnach and Chambon, 1981; Kadonaga, 2002). Within RNA Pol II promoters with multiple TSSs, CpG islands and ATG-deserts are commonly found (Lee *et al.*, 2005; Akan and Deloukas, 2008; Lenhard *et al.*, 2012). CpG islands are regions rich in CG dinucleotides and are frequently methylated at the C residues. ATG-deserts are 1 kbp sequences in both directions around the TSSs where the frequency of the ATG trinucleotides is lower

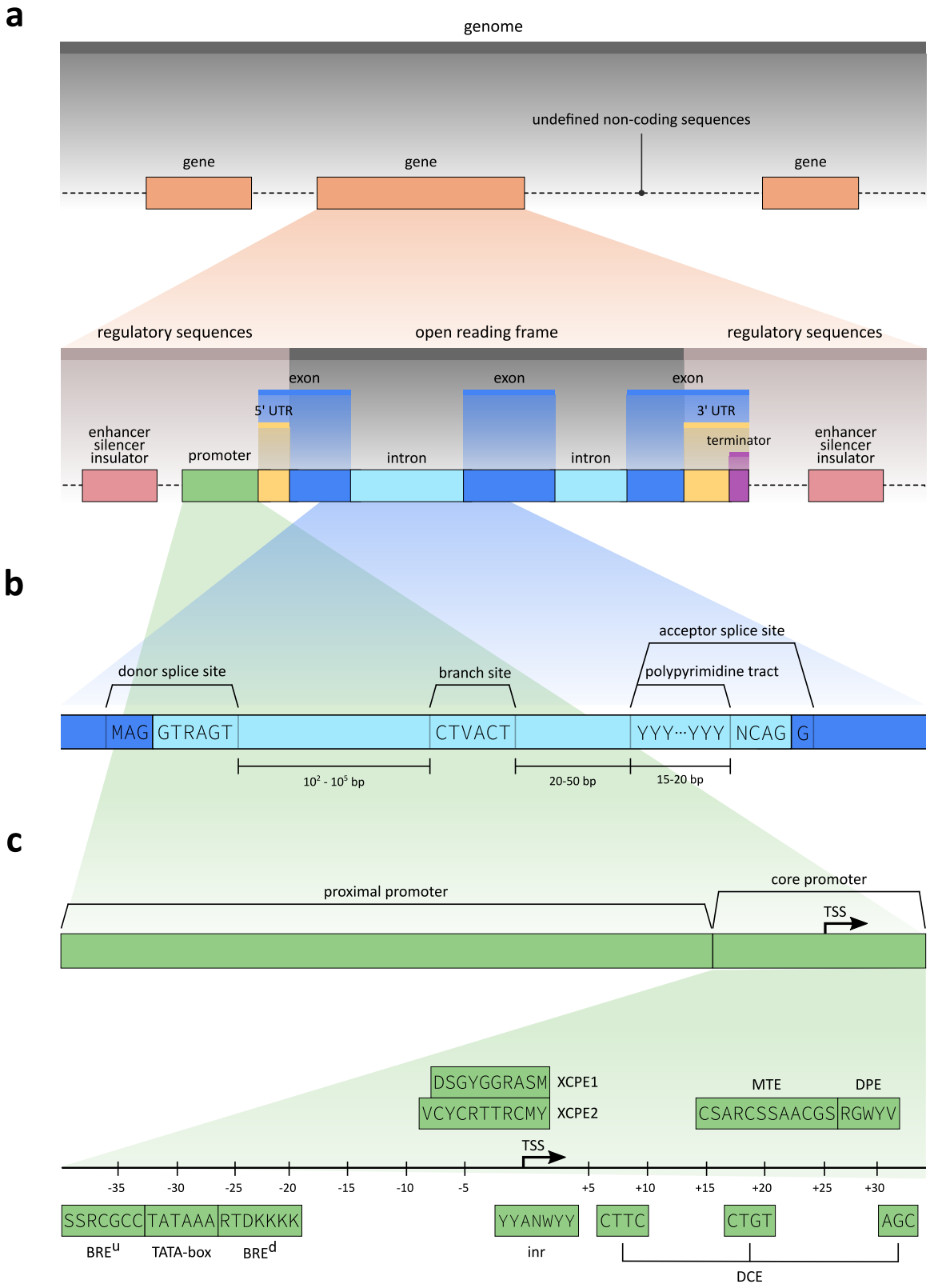


Figure 2.3: Structural blocks within DNA. **(a)** Positions of eukaryotic genes within the genome. One gene is enlarged to show its inner structure in more detail. **(b)** U2-type intron with its surrounding exons, showing the three necessary splice sites with their consensus sequences. **(c)** Structure of a eukaryotic RNA Pol II promoter with one TSS. The core promoter is enlarged and its most commonly found elements with their consensus sequences are shown. This promoter will never be encountered in real life as only a subset of these elements are needed to initiate transcription. It is therefore merely used as an illustration tool to visualise the approximate location of the individual elements.

Table 2.2: Most frequently found core promoter elements with their consensus sequence. Unless stated otherwise, the start position is approximately and relative to the TSS, and the consensus sequence is highly conserved across nearly all species (Breathnach and Chambon, 1981; Brenner *et al.*, 2002; Kadonaga, 2002; Lim *et al.*, 2004; Deng and Roberts, 2006; Juven-Gershon and Kadonaga, 2010; Roy and Singer, 2015).

Name	Location (bp)	Consensus sequence
upstream B recognition element (BRE^u)	-35	SSRCGCC
TATA-box	-30	TATAWAAR
downstream B recognition element (BRE^d)	-25	RTDKKKK
x gene core promoter element 1 (XCPE1)	-4	DSGYGGRASM
x gene core promoter element 2 (XCPE2)	-4	VCYCRTTRCMY
initiator element (inr)	+1	YYANWYY (mammals)
downstream core element (DCE)	+9, +18, +32	CTTC,CTGT,AGC
motif ten element (MTE)	+20	CSARCSSAACGS
downstream promoter element (DPE)	+30	RGWYV

than in the surrounding sequences. RNA Pol I promoters are typically rich in AT nucleotides around its initiator element (inr), and contain an upstream promoter element (Paule and White, 2000). The class of RNA Pol III promoters contains another three different types on its own, with box A, box B and box C found in the first two types, and a TATA-box in the third type (Paule and White, 2000).
An overall structure of the proximal promoter cannot be defined for any of the three promoter classes as each gene type has different needs for expression regulation. Some examples of specifically needed sequences are the sterol response element found in genes involved in lipid metabolism and the N-box with consensus sequence CACNAG located in genes who are expressed at synapses (EBI, 2019). More commonly found elements are the CCAAT-box and GC-box, with respectively CCAAT and GGGCGG as consensus sequences (Everett *et al.*, 1983; Cindy, 2007). However, they are sometimes also seen as part of the core promoter and little research is available on their exact location.

Enhancers are 10 to 1000 bp long elements that can positively influence the likelihood that transcription of a gene will take place (Clancy, 2008a; Li and Wunderlich, 2017). They are found up to 1 million bp upstream or downstream of a gene's TSS, and can also be found within a gene's introns or within another gene's exons (Pennacchio *et al.*, 2013). It is possible for one gene to have multiple enhancers and its orientation can be reversed without affecting its function (Murakami *et al.*, 1992). Similar to the proximal promoter, enhancers have no general structure and mostly contain TF binding sites that are specific to the kind of gene they regulate. Some binding sites can be found in both proximal promoters and enhancers, such as the vitamin D and serum response elements (EBI, 2019). Because of their similarity, enhancers are sometimes referred to as 'distal promoters' and the distinction between enhancer and promoter is not always clear.

2.4.3. Other regulatory elements

Silencers are structural blocks found in the same regions as enhancers that use similar mechanisms to affect transcription, with the exception that they influence the probability of transcription in a negative way (Maston *et al.*, 2006). Research on them is still ongoing and not much is known about them yet.
Other regulatory sequences are the UTRs found immediately before and after either end of the ORF. The preceding UTR is called the 5' UTR, while the one succeeding the last exon is the 3' UTR. Both regulate gene expression, which can be achieved by affecting the stability of the mRNA (Bashirullah *et al.*, 2001) and the translation efficiency (van der Velden and Thomas, 1999), among other mechanisms. Each UTR is part of the exon which its precedes or succeeds (Mignone *et al.*, 2002; Twyman, 2003) and is hence transcribed into mRNA. However, UTRs are not found in the final protein and they are therefore considered non-coding. The 3' UTR ends with a sequence called a terminator that signals for transcription to stop. Both UTRs are visualised in figure 2.3a.

In order to make sure the transcriptional elements of one gene do not influence the transcription of their neighbouring genes, insulators or boundary elements are inserted in the DNA (Maston *et al.*, 2006). These are 500 to 3000 bp sequences that block the activity of an enhancer (enhancer blockers) or a silencer (barriers) (Kolovos *et al.*, 2012). Some insulators can act as both enhancer blockers and barriers at once (West *et al.*, 2002). Similar to enhancers and silencers, they can function independently of their orientation in the genome.

3. Deep learning

Unless stated otherwise, references used for the neural network and deep learning approaches are Bishop (2006); Bengio (2009); Jones (2014); Karn (2016); Karpathy *et al.* (2016); Talwalkar (2016); Chollet (2017); Ng (2018); Ng and Katanforoosh (2018), and Ng (2019).

3.1. Origins within machine learning

Machine learning is a field of computer science that enables a computer to autonomously learn from a set of given data. By this automatic learning, the computer is able to make predictions on unseen data - akin to the data used to train the system - in order to solve a complex problem. For this learning to take place, the computer generally needs labelled training data to know what the output is expected to be, and a way to measure the distance between its current output and the expected output. This measurement provides feedback to the algorithm so it can adjust its inner workings in order to come closer to the expected predictions, and thus provides the algorithm with the ability to learn. If the training data are labelled, the process is called supervised learning, yet other cases exist where the algorithm can learn from unlabelled data. This is referred to as unsupervised learning, and examples of such techniques are k-means clustering and autoencoders. However, in this master dissertation, only supervised learning is applicable and unsupervised learning will therefore not be discussed further. The labels used in a supervised learning approach can either be continuous numerical values, or discrete values belonging to a certain class. The first is referred to as regression, and the latter as classification.

Today, various sorts of machine learning techniques are widely used worldwide in domains such as speech recognition, search engines, and bioinformatics Koza *et al.* (1996).

3.1.1. Data fitting and splitting

A machine learning model can be tuned to fit the training data nearly perfectly, but this does not necessarily mean it will produce good predictions on previously unseen data. When a model fails to generalise to additional data, this is called overfitting. Underfitting is also possible, where the model cannot capture the structure within the training data and fails to even output good predictions on seen data. Both overfitting and underfitting will lead to poor prediction results on unobserved data, and should therefore be avoided. This can be done by a variety of techniques, elaborated in section 3.2.5.

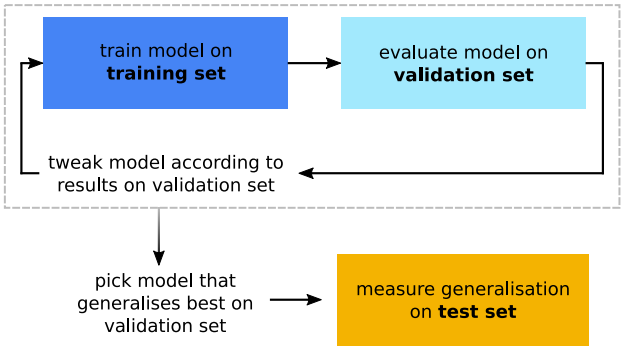


Figure 3.1: Machine learning model fitting workflow using holdout validation. The dashed box contains the training stage through which both the training and validation data make multiple passes (adapted from Google Developers (2018)).

To obtain a good model fit, the dataset is typically split into three disjoint subsets in order to evaluate the algorithm's performance. These subsets are referred to as the training data, the validation data, and the test data. All three datasets should be independent from each other and represent the same characteristics and structure. The training data are used to fit the model, and contain the samples from which the algorithm learns on its own. The model knows which label each sample within the training set has, and iteratively updates its parameters accordingly in order for its predicted outcome to come closer to the expected one. A

validation set is adopted to evaluate the current model's fit. The algorithm sees this validation set, but does not learn from it. This set provides an estimation on how the model will perform on unseen data, and by observing its results small tweaks can be made to the model in order to obtain a better generalisation. Both validation and training sets make multiple passes through the algorithm, and the continuous training, evaluating and tweaking is referred to as the training stage. After several passes of the data through this stage, the model with the best predictions on the validation set is chosen. However, this model should again be evaluated for its ability to generalise to unseen data as small bits of information about the validation set leak indirectly into the model. This is due to the repeated small tweaks made to the model based on its validation set performance, which can lead to overfitting. To avoid this, a test set is used to measure how well this final model generalises to data that is not directly, nor indirectly, seen by the model. The test set should therefore only be utilised once by the algorithm. A visualisation on how the different subsets are utilised during holdout validation is seen in figure 3.1.

The choice of the data split percentages depends on the number of samples in the dataset, and on the model and its number of parameters. If the dataset is not too large, k-fold cross-validation (CV) is used to reduce the risk of overfitting on the validation set Brownlee (2018). With this technique the original dataset is first split into two randomly chosen sets, called the train and test sets. A ratio of 80/20 is frequently used for this initial split. The test set is put aside to function as an actual test set, but the train set is randomly divided into k equally sized and disjunct groups or folds, and passes through k rounds of training stages. In each iteration, a different fold functions as the sole validation set, while the other $k-1$ folds function as the training data. Each fold can only be used once as a validation set. After model fitting, a performance metric M_i is given to and retained for each iteration i, and the model discarded so a new model can be trained on the next division of training and validation sets. After k iterations have passed, the average over all the performance scores is determined to yield an overall performance score $\mathrm{M} = \frac{1}{k}\sum_{i=1}^{k}\mathrm{M}_i$. This score M reflects how well the model is able to generalise to unseen data, and based on its value, the global properties of the model are optimised. Once the model properties have been fixed, the model is fit onto the training dataset as a whole without its subgroup distribution, and evaluated on the test set that was put aside. Figure 3.2 shows an example of 5-fold CV.

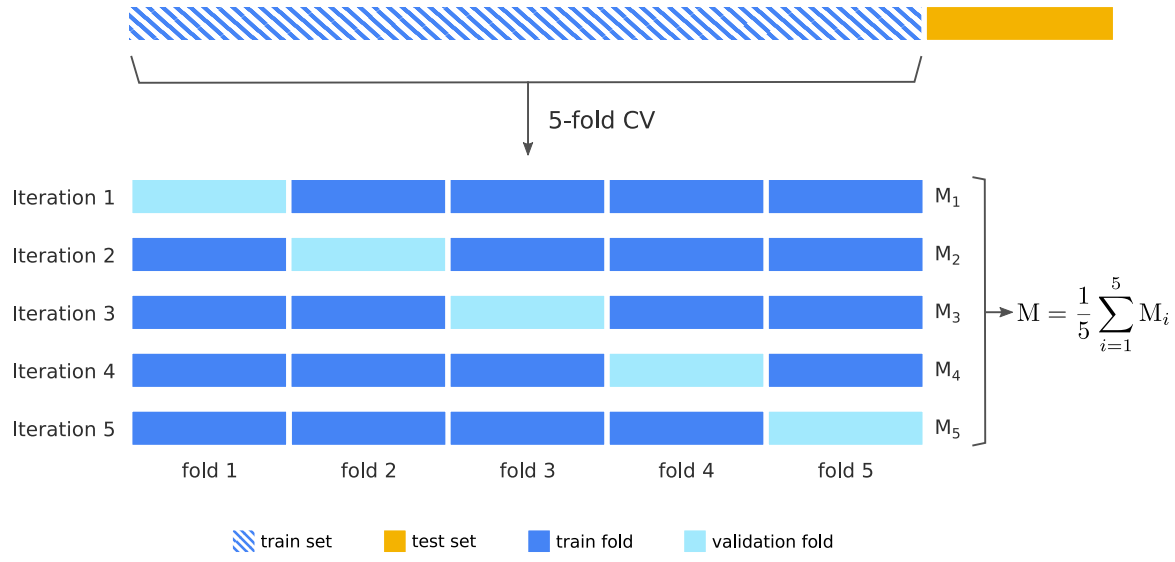


Figure 3.2: Example of k-fold CV with $k = 5$. The train and test set make up the original dataset that has been split using a ratio of 80/20, and the training set is subsequently divided into five groups or folds for use during 5-fold CV. The CV consists of five iterations, where in each iteration i a new model is fitted using four folds as training sets and the fifth fold as a validation set, and a performance score M_i is calculated for the best fitted model. Every iteration, the validation fold switches so every fold is used as a validation set only once. When CV has ended, an overall performance score M is calculated that reflects how well the model is able to generalise to unseen data, and the model is fitted onto the whole training dataset (striped block) and evaluated on the test set that was put aside (yellow block).

The programmer chooses the value of k, with $k = 10$ being a common option for small datasets. When k is equal to the number of samples within the training set, this technique is called leave-one-out CV. Random sampling can result in having folds where the number of labels within the same category differs significantly. To retain the same class distributions, stratified k-fold CV can be used to ensure that each fold gets assigned the same proportion of samples of each class category. For the performance metric M several approaches can be used, such as the proportion of the number of true results to the number of cases sampled (accuracy) and mean squared error (MSE).

3.2. Neural networks

Deep learning is part of the family of machine learning techniques that is generally referred to as artificial neural networks. It uses several levels of abstraction to learn from the data and solve the problem at hand. The data are broken down into simpler concepts that gradually get more complicated, and pass through various layers that can make transformations on the data so a solution to the problem is reached. An example of this broken-down abstraction can be found in figure 3.3. Here, the objective of the deep learning algorithm is to identify faces within a picture, and the algorithm will learn on its own which pixels, edges, and shapes are relevant for human face recognition and which are not. This process is called feature extraction, and is done by the deep learning model itself. This is in contrast to other machine learning techniques, where feature extraction is most commonly done explicitly by the programmer.

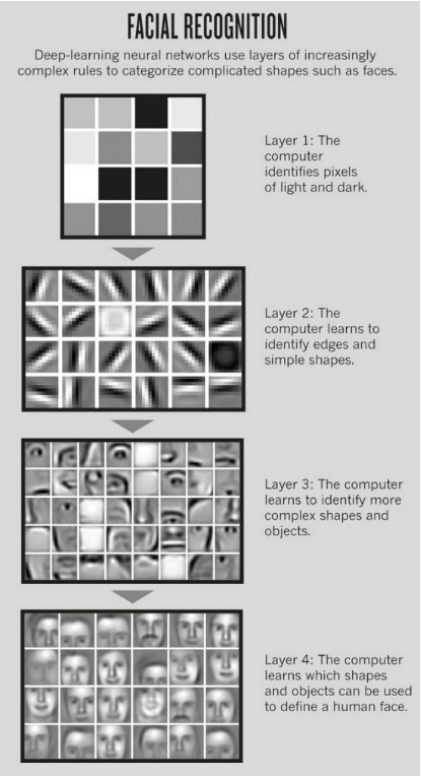


Figure 3.3: Levels of abstraction in a face recognition deep learning algorithm (Jones, 2014).

3.2.1. Input data

With supervised learning, the training data on which the neural network (NN) learns have to consist out of a numerical input vector **x** and a numerical true label vector **y**. Every sample should have the same length in order to fit into the input vector **x**. If this is not the case, the input vector gets as width the length of the longest sample, and shorter samples are padded with zero values.
If the raw data do not consist of numerical values, the data have to be encoded before they get passed into the network. Several encodings exist, such as one-hot encoding where different values are mapped onto different bits, and ordinal encoding where values are mapped onto decimal values between 0 and 1. Different encodings can lead to different research results and influence how well the NN is able to make predictions.

The input data are fed in batches to the network. One batch contains a certain number of samples that pass through the network, and the model's internal parameters are updated accordingly. Such a pass

is called an iteration. The algorithm goes through several of these iterations, until the total number of samples passed through the network is equal to the number of samples present in the training set. Then the so-called epoch ends, and the process of batches and iterations starts again.

3.2.2. General layout

A deep learning model can be considered as a deep NN, a term which has its origin within nature as it vaguely resembles the way biological nervous systems work. Both models' computational cornerstones are the neurons, which can transfer signals to other neurons within their network. In an artificial NN, the neurons are grouped into layers, and individual neurons are connected across layers through edges. Neurons receive signals from either an external source or from a neuron from another layer, and the i^{th} input of a neuron is referred to as x_i, with $\mathbf{x} = [x_0, \ldots, x_n]^\intercal$ the column vector of all inputs of that neuron. This input vector also has a column vector of weights $\mathbf{w} = [w_0, \ldots, w_n]^\intercal$ associated with it, with the i^{th} weight of the x_i^{th} input referred to as w_i. The first elements of the input vector **x** and the weight vector **w** are special cases as they have no other connections with the network except for the one going into the calculating neuron. These values are respectively called the bias x_0, which has a non-adjustable value of 1, and the bias weight w_0, which has a variable value.
All weights tied to a neuron are associated with the edges across the layers, and indicate how important each input is relative to the others, with higher absolute values indicating a higher importance. They are parameters of the NN that can be adjusted by the algorithm during learning. To produce an output, the neuron calculates the weighted sum $\sum_{i=0}^{n} w_i x_i$ over its inputs, which can also be rewritten as the dot product $\mathbf{w}^\intercal\mathbf{x}$. After this linear operation, an activation function is applied to provide non-linearity. The choice of activation function can vary and is generally referred to as $f(\cdot)$. By adjusting the bias weight, the neuron is able to translate the activation function. If no translation is needed, the bias weight is simply set to zero. Applying the activation function on the weighted sum leads to following equation to calculate the activated output a in a neuron:

$$\begin{aligned} a &= f\left(\sum_{i=0}^{n} w_i x_i\right) \\ &= f\left(\mathbf{w}^\intercal\mathbf{x}\right) \end{aligned} \tag{1}$$

This output a can be the final output of the NN, or it can be passed on to the next neuron to serve as new input. It can be regarded as a new feature learned by the neuron based on the already existing features **x**. As an NN consists of a series of these neurons grouped into several layers which transfer their activated output to each other, the NN will learn a hierarchy of features which get adjusted by altering the weights associated with the neurons, and which gradually get more complex as they are a mix of previously learned features. This allows for an NN to create potentially better predictions than more classical machine learning approaches which only work on the original features within the data. An illustration of a single neuron can be found in figure 3.4.

Three sorts of layers can be distinguished in an NN. The layer which receives the data is called the input layer, and its inputs represent the original features within the dataset. This layer passes the features to another layer, called the hidden layer. A hidden layer consists of several neurons which take the activated weighted sum over their inputs, and then pass this sum along to the next layer. This next layer can be another hidden layer, or an output layer. The output layer is similar to a hidden layer, but as it is the last layer within the network, it produces the predictions. For a regression problem, only one neuron is needed that outputs a single value $\hat{y} \in \mathbb{R}$. For classification problems, the number of neurons is equal to the number of classes K within the input data. Per sample, the neurons give back a probability vector $\hat{\mathbf{y}} \in \mathbb{R}^K$ representing how sure the NN is that the fed data belongs to each class. All probabilities inside this vector are within the $[0, 1]$ range and sum up to 1. Binary classification is a special case, as here one neuron in the output layer is sufficient. A fully connected DNN with three hidden layers is shown in figure 3.5. The totality of the number and types of layers, their number of neurons and activation functions, as well as the way they are interconnected is typically referred to as the architecture of the

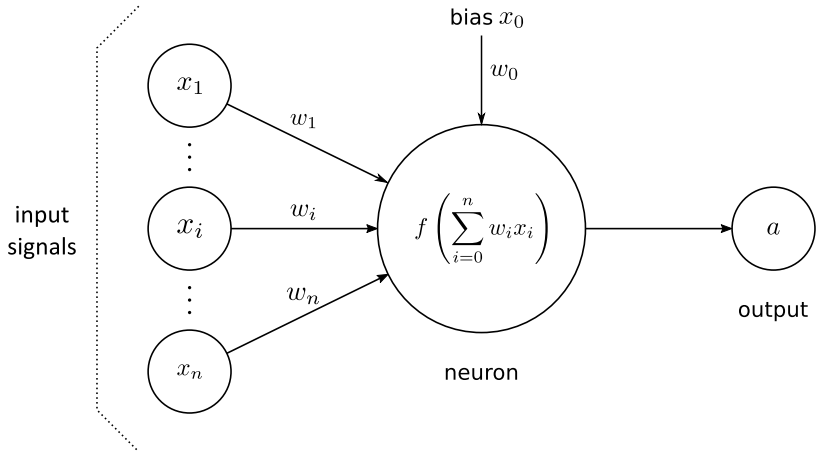


Figure 3.4: Visual representation of a single neuron in a NN.

network.

Equation (1) can now be redefined for the output $a_{j,k+1}$ of a single neuron j within hidden layer $k+1$:

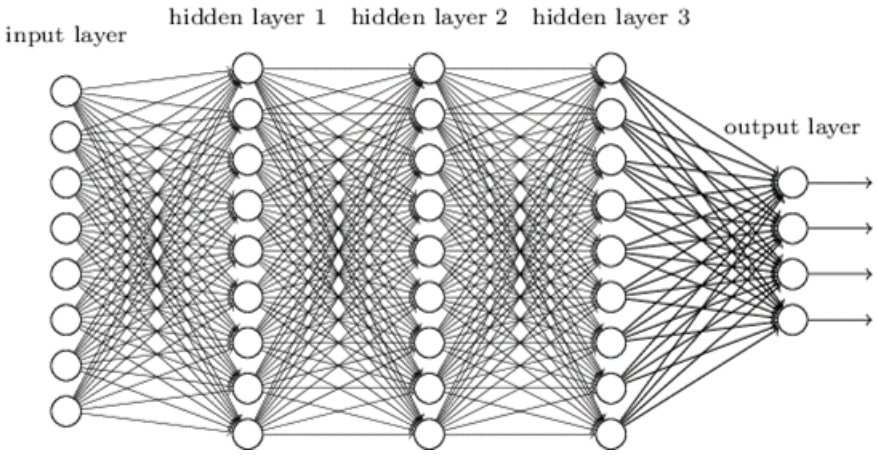


Figure 3.5: Fully connected feedforward DNN with three hidden layers (Ho, 2017).

$$\begin{aligned} a_{j,k+1} &= f\left(\sum_{i=0}^{n} w_{(i,k),(j,k+1)} \cdot a_{i,k}\right) \\ &= f\left(\mathbf{w}_{k,(j,k+1)}^{\mathsf{T}} \mathbf{a}_k\right) \end{aligned} \tag{2}$$

with $w_{(i,k)(j,k+1)}$ the connection weight from neuron i in layer k to neuron j in layer $k+1$, $a_{i,k}$ the output of neuron i in layer k, $\mathbf{w}_{k,(j,k+1)} = [w_{(0,k),(j,k+1)}, \ldots, w_{(n,k),(j,k+1)}]^{\mathsf{T}}$ the column vector holding all the weights of the connections coming from the n neurons in layer k into neuron j in layer $k+1$, and $\mathbf{a}_k = [a_{0,k}, \ldots, a_{n,k}]^{\mathsf{T}}$ the column vector holding all the outputs from the n neurons in layer k. Note that in the last two column vectors $w_{(0,k),(j,k+1)}$ and $a_{0,k}$ are special cases that correspond to respectively the variable weight and fixed value of the bias of neuron j in layer $k+1$.

As the output vector $\mathbf{a}_k$ is needed to calculate the output $a_{j,k+1}$ of a single neuron j within hidden layer $k+1$, this output vector should also be defined. This is done by associating the current layer with the output of the previous layer, so that the output vector $\mathbf{a}_{k+1}$ for a hidden layer $k+1$ is given by:

$$\mathbf{a}_{k+1} = f\left(\mathbf{W}_{k,k+1}^{\mathsf{T}} \mathbf{a}_k\right) \tag{3}$$

with $w_{(i,k),(j,k+1)}$ the connection weight from neuron i in layer k to the j^{th} neuron in layer $k+1$, $a_{i,k}$ the output of neuron i in layer k, $\mathbf{a}_k = [a_{0,k}, \ldots, a_{n,k}]^{\mathsf{T}}$ the column vector holding all the outputs from the

n neurons in layer k, and $\mathbf{W}_{k,k+1}$ an $n \times m$ weight matrix associated with the biases of the m neurons in layer $k+1$ and the connections from the n neurons in layer k going into the m neurons in layer $k+1$ or:

$$\begin{aligned}\mathbf{W}_{k,k+1} &= \begin{bmatrix} w_{(0,k),(1,k+1)} & \cdots & w_{(0,k),(j,k+1)} & \cdots & w_{(0,k),(m,k+1)} \\ \vdots & \ddots & \vdots & & \vdots \\ w_{(i,k),(1,k+1)} & & w_{(i,k),(j,k+1)} & & w_{(i,k),(m,k+1)} \\ \vdots & & \vdots & \ddots & \vdots \\ w_{(n,k),(1,k+1)} & \cdots & w_{(n,k),(j,k+1)} & \cdots & w_{(n,k),(m,k+1)} \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{w}_{k,(1,k+1)} & \cdots & \mathbf{w}_{k,(j,k+1)} & \cdots & \mathbf{w}_{k,(m,k+1)} \end{bmatrix}\end{aligned} \tag{4}$$

with $w_{(i,k),(j,k+1)}$ the weight of the connection from neuron i in layer k to neuron j in layer $k+1$, and $\mathbf{w}_{k,(j,k+1)} = [w_{(0,k),(j,k+1)}, \ldots, w_{(n,k),(j,k+1)}]^\mathsf{T}$ the column vector holding all the weights of connections coming from the n neurons in layer k into the j^{th} neuron in layer $k+1$. The first row of this matrix holds the bias weights for the m neurons in layer $k+1$.

Equation (3) can be used to associate the output layer $k = L+1$ and its prediction outputs $\hat{\mathbf{y}}$ with all the previous layers, up until the first hidden layer $k = 1$ whose output depends on the input vector $\mathbf{x}$. The network is then represented as a composition of a series of activation functions, such that:

$$\begin{aligned}\hat{\mathbf{y}} &= f_{L+1}\left(\mathbf{W}_{L,L+1}^\mathsf{T}\mathbf{a}_L\right) \\ &= f_{L+1}\left(\mathbf{W}_{L,L+1}^\mathsf{T} f_L\left(\mathbf{W}_{L-1,L}^\mathsf{T}\mathbf{a}_{L-1}\right)\right) \\ &= f_{L+1}\left(\mathbf{W}_{L,L+1}^\mathsf{T} f_L\left(\mathbf{W}_{L-1,L}^\mathsf{T} \cdots f_{k+1}\left(\mathbf{W}_{k,k+1}^\mathsf{T} f_k\left(\mathbf{W}_{k-1,k}^\mathsf{T} \cdots f_1\left(\mathbf{W}_{0,1}^\mathsf{T}\mathbf{x}\right)\right)\right)\right)\right) \\ &= h\left(\mathbf{x}, \mathbf{W}\right)\end{aligned} \tag{5}$$

with f_k the activation function used in the k^{th} layer of the network, and $\mathbf{W} = [\mathbf{W}_{0,1}, \cdots, \mathbf{W}_{k,k+1}, \cdots, \mathbf{W}_{L,L+1}]$ the matrix holding all the weight matrices associated with each layer. A DNN can thus be regarded as implementing a function $\hat{\mathbf{y}} = h(\mathbf{x}, \mathbf{W})$ that maps a set of inputs $\mathbf{x}$ to a set of outputs $\hat{\mathbf{y}}$, controlled by a matrix $\mathbf{W}$ holding the adjustable weight and bias weight parameters. As each layer needs the previous one to calculate its outputs, data flows through the network in a feedforward manner. No connections are found between neurons within the same layer or across non-consecutive layers, although special network structures exist with feedback loops such as recurrent neural networks (RNNs). Initially, the weights of the NN are set to random values, and the algorithm alters them by comparing the final predictions $\hat{\mathbf{y}}$ to the true values $\mathbf{y}$. This comparison is done by the use of a cost or loss function $J(\mathbf{W}) = \mathcal{L}(\mathbf{W}) = L(\mathbf{y}, \hat{\mathbf{y}})$ which expresses the importance of the errors that are made. The cost function is what the algorithm needs to minimise in order to come closer to the expected output. As the only variable values in the cost function are the weights $\mathbf{W}$, an optimal weight matrix $\mathbf{W}^*$ exists which will result in the smallest loss possible. It is found by minimising the loss function:

$$\mathbf{W}^* = \arg\min_{\mathbf{W}} \mathcal{L}(\mathbf{W}) \tag{6}$$

The optimal prediction vector $\hat{\mathbf{y}}^*$ is then defined by:

$$\hat{\mathbf{y}}^* = h(\mathbf{x}, \mathbf{W}^*) \tag{7}$$

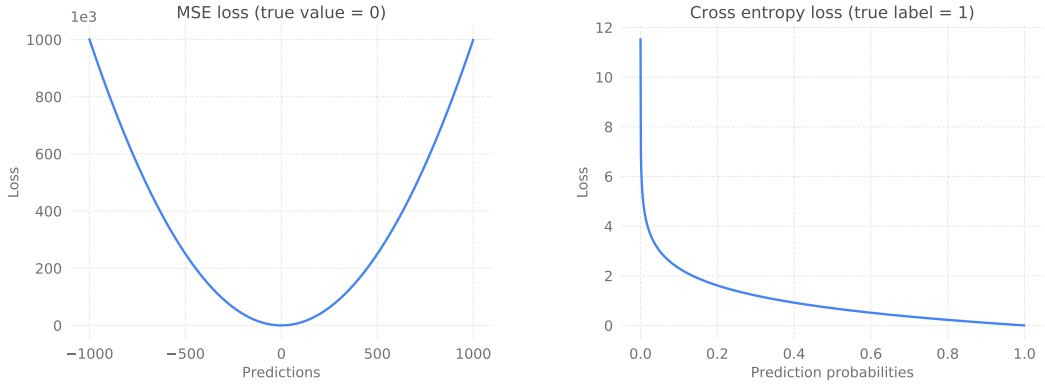
3.2.3. Loss functions and their minimisation

The two most important and most frequently used loss functions are the MSE function and the cross-entropy function. They are respectively used in regression and classification problems, and have following formulas:

$$\mathcal{L}_{\text{MSE}}(\mathbf{W}) = \frac{1}{n-1}\sum_{i=0}^{n-1}(y_i - \hat{y}_i)^2 \qquad \mathcal{L}_{\text{cross-entropy}}(\mathbf{W}) = -\sum_{i=0}^{n-1}\sum_{c=1}^{M} y_{i,c} \log \hat{y}_{i,c} \tag{8, 9}$$

with n the total number of samples present in the dataset, M the number of classes within the dataset, $y_{i,c}$ a binary indicator showing if class c is the correct classification for sample i, and $\hat{y}_{i,c}$ the predicted probability of sample i belonging to class c. The MSE loss will result in a high loss when the predicted value is far away from the true value, and the cross-entropy loss punishes uncertain prediction probabilities. Plots for both functions are given in figure 3.6.

The minimum value of a loss function is reached when the predicted values $\hat{\mathbf{y}}$ are equal to the true



(a) MSE loss function

(b) Cross-entropy loss function

Figure 3.6: Loss plots for the MSE and cross-entropy loss functions. Both functions rapidly rise to a higher loss value when the prediction values get further away from the true value.

values $\mathbf{y}$. However, analytically solving equation (7) is computationally impossible due to the large number of parameters present in a DNN. Therefore, an algorithm called gradient descent is applied to find a good approximation to the true minimal value of the loss function, and its associated approximation of the optimal weight matrix $\mathbf{W}^*$.

Various variants of the gradient descent algorithm exist, such as gradient descent with momentum or an adaptive learning rate. These variations address several problems with the base algorithm and the choice of which variant to use depends on the problem at hand. Their core mechanisms are similar to the base algorithm of stochastic gradient descent discussed in the next paragraph, and these variants are therefore not discussed further.

Stochastic gradient descent and backpropagation

Stochastic gradient descent is an optimisation algorithm designed to find the minimum of a given function. It does this by calculating the function's negative gradient in a certain point, updating the function's parameters accordingly, and evaluating the function again in the same point but with its newly set parameters. This results in a new point with a lower function value than the initial one. The loop repeats itself by the calculation of the negative gradient in the newly found point, and goes on until no or only small changes occur in the values of these newly calculated points. The gradient descent algorithm is thus slowly descending the function in small steps in order to reach the lowest value, while continuously updating the parameters of the function.

A typical NN consists of millions of weight parameters, resulting in a high dimensional space in which the loss function exists. The initial point for a certain step in the gradient descent algorithm can be defined by the weight matrix $\mathbf{W}_0$, and its gradient by $\nabla\mathcal{L}(\mathbf{W}_0)$. Every entry in this gradient matrix indicates how the loss value is influenced if only that certain entry is modified, while the whole gradient matrix describes the curvature of the loss function around the point $\mathbf{W}_0$. By taking the negative gradient $-\nabla\mathcal{L}(\mathbf{W}_0)$, one goes against this curvature and descend in the high dimensional space. The gradient descent algorithm thus goes from an initial weight matrix $\mathbf{W}_0$ to a point that is slightly lower by descending along its gradient, resulting in a new weight matrix $\mathbf{W}_1$:

$$\mathbf{W}_1 = \mathbf{W}_0 - \gamma \cdot \nabla\mathcal{L}(\mathbf{W}_0) \tag{10}$$

with γ the learning rate. The learning rate controls the size of the steps that the gradient descent algorithm takes while descending along the gradient. If γ is set to a large value, the algorithm will tend to overshoot the minimum value, potentially leading to an infinite loop. A small value for γ leads to in a slow convergence towards the lowest value, resulting in an algorithm that takes a long time to finish. The learning rate can be evaluated by plotting the error on the training set during training, where a good learning rate results in a steady descend towards zero loss. A learning rate that is set too high will result in a loss that stays high, while a learning rate that is set too low will result in a loss that descends very slowly towards zero.

As seen in equation (5), the predictions $\hat{\mathbf{y}}$ of a NN are the product of a series of activation functions. Calculating the gradient of the loss function and updating the weights is therefore a complex operation, and both are done by the use of a technique called backpropagation. This technique is based on the chain rule, where the derivative of a composition of functions can be calculated by the product of their derivatives. As the deepest layer of the DNN is the one that depends upon all the previous ones, backpropagation goes through the network in a backwards manner and the first layer is the last one updated. Each activation function should also be differentiable in every point, as otherwise no gradient can be determined.

Vanishing and exploding gradients

Two problems that commonly occur when training DNNs are vanishing and exploding gradients. Both lead to a network that fails to learn meaningful features of the data.

The vanishing gradient problem arises when the gradient of the loss function gets close to zero. This is due to the chain rule used in the backpropagation algorithm, where the derivative of a layer is equal to the multiplication of the derivatives of all the following layers. Small derivative values that occurred in the last layers get multiplied while backpropagating through the network, leading to even smaller derivative values in the first layers. These first layers then fail to get meaningful updates to their weights and biases, resulting in a network where no learning occurs in those first layers. As these layers are essential in recognising the core features in the data, this results in a network with poor prediction abilities. Small derivative values are typically seen when activation functions are used where a large input range is mapped onto a small output range. The simplest solution is therefore choosing an activation function where the input data is not mapped onto a closed output range, but onto an unbounded one. Batch normalisation is also frequently used, where the data that a hidden layer receives is normalised - and thus mapped onto a smaller input range - before its output is calculated. A more complex approach is the use of residual connections in the network. While normally each layer passes its output to the next layer, residual connections can skip one or more layers and pass their output to a layer that is more than one step away from them. This results in a smaller chain of multiplications of small derivatives, leading to an overall larger gradient value for the entire loss function.

Exploding gradients refer to gradients that get uncontrollably large, again due to the chain rule where the multiplication of large values eventually leads to even larger values in the first hidden layers. Large gradient values result in large updates to the weight parameters, and in large weights in general. These make the network unstable, such that a small variation in the input data will lead to large differences in the output. The network will be sensitive to noise in the input data, and fails to output meaningful predictions. In the worst case, the exploding gradients lead to an overflow in the loss or weight values, resulting in not-a-number (NaN) values which completely stop the learning process.
Apart from changing the model's architecture, gradient clipping and weight regularisation can be applied to solve the exploding gradient problem. Gradient clipping does this by mapping the calculated gradients back to a smaller range, or cutting off gradients that are too large by setting them back to a smaller absolute value. The weights are then calculated with smaller gradients, leading to smaller weights than when calculated with the non-clipped gradients. While gradient clipping solves the problem on the gradient level, weight regularisation still allows large gradients but will punish the network for having large weights. To achieve this, a regularisation term is added to the cost function $J(\mathbf{W})$, such

that $J(\mathbf{W}) = \mathcal{L}(\mathbf{W}) + \lambda \cdot \Phi(\mathbf{W})$, with λ the regularisation parameter that indicates the amount by which large weights are penalised, and $\Phi(\mathbf{W})$ the regularisation function. The regularisation term outputs a higher value for larger weights, resulting in a higher cost value. This way, the network is forced to keep the weights small in order to minimise the errors. For the regularisation function, typically the L1- or L2-norm are used, or a combination of both (referred to as elastic net). The L1-norm regularisation term is calculated by taking the sum over all the absolute values of the entries in the weight matrix **W**. Due to its derivative, it introduces a sparse weight matrix where the majority of the weights are equal to zero. Because of this, the L1-norm is able to perform feature selection by setting the weights associated with non-useful features to zero. It is robust to outliers, but will not be able to generate complex models. The L2-norm on the other hand, takes the squared value of all the entries in the weight matrix **W** and sums them up, generating complex models where weights are never set to zero, but only to very small absolute values. All features are thus still taken into account and no feature selection is performed. It is not robust to outliers as the squared value of the weights will stress the outliers even more.

3.2.4. Activation functions

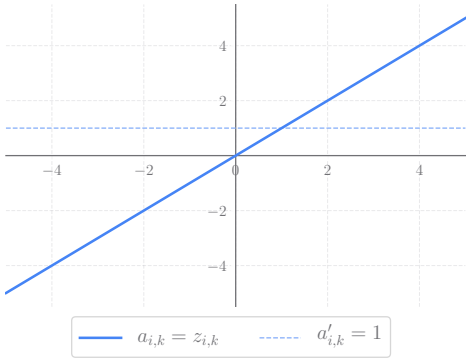
Activation functions define the output of a neuron, and are typically non-linear. For easy clarification, equation (2) for the activated output $a_{i,k}$ for a neuron i in layer k is redefined as $a_{i,k} = f(z_{i,k})$, with f the activation function, and $z_{i,k} = \mathbf{w}_{k-1,(i,k)}^{\mathsf{T}} \mathbf{a}_{k-1}$ the non-activated value of neuron i in layer k. Some commonly used activation functions and their derivatives are plotted in figure 3.7.

The first activation function, the linear function (figure 3.7a), is typically not used in the hidden layers of a NN. This is due to three main reasons. A derivative that is equal to a constant value will result in a backpropagation that makes no progress in updating the weights of the network. Secondly, when only linear activations are used, the final output of the network will be a linear combination of its input, reducing the NN to a simple linear regression model that lacks the power to handle complex input data. The last reason is the unconstrained nature of the output range of the linear function. It can produce large values which only get larger when propagated through the further network, eventually leading to uncontrollably large calculations. However, the linear function has its use in regression problems, where only the output layer of the network has a linear activation as here the predicted values need to be unconstrained.

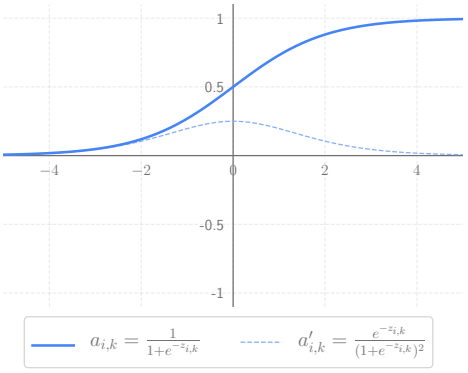
The sigmoid (figure 3.7b) and hyperbolic tangent (figure 3.7c) functions solve the problems that come with the linear function. They are able to introduce non-linearity in the NN, have a non-constant function as derivative, and map large inputs back to small outputs due to their constrained nature. The sigmoid function is however prone to vanishing gradients, and is not centered around zero. This latter results in gradients that go too far in either the positive or negative direction, making optimisation harder when the sigmoid function is used. The hyperbolic tangent does not suffer from a harder optimisation as its values are centered around zero. It however does not solve the problem of vanishing gradients.
The currently preferred activation function to use in hidden layers is the rectified linear unit (ReLu) function (figure 3.7d). It has a six times faster convergence than the hyperbolic tangent function due to its formula being simpler in nature, and does not suffer from vanishing gradients. However, it can introduce dead neurons, where neurons that are not activated will never be updated again during backpropagation. This can be solved by replacing the zero value for negative values by a linear function with a slight slope. This solution is referred to as the leaky ReLu. Note that the derivative for the ReLu function should be undefined in $a_{i,k} = 0$, but is instead set to 1 in order to avoid problems with gradient descent.

The last important activation function is the softmax function, which is used in the output layer during classification tasks. This function will turn scalar values into probabilities for each of the n classes. Each probability lies in the $[0, 1]$ interval, and the sum over all n probabilities is equal to 1. Its formula is given by:

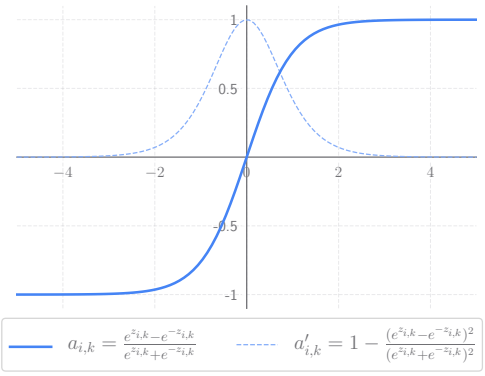
$$a_{i,k} = \frac{e^{z_{i,k}}}{\sum_n e^{z_{n,k}}} \tag{11}$$



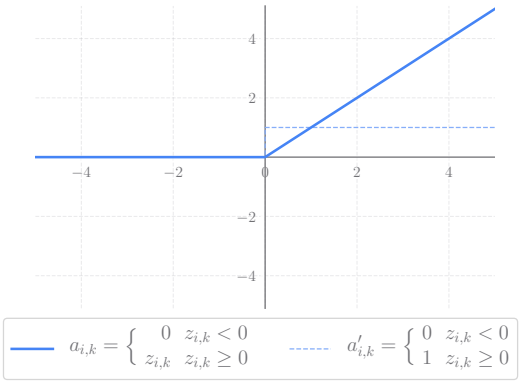
(a) Linear function



(b) Sigmoid function



(c) Hyperbolic tangent function



(d) ReLu function

Figure 3.7: Activation functions typically used in NNs. The blue lines indicate the activation funtion itself, while the dotted blue lines are their derivatives. For all subfigures, the x-axis is equal to $a_{i,k}$, and the y-axis to $z_{i,k}$.

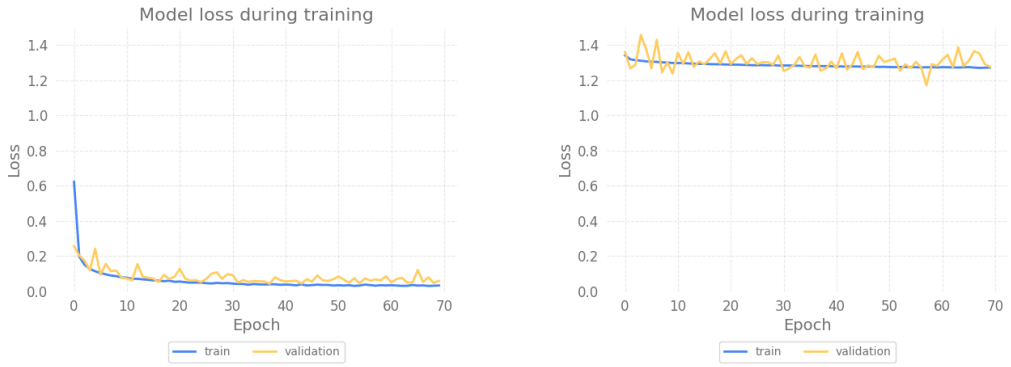
with $z_{n,k} = \mathbf{w}_{k-1,(n,k)}^{\mathsf{T}} \mathbf{a}_{k-1}$ the non-activated value of the n-th neuron in layer k, and $\sum_n e^{z_{n,k}}$ the sum over all the non-activated values of the n neurons in layer k.

3.2.5. Recognising and solving over- and underfitting

Both over- and underfitting refer to a model that is not able to generalise well on unseen data, and can be recognised by looking at the evolution of the loss value during model training. Under ideal circumstances, the loss of both the training and validation set should be low (figure 3.8a). When one or both of these losses has a significantly high value, under- or overfitting occurs.
A high training loss means the model is underfitting. It is accompanied by a high validation loss, as the model fails to capture the structure in the training data and thus will not generalise well to unseen data either (figure 3.8b). Overfitting is encountered when a model has a high validation loss, but a low training loss (figure 3.8c). It happens when a model learns too much detail or random noise in its training data. These learned details and noise are however not present in unseen data, resulting in a model that captures the training data nearly perfectly yet outputs poor predictions for unseen data.

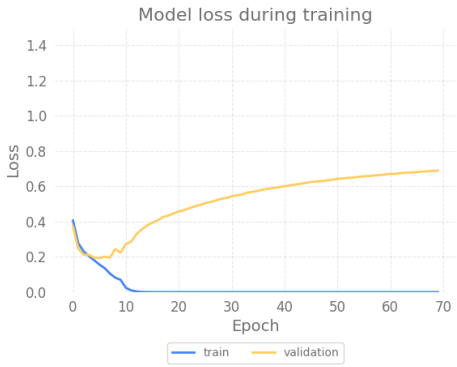
Underfitting is a problem that can easily be solved by extending the network and introducing more parameters that can capture the complexity of the input data. Overfitting on the other hand is a more complex problem that requires more thorough techniques to reduce it.

The easiest solution to reduce overfitting is simply to gather more data. This is however not an option in most of the cases and data augmentation is then a viable alternative. With data augmentation, new



(a) Ideal situation where no under- or overfitting occurs. The NN has a nearly perfect fit to the data at hand.

(b) Underfitting on the dataset. The NN fails to capture the structure in the training data, and thus cannot generalise well to the validation data either.



(c) Overfitting on the dataset. The NN learns to pick up tiny details in the training set, resulting in a poor fit on the validation set which lacks those same details.

Figure 3.8: Three different DNN architectures run on the same dataset, with the train and validation loss plotted after training for 70 epochs.

samples are created by slightly altering the original ones. In the case of images as input data, a variety of transformations exist such as flipping, translation, and rotation. The size of the dataset is increased by a factor equal to the number of transformations that were performed. If the dataset is small enough to fit into a computer's memory, the augmentation can be done offline by applying it before training takes place. However, if the dataset is too large, real-time or online augmentation is used, where the augmentation is applied on the batches that are fed to the network during training. Another simple technique is to reduce the size of the network. A larger network equals more parameters, resulting in a network that is able to pick up more detail and noise than a smaller one. By making the network smaller, the model is forced to shift its main focus back to patterns that actively contribute to the task at hand. A smaller network size can be achieved by removing hidden layers, or by reducing the number of neurons in the different layers.

More advanced methods to solve overfitting are early stopping and dropout. Early stopping stops the training process before overfitting can occur. This is done by monitoring a certain metric of the validation set, such as its loss. Several early stopping schemes exist, such as monitoring if the loss keeps increasing over a number of epochs or if the absolute loss increase is equal or bigger than a certain value. When the applied scheme is triggered and training stops, the model with the last most optimal loss value on the validation set is then set as the final model. When the dropout method is used, the output of randomly chosen neurons is set to zero during training. This helps with overfitting as neurons in a network will become co-dependent on each other during training. By dropping some of them, the others neurons are forced to learn meaningful features on their own again, resulting in a more robust network. The

chance that a neuron is ignored during training is equal to p, with p a hyperparameter of the dropout layer. During testing no neurons are set to inactive, but every neuron's output is reduced by a factor p in order to account for the missing activations during the training phase.

3.2.6. Hyperparameters

Hyperparameters are parameters of the network that are chosen by the scientist, and are set before training takes place. Examples are the number of hidden layers, the total number of neurons in each layer, activation functions, and number of epochs. Hyperparameters either determine the network size and structure (model parameters) or indicate how the network is trained (optimiser parameters). The performance of a model can be optimised by tweaking the hyperparameters. This can be done manually or by an automatic search. While the former requires thorough understanding on how deep learning works and is labour-intensive, the latter comes with a high computational cost to loop through a high number of parameter combinations.

3.3. Convolutional neural networks

A fully connected DNN gives rise to a rapidly exploding number of parameters. This is especially troublesome when the input datahas three or more dimensions, such as images (3D; 2D and three colour channels) and videos (4D, 3D and numerous frames), as one neuron in a fully connected layer would have as many connections as the element-wise multiplication of the dimensions of the input data. For example, an input image of size $100 \times 100 \times 3$ would lead to $100 \cdot 100 \cdot 3 = 30.000$ connections for each neuron. CNNs deal with this problem by constricting the number of connections a neuron has between two consecutive layers. This connection to only a small subset of neurons in the previous layer is called the receptive field of a neuron, and greatly limits the number of parameters present in the network. Learning takes place by looking at smaller and simpler patterns in the data, which are later assembled into bigger and more complex ones in the deeper layers. A typical CNN has an architecture similar to that of a normal DNN, but has two extra layers called the convolutional and pooling layer stacked between the input layer and the fully connected layers. The neurons of these layers are stacked in a 3D manner, as opposed to the typical 2D arrangement seen in normal DNNs. These three dimensions are referred to as width, height and depth, and the data that a layer receives or produces are respectively called the input or output volume. An overview of an example CNN architecture is given in figure 3.9.

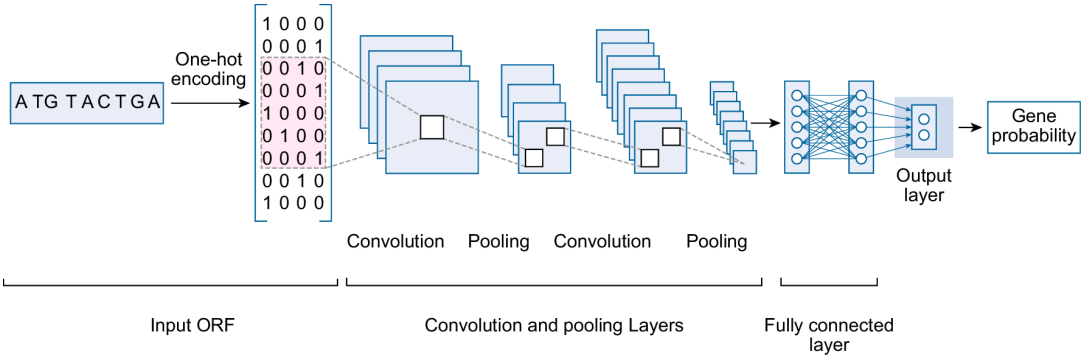


Figure 3.9: High level overview of a CNN for use with one-hot encoded DNA data (Al-Ajlan and El Allali, 2018). It consists of two consecutive series of a convolution and max-pooling layer, followed by two fully-connected layers.

3.3.1. Convolutional layer

Most of the core computations of a CNN are all done in this layer. Here, a matrix window that is small in width and height but goes through the full depth of the input volume slides in small steps across the entire width and height of the input volume. This window is called a filter or kernel and has a size F

that is seen as a hyperparameter of the convolutional layer. Every one of its elements can be adjusted independently from the other elements in the window, and can be regarded as the weights of the layer. For every slide the window does, it computes the dot product between its own entries and the seen input, and outputs thus a single value for that exact position. As it slides over the input volume, a series of single values is outputted, resulting in a 2D feature map of the 3D input volume. Several of these filters can be applied to the same input volume which all produce a 2D feature map, and every one of these maps recognises other patterns in the data. The feature maps are stacked along the depth dimension, creating the new 3D output volume. This output volume will thus not always have the same width and height dimensions as its associated input volume, and its depth K is equal to the total number of filters applied to the input volume. This number K is also considered a hyperparameter of the convolutional layer.
The width and height dimensions of the output volume are controlled by two other hyperparameters called the stride S and the zero-padding P. The stride S refers to the size of the steps that are taken when a filter is sliding over the input volume. When the step size is 1, then the filter moves from one entry in the input volume to the other consecutively. When the stride is set to a larger number, the filter will skip some entries, resulting in a smaller output volume. The zero-padding hyperparameter P indicates if an extra border of zeros is added around the input volume and how wide that border is. By adding padding, a filter can also be applied at the edges and corners of an input volume. If no padding is added, these entries cannot be used as they lack certain neighbouring values needed to compute the dot product. This way, the original dimensions of the input volume can be preserved or even expanded. The stride S and zero-padding P hyperparameters can be used together with the size F of the filter and the width and height dimensions of the input volume V to calculate the width and height dimensions of the output volume W:

$$W = \frac{V - F + 2P}{S} + 1 \tag{12}$$

Every entry in the output volume can be regarded as the output of a single neuron. This neuron only has connections with the neurons in its immediate vicinity, namely the neurons whose output values were used in the calculation of the dot product with the kernel. This reduced number of connections along the first two dimensions is called the receptive field of the neuron and is equal to the size F of the kernel. While this kernel sees only a small part of the input volume along these dimensions, it goes through the full depth of the input volume. This means a neuron has as many connections along the depth axis as the depth of the original input volume. Its total number of connections along all dimensions is then equal to the element-wise multiplication of the width and height size F of the kernel and the depth of its received input. Also note that each neuron has as many weights as it has connections plus 1, as a bias still has to be added. If one now looks back at the example of an input image with size $100 \times 100 \times 3$, the convolutional layer that directly follows the input layer will receive an input volume with the exact same dimensions as the original data. When a filter with size 2×2 is applied, a single neuron in that layer will then have only $2 \cdot 2 \cdot 3 = 12$ connections, instead of the 30.000 in a fully connected NN.
Another intervention is needed however to reduce the number of parameters in a convolutional layer. To illustrate this, we will calculate the size of the output volume of the convolutional layer in the above example with a stride $S = 1$, a padding of $P = 0$, and $K = 128$ applied filters. Using equation (12), the output volume W is equal to:

$$\begin{aligned} W &= \frac{100 - 2 + 2 \cdot 0}{1} + 1 \\ &= 99 \end{aligned}$$

The output volume thus has a dimension of $99 \times 99 \times 128$. As each output is associated with a neuron, the number of neurons in this convolution layer is equal to $99 \cdot 99 \cdot 128 = 1.254.528$. As previously calculated, each neuron has 12 connections with 13 accompanying weights. This finally results in a total parameter number of $1.254.528 \cdot 13 = 16.308.864$ for just this one single convolutional layer, which

would quickly lead to overfitting. To solve this problem, parameter sharing is applied. The idea behind this is that if a feature is useful to calculate at one position of the input volume, then it will also be useful to calculate that exact same feature at another position of the input volume. Every neuron that is part of the same feature map can thus share the same parameters, resulting in K unique sets of weights and biases. In the given example, this means that there would only be 128 different sets of weights and biases, where each set consists of 13 parameters, resulting in a total number of parameters of $128 \cdot 13 = 1664$. This parameter sharing scheme can also be relaxed if the network has to learn different features on each side of its input.

After the output volume of a convolutional layer is calculated, it is typically activated by the ReLu function before it gets passed to the following layer. To illustrate the inner workings of a convolutional layer, a visual example is given in figure 3.10.

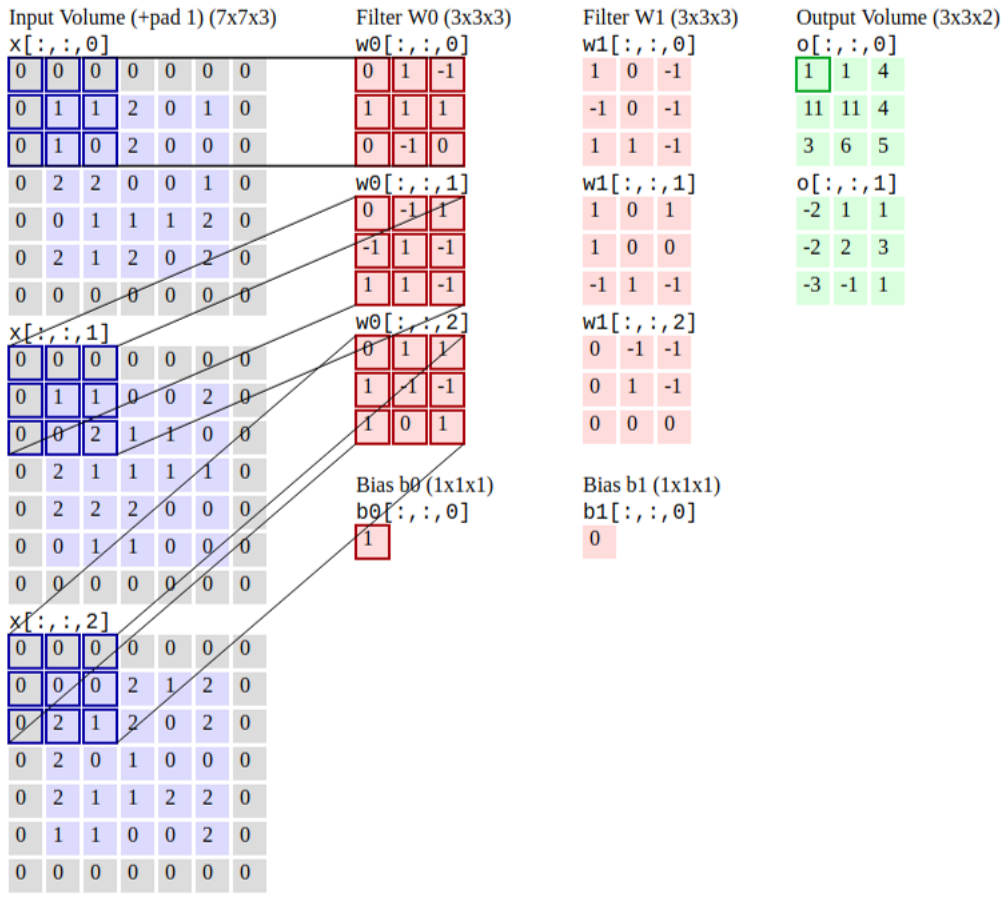


Figure 3.10: Inner workings of a convolution layer (Karpathy *et al.*, 2016). Blue indicates the 3D input volume, where the third dimension (depth = 3) is illustrated as a stack of 2D inputs. Red indicates the filters (size F=3x3), and green the 3D output volume (depth = 2; equal to the number of filters used). The filter W0 is applied on the full depth of the input volume (highlighted in blue). The values are multiplicated elementwise, summed up, and offset with a bias b0. This results in the highlighted green output, which is found in the first slice of the depth stack. When filter W1 is used, its result will be found within the second slice of the output volume depth stack.

3.3.2. Pooling layer

After one or more consecutive convolution layers, a pooling layer is added to reduce the output volume along its width and height dimension. This is done to reduce the number of parameters in the network which consequently also combats overfitting. In this layer, a kernel slides over every feature map and applies a function to its input. This function can be the average or L2-norm but most commonly, the max function is used where only the maximum value over all its seen inputs is retained. Note that the

pooling kernel does not go through the full depth of its input volume, and is instead applied on every feature map separately. The depth dimension of the input volume is therefore not changed.
A pooling layer has no parameters associated with it as it only applies a fixed function. It however consists of two hyperparameters, namely the size F of the kernel and its stride S. Using these hyperparameters together with the input volume V, a slight variation of equation (12) is used to calculate the size of the output volume W:

$$W = \frac{V - F}{S} + 1 \tag{13}$$

A visualisation of how a max-pooling layer works can be seen in figure 3.11.

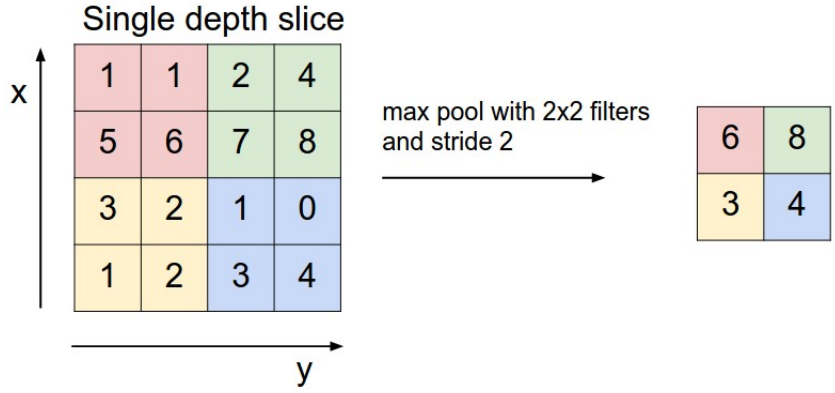


Figure 3.11: Visualisation of application of a max pooling layer onto a single depth splice. (Karpathy *et al.*, 2016). A single depth slice extracted from the input volume with a height and width equal to 4x4 is illustrated on the left. The max-pool kernel of size 2x2 and with stride 2 is applied onto the depth slice. Each colour block indicates an application of the max-pool kernel. This results in the output volume on the right, where the result of each applied kernel operation is visualised by its accompanying colour.

3.3.3. Following layers

A CNN always ends with one or more of the classical fully connected hidden layers. The neurons in these fully connected layers are arranged in a 1D manner, as opposed to the 3D arrangement in the pooling or convolutional layers. To ensure the neuron connections between the last convolutional layer and the first fully connected layer, a flatten layer is added. This layer takes the 3D output from the last convolutional layer and reads it one feature map at a time. While reading a feature map, all values are concatenated, resulting in one big vector. The other feature maps are added to the same vector after the values of the previous maps. This eventually results in a vector with a length equal to the element-wise multiplication of the three dimensions of the output volume of the convolutional layer. The following fully connected layer takes in the vector-output of the flatten layer, and passes it to the next fully connected layers. When the last fully connected layer is reached, a prediction is made by the network and outputted.

4. State of the art

4.1. Splice site prediction

Research into splice site detection and prediction began during the mid to late eighties of the previous century, where the first developed algorithms used PWMs (Staden, 1984), syntactic pattern analysis (Kudo *et al.*, 1987), and homology to consensus splice sequences (Ohshima and Gotoh, 1987) to predict donor splice sites in respectively viruses, mammals and eukaryotes. With an accuracy of 55% or worse and many false positives (FPs), these algorithms failed to accurately predict splicing in test data. In the meantime, NNs were also explored with pioneer work done by Lapedes *et al.* (1988), showing promising results. Their network consisting of one hidden layer (figure 4.1a) was able to correctly identify 85% or more of both acceptor and donor sites in an eukaryotic test dataset with sequences of length 11 bp , a number which increased to 91% if the sequences covered a length of 41 bp. The observation that higher accuracy is reached if the input sequences cover a broader range of nucleotides around the splice site was confirmed by Brunak *et al.* (1991) and Hebsgaard *et al.* (1996). The network developed by Brunak *et al.* (figure 4.1b) was able to reduce the number of false negatives (FNs) by a factor up to 30 in a human dataset, while the NetPlantGene algorithm (Hebsgaard *et al.*, 1996) programmed to identify splice sites in *A. thaliana* had false positive rates (FPRs) lower than 1% if longer input sequences were used.

(a) NN used by Lapedes *et al.* (1988). **(b)** NN used by Brunak *et al.* (1991).

Figure 4.1: Two of the first NNs used for splice site prediction. Both networks are a feed-forward NN with one hidden layer. The input layer takes sequences in one-hot encoding, and the output layer gives back an array of values between 0 and 1, holding the probabilities that the input sequences correspond to a donor (0) or an acceptor (1) splice site.

Characteristic for all these approaches is that the data used was highly skewed with the majority being negative samples, and rarely covered more than 1000 samples for all classes together. Especially NNs rely on a high number of training samples in order to make meaningful predictions, and as other methods were able to perform better on the same amount of data available, focus slowly turned away from NNs to more conventional machine learning methods. An important break-through was made by the GeneSplicer algorithm by Pertea *et al.* (2001). This approach used a combination of a hidden Markov model (HMM) and a maximal dependency tree (MDT) and became the standard to benchmark new algorithms to, as it had less missed true splice sites compared to other models such as NetPlantGene (Hebsgaard *et al.*, 1996) and Genie (Reese *et al.*, 1997), while still maintaining the same FPs rate. The number of sequenced genomes was however growing rapidly (figure 4.2), meaning NNs could theoretically again be competing with the now more popular other machine learning methods. However, no hardware was available at that time that could efficiently handle the fine-tuning of the millions of parameters present in large scale NNs, and NNs were mostly - temporally - abandoned for use in DNA annotation. While there were still several publications using NNs, reported networks were small and used for specific problems such as alternative splicing in humans (Wang and Marín, 2006) and *Aspergillus* (Wang *et al.*,

2009), prediction using secondary structure information (Marashi *et al.*, 2006), and in combination with other approaches such as HMMs (Liu *et al.*, 2007). An alternative to NNs, SVMs, quickly became popular as they too are able to introduce non-linearity when the kernel trick is used. However, SVMs were less computationally intensive and were therefore preferred. Algorithms based on a SVM were able to generalise well on a number of species (Dror *et al.*, 2005; Zhang *et al.*, 2006; Sonnenburg *et al.*, 2007) and became the new norm for splice site prediction at that time.

A turning point in deep learning application came in 2012 with the publication of the AlexNet algorithm (Krizhevsky *et al.*, 2012) to classify images using a CNN consisting of eight layers. In their paper, Krizhevsky *et al.* explained their newly invented dropout method; a method which is nowadays still used as one of the standard approaches to reduce overfitting. Yet the most important development was their use of GPUs to train their model. GPUs are able to process large numbers of data in parallel, and thus offer a significant speed boost in parameter tuning during training (Beam, 2017). This meant that larger models could now be programmed, which led to smaller error rates in predictions. With the still exponentially growing amount of sequence data (figure 4.2), AlexNet prompted a new surge of interest in NNs and initiated their use as a promising golden standard for splice site annotation. Nguyen *et al.* (2016) developed a six layer CNN, taking one-hot encoded input, for annotation of splice sites in primates. The authors reported an average accuracy of 96%, which is a 1.5% absolute increase compared to the previous best achieved average accuracy by Li and Wong (2003) on the same dataset. However, no mention was made of the FPR and the dataset used contained only 3000 samples. Excellent use of the growing amount of DNA data was done by Naito (2018), with their human splice site dataset consisting of over 50000 sequences in total. Their network consists of a mix of a CNN and a RNN and outperformed previous methods utilising the same dataset, proving that the use of NNs can lead to better prediction performance. This conclusion was shared by Zuallaert *et al.* (2018), who managed to improve the false discovery rate (FDR) on plant and human datasets by a relative value of 80.9%. Zuallaert *et al.* also have an online version of their prediction algorithm called SpliceRover, making it one of the most recent splice prediction programs. Other current state of the art online algorithms and their techniques used to identify the splice sites are listed in table 4.1.

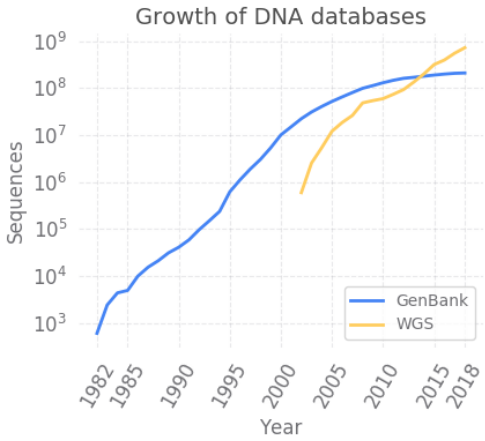


Figure 4.2: Growth of added DNA sequences in both the GenBank and WGS database. The number of bases present in each database doubles approximately every eighteen months (data collected from NCBI, 2018).

4.2. Promoter prediction

Automatic promoter prediction gained traction around the same time as automatic splice site prediction, with prokaryotes as the initial species of interest. Prokaryotic promoters consist of only two elements which are highly conserved and always present in prokaryotic species. This type of promoter thus provided an ideal situation for developing the first promoter recognition algorithms, as it lacks the high diversity typically found in eukaryotic promoters. Similar to splice site recognition, focus was divided between more conventional machine learning techniques and NNs.

One of the first state of the art algorithms was the PROMSEARCH algorithm by Mulligan and Mcclure (1986) which used the base composition of the observed sequence for promoter prediction. It reached

an accuracy of 80.6% on an unseen prokaryotic promoter test set, with only 0.85% of the 49 test instances reported as false positives (Horton and Kanehisa, 1992). It was succeeded by the consensus sequence algorithm of O'Neill and Chiafari (1989), which reached a nearly equal accuracy of 78% in a dataset of 52 *E. coli* sequences. However, both algorithms were heavily outperformed by the NN designed by Demeler and Zhou (1991) as their network with one hidden layer was capable of correctly identifying nearly 100% of the 130 true promoter instances in their *E. coli* dataset.
In the mid nineties, efforts were made to construct a database holding known eukaryotic promoter sequences, resulting in the Eukaryotic Promoter Database (EPD, Cavin Perier *et al.* (1998)). Algorithms focusing on species other than prokaryotic organisms became popular, such as PromFind (Hutchinson, 1996) (vertebrates) and PROMOTER SCAN (Prestridge, 1995) (primates). The algorithms used respectively a differential hexamer frequency approach and a PWM, with reported accuracies of 62.1% and 70%. However, this dropped to 29% and 13% in an independent study conducted by Fickett and Hatzigeorgiou (1997) using 18 previously unpublished mammalian sequences, proving that eukaryotic promoter prediction is a problem of a vastly larger extent than prokaryotic promoter prediction. NN approaches were conducted by Matis *et al.* (1996) and Larsen *et al.* (1995), yielding accuracy rates of respectively 73% and 44.6%. Contrary to splice site prediction, NNs for eukaryotic promoter recognition thus provided no immediate advantage over algorithms implementing other machine learning techniques.

In the following years, little progress was made in improving the accuracy and FPRs for recognition of mammalian eukaryotic promoters. Ohler stated in 2000 that focus needed to be shifted away from the primary DNA sequence onto additional features such as DNA bendability in order to make meaningful progress. This was in strong contrast with prokaryotic promoter prediction and plant promoter prediction, where accuracy rates as high as 97% were reached for bacterial sequences in a dataset of 125 samples using a NN (Kalate *et al.*, 2003), and 82% for a plant dataset with a 166:1 false/positive sample ratio (Shahmuradov *et al.*, 2005). The prediction of human promoters also became popular during that time. Bajic *et al.* (2004) reviewed eight state-of-the-art human promoter prediction software tools on nearly 8000 whole genome sequences. They showed that none of the models reached precision nor recall rates over 65%, although FirstEF was previously reported achieving over 83% precision and recall (Davuluri *et al.*, 2001). Bajic *et al.* assigned this performance gap to the way the 8 models were tested on only chromosome 22, which is an atypical chromosome due to its elevated G+C content, and results could therefore not be extrapolated onto other chromosomes. They concluded that the current state of technology did not allow for accurate promoter predictions on the whole human genome, and that other features aside from the base sequence need to be taken into account as well in order to make improvements. Although 5 of the 8 models under review were NNs, no evidence was found that they perform significantly better in predicting human promoters than other non-NN algorithms.
In the meantime, several online promoter prediction tools were developed that still can be used to this date. Promoter 2.0 (Knudsen, 1999) utilises a NN to predict vertebrate Pol II promoters, but has as limitation that it only accepts 50 sequences at once. The NNPP (Reese, 2001) tool runs a NN to predict eukaryotic promoters. It however still uses the 1999 release of the program, and has only been trained on 565 and 1941 positive promoter sequences of respectively fruitflies and humans. More recent algorithms able to predict promoters are CNNProm (Umarov and Solovyev, 2017), TSSPlant (Shahmuradov *et al.*, 2017), and ElemeNT (Sloutskin *et al.*, 2015). Their target species and technique utilised to develop the model are listed in table 4.1, along with other frequently cited or recently developed PPPs.

A significant problem in the reported articles is the availability and construction of negative promoter samples. This was already highlighted in the early years of promoter prediction, with Demeler and Zhou (1991) achieving an accuracy of nearly 100% on their test set. Demeler and Zhou used randomly generated DNA sequences for their negative samples, inadvertently creating a dataset which consists of two nearly completely distinct classes. An algorithm trained on such a dataset will have little trouble separating the two classes due to their lack of shared features, resulting in biased predictions that seem overly positive, and failing to make accurate generalisations on more intricate datasets where the distinction between the promoters and non-promoters is not as clear. Other approaches used to circumvent this

negative sample problem are selecting sequences from the same species but with a different annotation (Prestridge, 1995), random extraction of coding sequences within the same species (Liu *et al.*, 2018), or sequences from a related organism (Towell, 1993). Gusmao and de Souto (2014) conducted a comparative study for the prediction of promoters in the *E. coli* bacteria with different negative samples constructed using the approaches previously explained. They found that the different negative sample scenarios greatly impact the performance scores of the PPPs, and suggest to assess each scenario by training a new classifier and evaluating them separately or to use a mix of different approaches to construct the final negative promoter dataset.

Table 4.1: State of the art prediction programs for splice sites (A) and promoters (B). The first column specifies the name of the prediction tool, the second the species for which the tool was developed, the third the technique which the underlying algorithm utilises to make its predictions, and the fourth the reference to the article in or site on which the tool is described.

Name	Species	Technique	Reference
A. Splice sites			
FSPLICE	eukaryotes	not specified	http://www.softberry.com
GeneSplicer	*A. thaliana*, human	decision tree, Markov model	Pertea *et al.* (2001)
HSF	human	PWM	Desmet *et al.* (2009)
HSplice	vertebrates	SVM	Meher *et al.* (2016c)
MaLDoSS	human	random forest	Meher *et al.* (2016b)
NetAspGene 1.0	*Aspergillus*	NN	Wang *et al.* (2009)
NetGene2 Server	*A. thaliana*, *C.elegans*, human	NN	Brunak *et al.* (1991); Hebsgaard *et al.* (1996)
NetPlantGene	*A. thaliana*	NN	Hebsgaard *et al.* (1996)
NNSPLICE	fruitfly, human	NN	Reese *et al.* (1997)
PreDOSS	vertebrates	SVM, NN, random forest	Meher *et al.* (2016a)
SPL	*A. thaliana*, *C.elegans*, yeast, fruitfly, human	linear discriminant analysis	Solovyev *et al.* (1994)
SplicePort	human	feature generation	Dogan *et al.* (2007)
SplicePredictor	eukaryotes	bayesian models	Brendel *et al.* (2004)
SpliceRover	*A. thaliana*, human	CNN	Zuallaert *et al.* (2018)
SPLM	human	PWM	Solovyev (2008)
B. Promoters			
70ProPred	prokaryotes	SVM	He *et al.* (2018)
BacPP	*E. coli*	NN	de Avila e Silva *et al.* (2011)
BPROM	bacteria	linear discriminant analysis	Solovyev and Salamov (2010)
CNNProm	*E. coli*, *B. subtilis*, *A. thaliana*, mouse, human	CNN	Umarov and Solovyev (2017)
ElemeNT	eukaryotes	PWM	Sloutskin *et al.* (2015)
EP3	eukaryotes	not specified	Abeel *et al.* (2008)
FPROM	human	linear discriminant analysis	Solovyev *et al.* (2010)
GPMiner	mouse, rat, dog, chimpanzee, human	SVM	Lee *et al.* (2012)
iPromoter-2L	*E. coli*	random forest	Liu *et al.* (2018)
NNPP	prokaryotes, eukaryotes	NN	Reese (2001)
PROMH	eukaryotes	linear discriminant analysis	Solovyev and Shahmuradov (2003)
Promoter 2.0	vertebrates	NN	Knudsen (1999)
PromoterInspector	mammals	not specified	Scherf *et al.* (2000)
PromoterPredict	*E. coli*	PWM	Bharanikumar *et al.* (2018)
TSSG	mammals	linear discriminant analysis	Solovyev *et al.* (2010)
TSSP	plants	linear discriminant analysis	Solovyev *et al.* (2010)
TSSPlant	plants	NN	Shahmuradov *et al.* (2017)
TSSW	human	linear discriminant analysis	Solovyev *et al.* (2010)

5. Materials and methods

5.1. Datasets - Splice sites

5.1.1. *Arabidopsis thaliana*

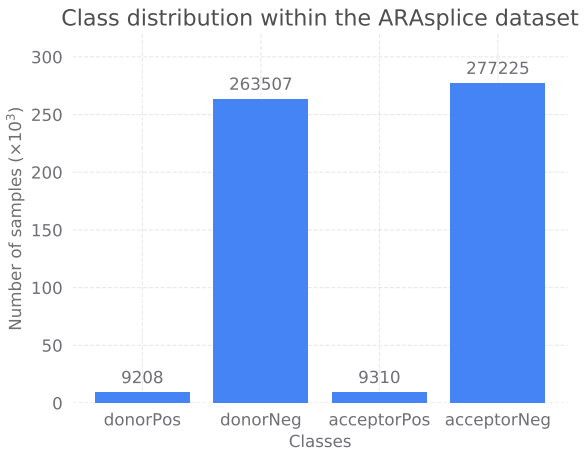


Figure 5.1: Class distribution within the *A. thaliana* splice dataset. The exact number of samples per class is shown on top of the bars.

The *A. thaliana* splice site dataset - referred to as ARAsplice - was acquired through the VIB in the fall of 2016. It contains 559250 sample sequences in four distinct classes, covering 1486 different genes. The four classes consist out of negative pseudo donor sites (donorNeg, 263507 samples), negative pseudo acceptor sites (acceptorNeg, 277225 samples), positive true donor sites (donorPos, 9208 samples), and positive true acceptor sites (acceptorPos, 9310 samples). Both negative sample classes cover all 1486 genes, while the donorPos class consists of samples from 1330 genes and the acceptorPos class from 1334. The data is heavily skewed in favour of the negative samples, with a ratio of 29 negative samples for every single positive sample (figure 5.1). Each sample is 402 bp long, and both true and pseudo donor and acceptor sites are found in the middle of the sequence. Benchmark prediction results exist for ARAsplice due to it being used as a dataset in articles by Degroeve *et al.* (2005) and Zuallaert *et al.* (2018). The construction of the dataset is described in full detail in the former, and can also be found in appendix A.

The difference in characteristics between the sequences from the four splice site classes is visualised in sequence logos in figure 5.2. Here it is seen that all classes share the major consensus sequences GT and AG innate to respectively donor and acceptor splice sites. However, the positive classes have a higher nucleotide conservation around their splice sites than their negative counterparts, showing that true splice sites can possibly be recognised by the presence of these nucleotide regions. Note that the number of genes in the dataset received from the VIB is different from the number of genes reported by Degroeve *et al.* (1486 vs. 1495). This is probably due to a loss of information when the dataset was handled in the VIB. This can be seen in the received dataset where 127169 sequences were given the meaningless gene description of 'fasta', with the vast majority of these sequences belonging to either the donorNeg or the acceptorNeg class. As this number of sequences is nearly one fourth of all samples in the entire dataset, these sequences were not removed and simply assigned 'fasta' as their accompanying gene.

5.2. Datasets - Promoters

Promoter prediction is done for two different species, namely *A. thaliana* and *H. sapiens*. As explained in section 4.2, special attention has to be given to the creation of the negative dataset in order to avoid biased prediction results. To this end, two main approaches for negative sample construction are explored called the balanced and conserved approach. The balanced approach ensures that the number of negative samples is equal to the number of the positive ones. The novel conserved construction method

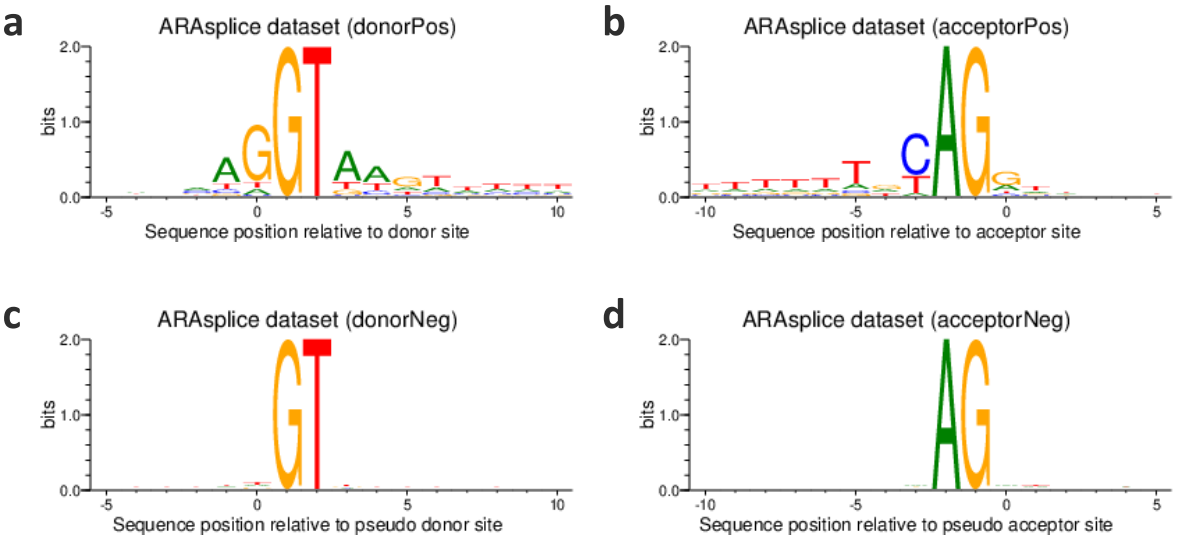


Figure 5.2: Sequence logo around around the (pseudo) donor or acceptor sites within the *A. thaliana* splice site dataset **(a)** True donor sequences (donorPos class). **(b)** True acceptor sequences (acceptorPos class). **(c)** Pseudo donor sequences (donorNeg class). **(d)** Pseudo acceptor sequences (acceptorNeg class).

is equal to the balanced scheme, with the alteration that the most conserved consensus sequences in the positive samples are always retained in the negative samples too. An overview of the resulting four different datasets and their composition is given in table 5.1. In all datasets, the positive samples are referred to as promPos and the negative samples as promNeg.

Table 5.1: Overview of the five different datasets used in promoter prediction. The first column holds the name of the dataset, the second column specifies which species the samples are extracted from, and the third column gives a brief description on the approach used to design the negative samples. The fourth indicates the positive to negative sample ratio.

Name	Species	Construction negative set	Sample ratio
Training sets			
ARAprom balanced	*A. thaliana*	40% similarity to positive samples	1:1
ARAprom conserved	*A. thaliana*	40% similarity to positive samples with conservation of TATA-box and TSS	1:1
HOMprom balanced	*H. sapiens*	40% similarity to positive samples	1:1
HOMprom conserved	*H. sapiens*	40% similarity to positive samples with conservation of TSS	1:1

5.2.1. *Arabidopsis thaliana* positive sample set

The *Arabidopsis thaliana* positive promoter dataset - referred to as ARAprom - was collected from the public EDPnew database (Dreos *et al.*, 2013) in spring 2019. It contains 22703 Pol II promoters from 22701 different genes, covering TATA-boxes and initiator elements as core promoter elements, and GC- and CCAAT-boxes as proximal promoter elements. The extracted sequence length was set to 300 bp, ranging from -249 to +50 bp relative to the TSS. The base composition around the TSS is visualised in **??**a, where two conserved regions can be distinguished. The first region is found between approximately -35 to -25 and makes up the TATA-box found in the core promoter. The second conserved region is the inr, encountered around the TSS's position at +1. The assembly pipeline as done by EDPnew is further elaborated and visualised in appendix B.

5.2.2. *Homo sapiens* positive sample set

Similarly as the ARAprom dataset, the H. sapiens positive promoter dataset or HOMprom set was acquired through the public EDPnew database (Dreos *et al.*, 2013) in spring 2019. It consists of 29598 Pol II promoters from 16455 different coding genes. Each sample contains at least one promoter element from the following list: a TATA-box, an inr, a GC-box, or a CCAAT-box. The sequence length of the downloaded samples was set to 300 bp, ranging from -249 to +50 bp relative to the TSS. The region around the TSS is pictured in **??**b. The sequence composition in human promoters is much more conserved around the TSS than for *A. thaliana* promoters, although the inr around position 0 seems to know more variation in humans. The consensus logo for the TATA-box is also nearly completely absent, indicating a more diverse sequence composition around position -35 to -30 compared to the ARAsplice dataset. The full assembly pipeline as done by EDPnew is explained in appendix B.

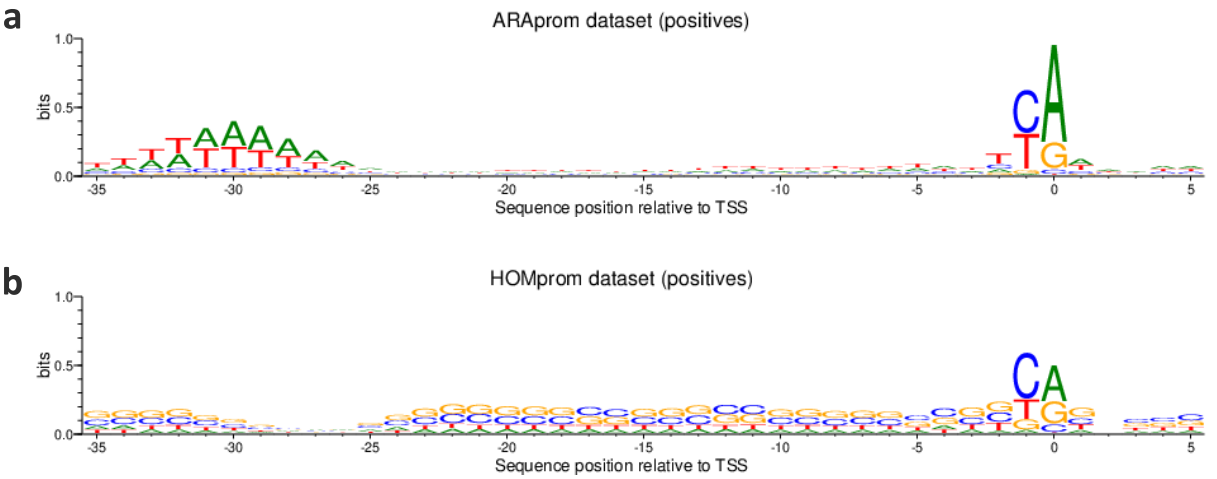


Figure 5.3: Sequence logos around the TSS of the positive samples in the promoter datasets of **(a)** *A. thaliana* and **(b)** *H. sapiens*.

.

5.2.3. Negative sample construction

Two approaches are tested, namely the balanced construction method by Oubounyt *et al.* (2019), and the newly proposed conserved approach. The effect of the two approaches on the base composition of the negative samples is visualised in respectively figure 5.4b and figure 5.4c for the ARAprom dataset. In figure 5.4a, the original base composition of the positive samples is given as a reference to show the differences between the positive and negative samples.

Balanced set

In this approach, the negative samples are constructed according to the method laid out by Oubounyt *et al.* (2019). Every positive sample is split into 20 windows with a length of 15 bp. From these subsequences, the contents of 12 randomly selected windows are replaced by randomly chosen DNA bases. The remaining 8 are not altered in any way in order to preserve a certain level of similarity between the two samples. The 12 new sequences and 8 conserved sequences are then concatenated together in their original order to form a negative sample.

This results in an artificial negative sample for each positive sample, and thus in a completely balanced dataset. As 8 out of 20 windows are conserved between the positive and negative sample, the two samples share approximately 40 percent similarity to each other. The negative sample construction procedure according to Oubounyt *et al.* is illustrated in figure 5.5, and the resulting consensus sequences in figure 5.4b for the ARAprom dataset. Here it is seen that the major consensus sequences present in the positive samples are still retained, although to a much lesser degree.

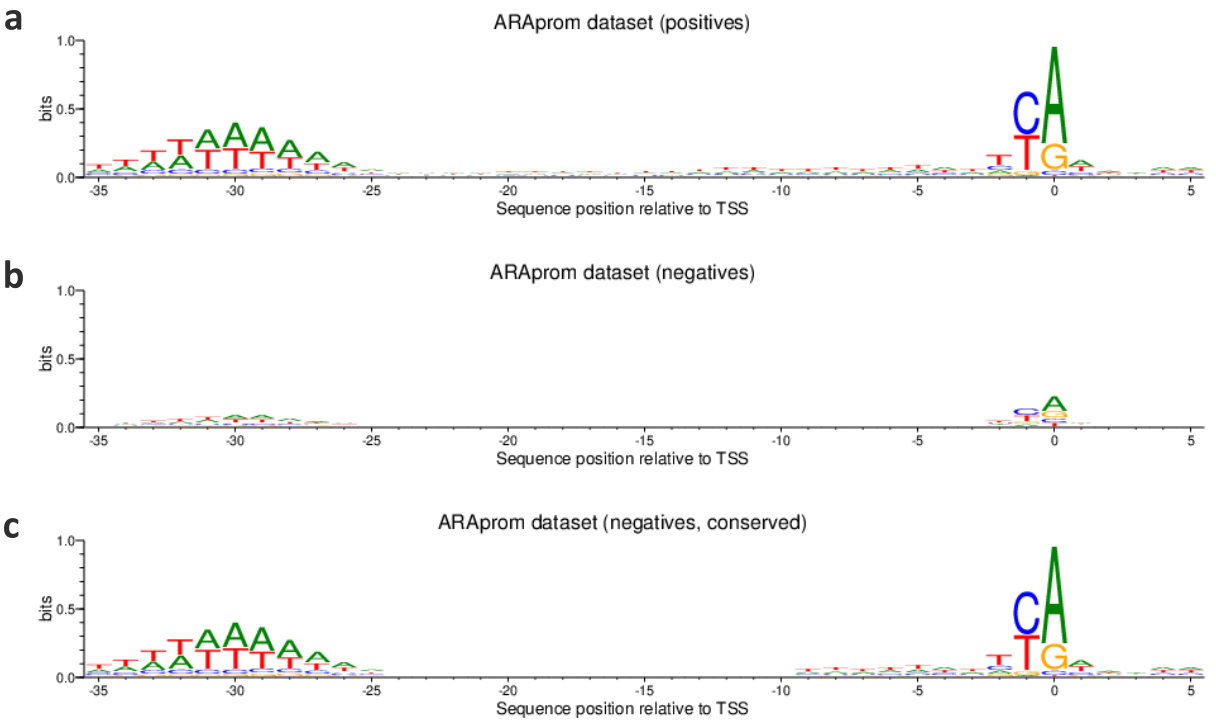


Figure 5.4: Visualisation of the differences between the negative sample construction approaches. **(a)** Positive samples. **(b)** Negative samples with balanced approach. **(c)** Negative samples with conserved approach.

Conserved set

When testing the negative sample method by Oubounyt *et al.*, it became clear that the technique was too liberal to provide a stimulating dataset for the deep learning algorithms. Therefore their method is slightly altered to yield the conserved method. In this approach, the windows with a highly conserved consensus sequence are always picked for incorporation into the negative sample, with the remaining windows being picked in such a manner that an overall similarity percentage of 40% is obtained. This is done so the DNN is forced to use other, less obvious, features to learn the distinction between promoter and non-promoter samples. For the ARAprom set, the windows containing the TATA-box and inr (respectively window number 14 and 16) are conserved in the negative sample, with 6 other windows being randomly selected for conservation from the 18 remaining ones. In the HOMprom dataset, only the inr (window number 16) is conserved across the different samples, resulting in 7 out of 19 remaining windows taken randomly for conservation in the negative sample. A sequence logo of the resulting negative samples is plotted in figure 5.4c for the ARAprom dataset. The TATA-box and inr are conserved in an equal degree between the positive and negative samples. Between these two major consensus sequences, it is seen that no conservation takes place due to the random sampling method used in the construction approach.

5.3. Preprocessing of the data and its labels

5.3.1. Data splitting

Each dataset is split into a training, validation, and test set, with respectively an 80-10-10 percent distribution of the original dataset. Two different data splits have been applied, namely the stratified and the grouped split. The first split focuses on the problem of a large class imbalance, while the second one ensures that no genes are shared over the three different data splits.

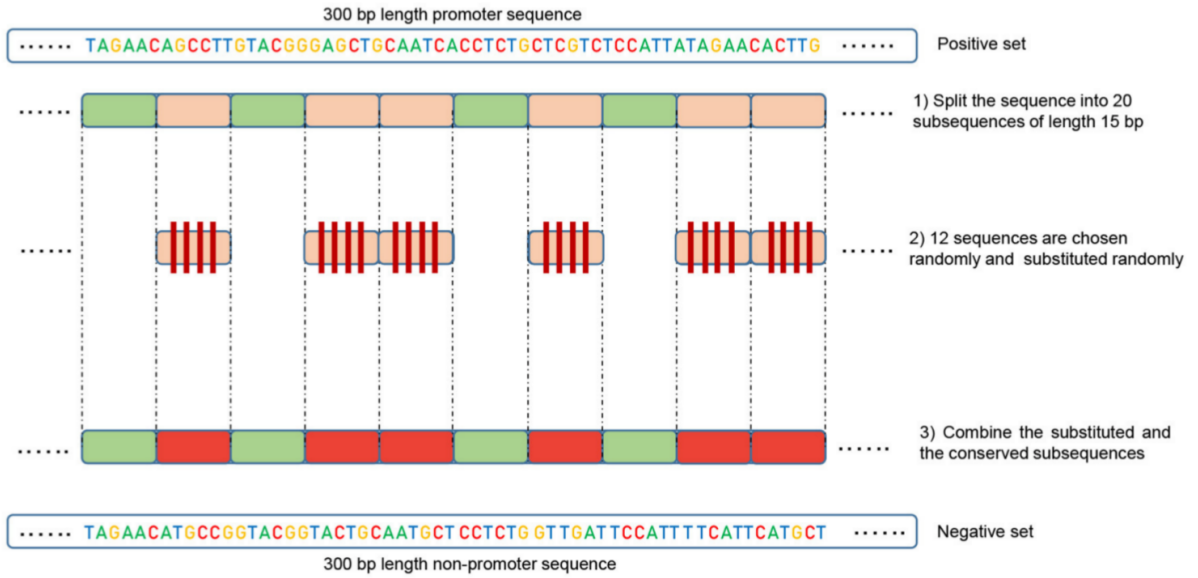


Figure 5.5: Negative promoter sample construction procedure as pictured in Oubounyt *et al.* (2019). Each positive sample is split into 20 windows of length 15 bp . 8 windows are randomly chosen to be conserved (green squares) in the newly constructed negative sample. The remaining 12 windows get substituted by randomly chosen sequences (red squares) and are assembled into the negative sample, yielding a fully constructed negative sample that shares 40% similarity to its accompanying positive one.

Stratified split

The stratified data split ensures that the train, test, and validation splits all have the same class distribution as the original dataset. When a certain class is under-represented in the training set, the network will fail learning to predict this class correctly, which will result in poor test results. If this happens in the validation or test sets, the final results will be overly positive or negative, and will not reflect the true state of the data under observation. By making a stratified split, each class has the same representation in every split and no skewing can take place during the training or testing phase of the deep learning network. It is achieved by applying the `train_test_split` function from the Scikit-learn Python package (Pedregosa *et al.*, 2011).

Grouped split

It is possible that sequences extracted from the same gene share underlying characteristics which the NN could pick up on. If these sequences are distributed over the training and test sets, test results could become biased as the model will have less difficulty predicting these sequences as it has already seen a near similar instance during its training phase. In order to avoid this situation, a grouped split was made. Here the original dataset is split in such a way that no overlapping genes are found between the training, test and validation set. This is done with the use of the `GroupShuffleSplit` function from the Scikit-learn Python package (Pedregosa *et al.*, 2011). This function has no option to stratify the samples, and thus each of the three different splits has a different class distribution compared to each other and the original dataset. Results from this split should therefore be interpreted with caution if the number of covered genes greatly differs from the number of samples.

As no gene information was given for the 'fasta' gene samples in the ARAsplice set, these were removed before the train-test-validation group split was made. This resulted in a removal of 127169 sequences in total - of which 65801, 59650, 859, and 859 and were removed from respectively the acceptorNeg, donorNeg, donorPos, and acceptorPos class - and gave rise to a new class distribution of 1:25 positive to negative samples. The grouped split was not applied on the promoter prediction datasets as the negative sample set partly consist out of randomly created sequences, abandoning the association with a certain gene.

5.3.2. Data augmentation

While creating the promoter prediction program PromoterInspector, Scherf *et al.* (2000) found that a predicted promoter sequence on a DNA strand came with another predicted promoter region on the opposite DNA strand at the same location. This could indicate that DNA consists out of promoter regions instead of orientation-specific promoter sequences tied to only one DNA strand. Because of this implication, a data augmentation scheme is proposed for use in unbalanced datasets where the reverse complementary strand is constructed for every positive sample. This way the class imbalance between positive and negative samples is lessened and the DNN is provided with more learning samples. The augmented samples are aligned to the other samples in the dataset in order to avoid highly similar sequences, again using the Clustal Omega program (version 1.2.4; Sievers and Higgins (2018)). For the ARAsplice set, the augmented samples are aligned to only a randomly chosen subset of the original samples in the dataset, as the available computational resources are not sufficient for alignment with the nearly 600000 sequences contained in the entire dataset.

5.3.3. Data encoding

A NN cannot handle non-numerical data. As DNA consists out of character data, each letter of the DNA alphabet has to be cast to a new, numerical variable. Two encodings are explored to achieve this, namely the commonly used one-hot encoding and the newly proposed k-mer encoding.

One-hot encoding

One-hot encoding is widely spread in the deep learning community as a standard for data encoding due to its simplicity and good performance. Each variable in the data is represented as a binary vector with a length equal to the total number of unique classes in the data, which were all assigned a temporary integer value. Its accompanying one-hot vector is then a vector of all zeros, except with a value of one found at the index of the assigned integer. For use on DNA data, this means that each letter of the genetic alphabet is represented as a 1D row vector of length 4. The exact encoding used in this master dissertation is A = $[0001]$, C = $[0010]$, G = $[0100]$, T = $[1000]$, and unknown nucleotides N = $[0.5\,0.5\,0.5\,0.5]$. A sequence of length l is then represented as a 2D matrix with a shape of $[l \times 4]$, and the entire dataset with n samples as a $[n \times l \times 4]$ matrix. If sequences of differing length are present within the dataset, the length of all sequences is set to the length of the longest sample and gaps are padded with N values.

K-mer encoding

K-mer encoding works by establishing a vocabularium and counting how many times each word in the vocabularium occurs in each sample. This way, every sample is transformed into a 1D numerical array with a length equal to the size of the vocabularium. It can thus transform a dataset where samples have different lengths into a dataset where samples all have an equal length. This is especially interesting in the application of deep learning for the automatic recognition of promoters, as promoters can range from 100 bp to as large as several thousands of bp . Setting the sample length immediately to a large fixed bp length, shorter promoters will be surrounded by non-contributing bases that add noise to the data. With k-mer encoding, every sample can still have its own length. Its use stems from the bag-of-words model used in natural language processing and has to this date not been applied in splice site or promoter prediction with deep learning, to the best of our knowledge.

The vocabularium consists of every possible 'word' or k-mer of at most length k using the four letters of the DNA alphabet. All shorter k-mer lengths are also taken into account, until a length of $k = 1$ is reached. This amounts to a vocabularium with a size v equal to $\sum_{i=1}^{k} 4^i$. After this, every sample in the dataset is split into their respective k-mers. This is done by sliding a window with size k over the sequence, one step at a time, and for every value of k between 1 and its maximum number. For every sample it is then counted how many times each unique k-mer from the global vocabularium occurs, and the count is divided by the length of the sample sequence to account for the imbalance that longer sequences will have more counts for the lower k-mer values. Unknown nucleotides (N) are cast to an

arbitrarily chosen nucleotide before they are counted. All counts are then inserted into a 2D matrix with shape $[n \times v]$, with n the number of samples in the dataset and v the total size of the vocabularium. Row i holds the k-mer counts for the i^{th} sample in the dataset. In figure 5.6, an example is given on how k-mer encoding works on a DNA dataset with samples of varying lengths. To see if k-mer encoding has the potential for being a successful DNA encoding, histograms of the kmer-counts per dataset class have been plotted to see if a difference in distribution is encountered. For every k-mer, the total count is divided by the number of sequences in the class to obtain the k-mer count per sequence. These plots are shown and further discussed in chapter 6.

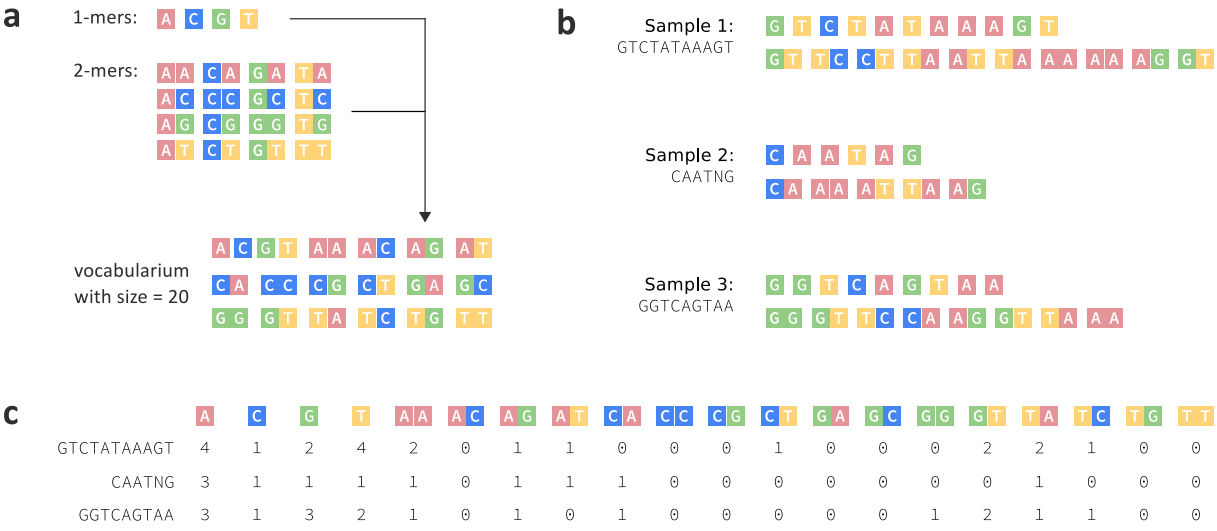


Figure 5.6: Example of k-mer encoding on a DNA dataset with samples of varying length, with $k = 2$. **(a)** Construction of the vocabularium. All possible combinations with the four DNA bases are made, starting from a length of 1 until a length of $k = 2$ is reached. All combinations are then joined together into one global vocabularium of size $v = 20$. **(b)** The DNA dataset holding three samples ($n = 3$) of varying length, and their split into their respective 1- and 2-mers. Unknown nucleotides (N) are cast to an arbitrary chosen nucleotide. **(c)** Table holding the absolute counts for each sample in the dataset and for every k-mer in the vocabularium. The table has a fixed shape of $[3 \times 20]$. The next and final step consists of dividing each row entry by the length of its accompanying sample (not pictured).

As the function $\sum_{i=1}^{k} 4^i$ is an exponential function, the size of the vocabularium will grow quickly with rising values of k. To save computational space, speed up the deep learning process and avoid meaningless features, k-mer encoding is only tested for $k \in [3, 5]$. Also note that k-mer encoded data needs an extra, second dimension if it is to be used with a 1D CNN designed for one-hot encoded data.

5.3.4. Label encoding

Labels from the ARAsplice set are one-hot encoded into 1D arrays of length 4, with [0 0 0 1] = accepterPos, [0 0 1 0] = acceptorNeg, [0 1 0 0] = donorPos, and [1 0 0 0] = donorNeg. For the promoter datasets vectors of length 2 are used, with the non-promoter class cast upon the [0 1], and the promoter class upon [1 0].

5.4. Deep neural networks

Different deep learning architectures are proposed according to the task at hand. For every one of these architectures, the validation loss is monitored and the model's weights are saved for every epoch during the training process. When training ends, the model is reverted back to the state it was in when the lowest validation loss was encountered, ensuring the best possible model is selected for further use. After the model is evaluated on the test set, for each sample it will output a probability vector $\hat{\mathbf{y}} = [p_0, \ldots, p_i, \ldots, p_k]$, with k the number of classes in the dataset and p_i the probability of a sample

belonging to class i. The predicted label of the sample $\hat{y}$ is then set to the corresponding class of the index where the highest value was encountered, or $\hat{y} = \arg\max \hat{\mathbf{y}}$.

5.4.1. Splice site prediction

One-hot encoded data

The first part of splice site prediction consists of establishing the baseline results obtained by Zuallaert *et al.* (2018). To this end their proposed network called SpliceRover is implemented as described in the research article by Zuallaert *et al.*, with the exception that four neurons are used in the output layer instead of two. SpliceRover exists out of five 2D convolutional layers with three 2D max-pooling layers, six dropout layers, one fully connected layer and an output layer, and is designed for use with one-hot encoded data. As 2D convolutional layers take 3D data as input, the one-hot encoded data is expanded with another dimension using the `expand_dims` function from the NumPy Python package (Oliphant, 2006). In order to avoid this expansion of dimensions, a new version of SpliceRover is implemented with 1D convolutional and max-pooling layers, using the same hyperparameters as provided by Zuallaert *et al.*. The networks are respectively referred to as SpliceRover2D and SpliceRover1D, and are both visualised in figure 5.7. The hyperparameters for the used layers in the network and for the network itself are found in table 5.2. For the learning rate, a step decay schedule is used. This means the learning rate is divided by a certain factor each time after an user-specified number of epochs has passed. Changing the learning rate during training helps with faster convergence of the algorithm.

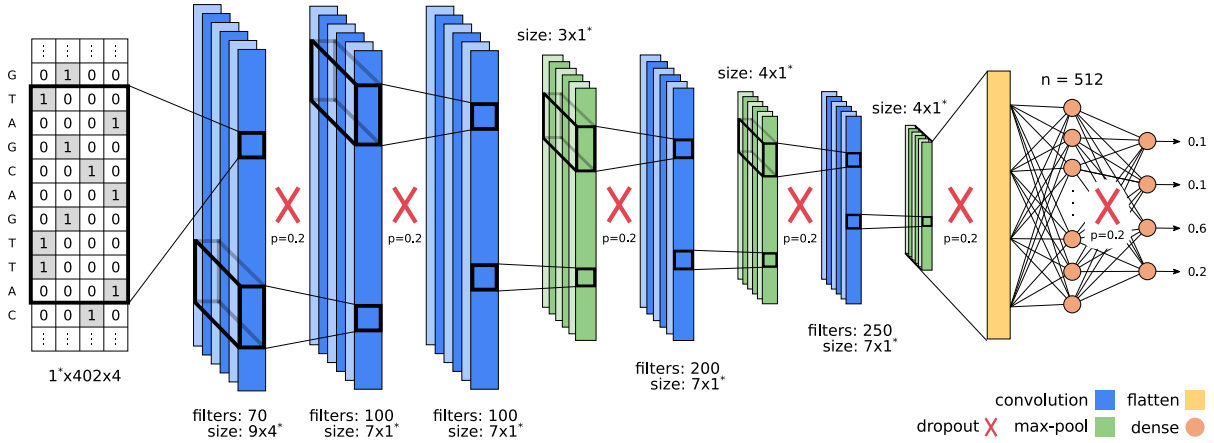


Figure 5.7: Visualisation of the SpliceRover2D model. Values indicated with an '*' are omitted to obtain the SpliceRover1D model. All convolutional and dense layers are activated with a ReLu, except for the output layer which has a softmax activation function.

K-mer encoded data

The second part of splice site prediction consists of evaluating the k-mer encoding. Different network architectures were designed to achieve this, yet none proved sufficient in accurately predicting the k-mer encoded splice data. This is further discussed in section 6.1.2.

5.4.2. Promoter prediction

Four networks are developed for the promoter prediction task, namely one for the different combinations of encoding and species. The networks are named to species promoter sequences they predict, with ARApromnet for *A. thaliana* prediction and HOMpromnet for *H. sapiens* prediction. No different names are used for the different encodings and will always be mentioned in plain text. After training of the HOMpromnet model on both the balanced and conserved *H. sapiens* promoter dataset, the trained models are used to predict the Berkeley Drosophila Genome Project (BDGP, Reese (2001)) *H. sapiens* test dataset.

Table 5.2: Overview of the SpliceRover2D and SpliceRover1D splice site models for use with *A. thaliana* splice site data. The first column specifies which layers are used in consecutive order. The second column gives more information about the parameter settings specific to the used layer, while the third column indicates their used activation function (if applicable). The fourth and fifth column show the hyperparameters and their accompanying value.

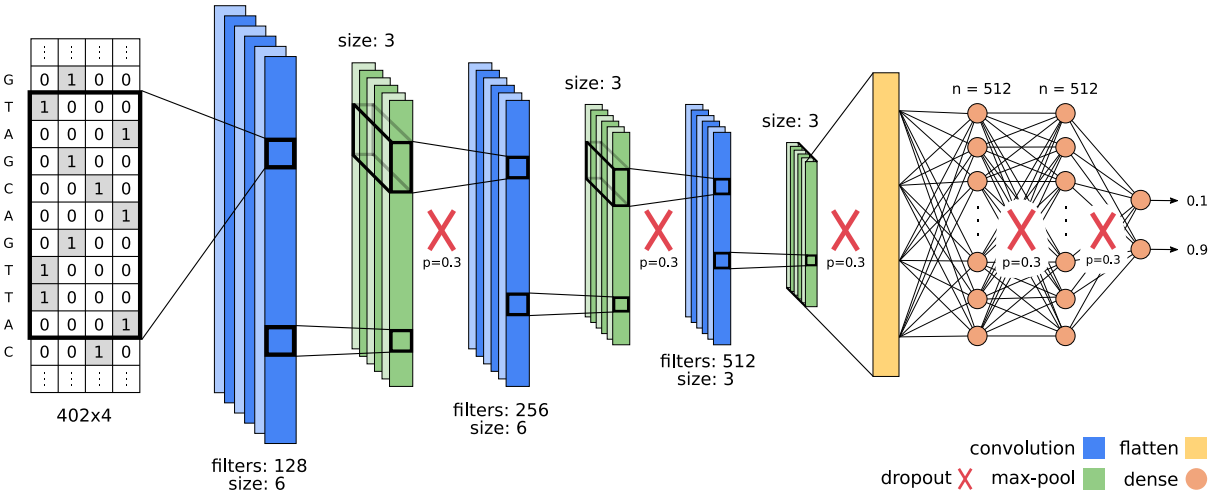
Layer	Details	Activation	Hyperparameter	Value
SpliceRover2D				
convolution 2D	70 $\times$ (9,4)	ReLu	batch size	128
dropout	p = 0.2	-	optimizer	SGD (nesterov TRUE)
convolution 2D	100 $\times$ (7, 1)	ReLu	learning rate	0.05
dropout	p = 0.2	-	learning rate decay	0.5 every 5 epochs
convolution 2D	100 $\times$ (7, 1)	ReLu	epochs	50
max-pool 2D	(3,1)	-	loss	categorical cross-entropy
dropout	p = 0.2	-		
convolution 2D	200 $\times$ (7, 1)	ReLu		
max-pool 2D	(4,1)	-		
dropout	p = 0.2	-		
convolution 2D	250 $\times$ (7, 1)	ReLu		
max-pool 2D	(4,1)	-		
dropout	p = 0.2	-		
fully connected	512 neurons	ReLu		
dropout	p = 0.2	-		
fully connected	4 neurons	softmax		
SpliceRover1D				
convolution 1D	70 $\times$ (9)	ReLu	batch size	128
dropout	p = 0.2	-	optimizer	SGD (nesterov TRUE)
convolution 1D	100 $\times$ (7)	ReLu	learning rate	0.5
dropout	p = 0.2	-	learning rate decay	0.5 every 5 epochs
convolution 1D	100 $\times$ (7)	ReLu	epochs	50
max-pool 1D	(3)	-	loss	categorical cross-entropy
dropout	p = 0.2	-		
convolution 1D	200 $\times$ (7)	ReLu		
max-pool 1D	(4)	-		
dropout	p = 0.2	-		
convolution 1D	250 $\times$ (7)	ReLu		
max-pool 1D	(4)	-		
dropout	p = 0.2	-		
fully connected	512 neurons	ReLu		
dropout	p = 0.2	-		
fully connected	4 neurons	softmax		

One-hot encoded data

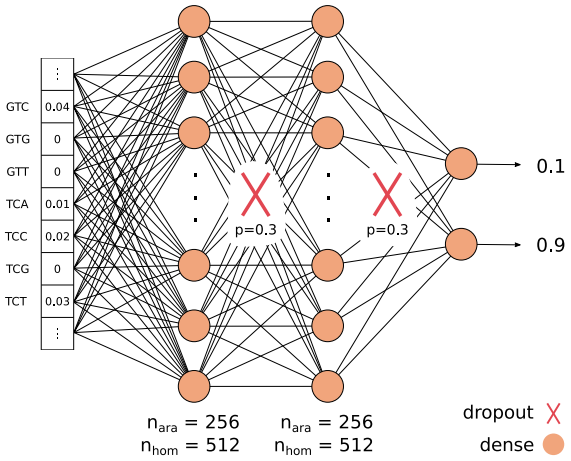
The same network architecture is used for predicting the one-hot encoded promoter sequences in the *A. thaliana* and *H. sapiens* promoter datasets, although with slightly different hyperparameter settings. They consist of three consecutive series of a convolutional layer, a max-pooling layer, and a dropout layer. After the last dropout layer, two dense layers and an output layer are added, where each one of the dense layers is followed by a dropout layer. The global network architecture is pictured in figure 5.8a. More details about the layers and used hyperparameters are found in table 5.3.

K-mer encoded data

Both the ARApromnet and HOMpromnet networks for k-mer encoded data consist of 2 fully-connected layers with respectively 256 and 512 neurons, and one output layer. The networks are visualised in figure 5.8, and more info on their layers and hyperparameters is given in table 5.3.



(a) Promnet for one-hot encoded data. All convolutional and dense layers are activated with a ReLu, except for the output layer which has a softmax activation function.



(b) Promnet for k-mer encoded data. All layers are activated with a ReLu, except for the output layer which has a softmax output function. The subscripts 'ara' and 'hom' indicate the layer-specific hyperparameters settings for use with respectively the *A. thaliana* and *H. sapiens* promoter datasets.

Figure 5.8: Visualisation of the promnet architectures for use with the one-hot and k-mer encoded promoter data.

Table 5.3: Overview of the ARApromnet and HOMpromnet models for use with respectively *A. thaliana* and *H. sapiens* promoter data. The first column specifies which layers are used in consecutive order. The second column gives more information about the parameter settings specific to the used layer, while the third column indicates their used activation function (if applicable). The fourth and fifth column show the hyperparameters and their accompanying value.

Layer	Details	Activation	Hyperparameter	Value
ARApromnet (onehot)				
convolution 1D	128 $\times$(6)	ReLu	batch size	128
max-pool 1D	(3)	-	optimizer	adam
dropout	p = 0.3	-	learning rate	0.001
convolution 1D	256 $\times$(6)	ReLu	epochs	30
max-pool 1D	(3)	-	loss	categorical cross-entropy
dropout	p = 0.3	-		
convolution 1D	512 $\times$(3)	ReLu		
max-pool 1D	(3)	-		
dropout	p = 0.3	-		
dense	512 neurons	ReLu		
dropout	p = 0.3	-		
dense	512 neurons	ReLu		
dropout	p = 0.3	-		
dense	2 neurons	softmax		
HOMpromnet (onehot)				
convolution 1D	128 $\times$(6)	ReLu	batch size	128
max-pool 1D	(3)	-	optimizer	adam
dropout	p = 0.3	-	learning rate	0.0005
convolution 1D	256 $\times$(6)	ReLu	learning rate decay	0.5 every 5 epochs
max-pool 1D	(3)	-	epochs	50
dropout	p = 0.3	-	loss	categorical cross-entropy
convolution 1D	512 $\times$(3)	ReLu		
max-pool 1D	(3)	-		
dropout	p = 0.3	-		
dense	512 neurons	ReLu		
dropout	p = 0.3	-		
dense	512 neurons	ReLu		
dropout	p = 0.3	-		
dense	2 neurons	softmax		
ARApromnet (kmer)				
dense	256 neurons	ReLu	batch size	128
dropout	p = 0.3	-	optimizer	adam
dense	256 neurons	-	learning rate	0.001
dropout	p = 0.3	ReLu	epochs	30
dense	2 neurons	softmax	loss	categorical cross-entropy
HOMpromnet (kmer)				
dense	512 neurons	ReLu	batch size	128
dropout	p = 0.3	-	optimizer	adam
dense	512 neurons	-	learning rate	0.0005
dropout	p = 0.3	ReLu	epochs	50
dense	2 neurons	softmax	loss	categorical cross-entropy

5.4.3. Performance measures

Three performance measures are used to evaluate a proposed DNN, namely precision (Pr), recall (Re), and the false discovery rate (FDR). Precision is the proportion between the true number of samples belonging to a certain class and the predicted number of samples of that class, revealing how many of the retrieved results are truly relevant. Recall indicates how many samples belonging to a certain class are correctly predicted by the model, and thus illustrates how good the model is at retrieving that class. FDR specifies the percentage of wrongly identified samples over all samples predicted as belonging to a certain class. All three measures are calculated separately for each class in the test set, using the multiclass confusion matrix. A generalised confusion matrix for classification with k classes is visualised in figure 5.9. Using a one vs. all approach, precision Pr_i, recall Re_i and false discovery rate FDR_i for a class i are then calculated by applying following formulas:

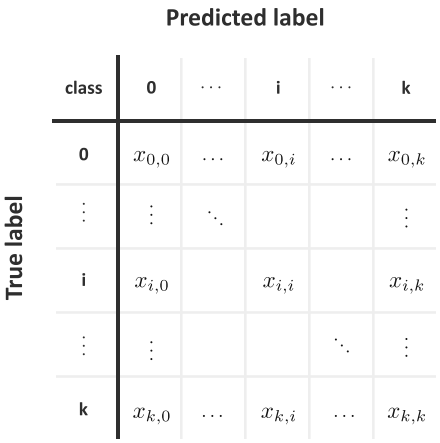


Figure 5.9: Generalised confusion matrix for multiclass classification with k classes.

$$\text{Pr}_i = \frac{\text{TP}_i}{\text{TP}_i + \text{FP}_i} = \frac{x_{i,i}}{\sum_{j=0}^{k} x_{j,i}} \qquad \text{Re}_i = \frac{\text{TP}_i}{\text{TP}_i + \text{FN}_i} = \frac{x_{i,i}}{\sum_{j=0}^{k} x_{i,j}} \qquad \text{FDR}_i = \frac{\text{FP}_i}{\text{TP}_i + \text{FP}_i} = \frac{\sum_{\substack{j=0 \\ j \neq i}}^{k} x_{j,i}}{\sum_{j=0}^{k} x_{j,i}} \tag{14, 15, 16}$$

with TP_i the number of samples of class i that are correctly predicted as class i, FP_i the number of samples wrongly predicted as class i, FN_i the number of samples of class i that were missed by the model, and x the confusion matrix. The F_1 score is also calculated in order to obtain one single value to compare the different trained algorithms. It takes into account both precision and recall by calculating their harmonic average, providing a single value to evaluate both. It lies between 0 and 1, with 1 indicating a perfect precision and recall. The F_1 score for class i (F1_i) is determined by:

$$\text{F1}_i = 2 \cdot \frac{\text{Pr}_i \cdot \text{Re}_i}{\text{Pr}_i + \text{Re}_i} \tag{17}$$

Class-specific precision-recall curves (PR-curves) are also plotted. To achieve this, a binary situation is created for each class i where $\hat{\mathbf{y}} = [p_i, 1 - p_i]$, with p_i the probability of the sample belonging to class i and $1 - p_i$ the probability of the sample belonging to any of the other classes. A probability threshold t is then applied onto $\hat{\mathbf{y}}$, where the sample is predicted as class i if $p_i \geq t$, and the corresponding precision and recall values are calculated. This is done for a range of thresholds between 0 and 1, yielding a series of recall and precision values. These series are plotted against each other in order to obtain the PR-curve for each class.

5.5. Post-processing of the results

Results are tested for homoscedasticity using the Bartlett's test. Student's t-tests are conducted between the various approaches in order to determine if the results differ significantly, with an α level of 0.05.

5.6. Soft- and hardware

All coding is done using the Python programming language (www.python.org, version 3.5.2), utilising the Keras (Chollet, 2015) and Tensorflow (Abadi *et al.*, 2015) frameworks for deep learning. To allow the use of a GPU to speed up the training progress, CUDA is used (version V9.0.176). Other versions of the Python and deep learning packages used can be found in appendix C.
Sequence logos are made using the command line interface of the WebLogo application (Crooks *et al.*, 2004), version 3.6.0. For constructing the logos for the *A. thaliana*, the following base composition is applied: A: 27, C: 18, G: 21, T: 34. For the HOMprom dataset, the *H. sapiens* base composition incorporated in WebLogo is used. An explanation on how to interpret sequence logos is given in appendix D.

All preprocessing of the data, except the similar samples control, was done on a laptop with 8 GB RAM and a quad core i5-6200U Intel Core CPU, clocked at 2.30 GHz with hyperthreading. The similar samples control, and deep NN training and testing was done on a GPU server with 32 GB RAM, a twelve core i7-3930K Intel Core CPU clocked at 3.20 GHz with hyperthreading, and a single NVIDIA Tesla K40c GPU.

6. Results and discussion

6.1. Splice site prediction

6.1.1. One-hot encoding: establishing baseline results by Zuallaert et al.

Both the SpliceRover2D and SpliceRover1D networks are run on the stratified and grouped split, and this for both the original ARAsplice dataset and the augmented one. The performance measures for the positive classes are found in table 6.1.

Table 6.1: Performance measures (PM) on the original and augmented one-hot encoded ARAsplice dataset, for both the SpliceRover2D and SpliceRover1D model. The reported donor and acceptor values are reached for respectively the donorPos and acceptorPos classes.

ARAsplice	PM	SpliceRover2D				SpliceRover1D			
		Stratified		Grouped		Stratified		Grouped	
		Donor	Acceptor	Donor	Acceptor	Donor	Acceptor	Donor	Acceptor
Original	**Pr**	0.872	0.860	0.873	0.852	0.858	0.880	0.871	0.858
	Re	0.939	0.931	0.930	0.929	0.939	0.905	0.902	0.928
	F1	0.904	0.894	0.901	0.889	0.897	0.893	0.902	0.892
	FDR	0.128	0.140	0.127	0.148	0.142	0.120	0.129	0.142
Augmented	**Pr**	0.874	0.895	0.886	0.915	0.880	0.883	0.890	0.875
	Re	0.910	0.832	0.942	0.877	0.924	0.859	0.933	0.907
	F1	0.891	0.863	0.913	0.895	0.901	0.859	0.911	0.891
	FDR	0.126	0.105	0.114	0.0854	0.120	0.117	0.110	0.125

Zuallaert *et al.* report precision values of 0.956 and 0.939 for respectively the positive donor and acceptor class, and a FDR of respectively 0.044 and 0.061. For both classes, the recall values are 95%, resulting in F_1 values of respectively 0.953 and 0.944. These results are obtained by dividing the ARAsplice set into two new datasets, where one holds the positive and negative donor classes and the other the positive and negative acceptor classes. The SpliceRover2D model is then trained separately on these two datasets, yielding a model designed for donor site prediction and a model designed for acceptor site prediction. The benchmark results by Zuallaert *et al.* are not reached by any of the self-implemented methods listed in table 6.1. The main reason for this is that the SpliceRover models trained in this dissertation are trained on the full ARAsplice dataset instead of on two distinct subsets. While this has the advantage that only one model has to be trained and thus less time is spent training, it also results in a severe drop in performance measures ($p < 0.001$ for all performance measures of the positive donors and acceptors). Another reason for this performance difference can be due to the data splits used in this dissertation. In the original SpliceRover article, the data split strategy is not mentioned and thus probably a random split was utilised. This can result in different class distributions in each one of the splits, leading to biased prediction results. The reported result can stem from a split where the train and test set contain respectively a high and a low number of positive class samples. The network will then be optimally trained on the - typically heavily undersampled - positive samples, resulting in high prediction rates for the test set where little positive samples are encountered. A solution to increase the credibility is to perform k-fold CV or multiple runs where each time the data is split in a different way. However, this is not feasible with most deep learning approaches due to their high training time when large datasets are used.

Degroeve *et al.* implemented a SVM to predict the splice sites in the ARAsplice data, with which they obtained 0.80 and 0.68 as precision values for respectively the positive donor and acceptor class, along

with a recall value of 0.95 for both classes. Their FDRs are respectively 0.20 and 0.32. The precision and FDR benchmarks by Degroeve *et al.* are all exceeded by the methods tested in table 6.1. Together with the results obtained by Zuallaert *et al.*, it is clear that DNNs have the ability to significantly outperform conventional machine learning methods when it comes to splice site prediction, due to their natural ability to autonomously extract features and learn from them. However, the recall value of 0.95 is not obtained by any of the SpliceRover models trained. This is an effect of the precision-recall trade-off, visualised in figure 6.1 using PR-curves for the SpliceRover2D model trained on the original ARAsplice set with a stratified split. While these PR-curves differ between models, a trend seen over all the methods tested is that the PR-curves show a larger drop for the positive classes than the negative ones, meaning that an increasing precision value results in a decreasing recall value and vice versa for the donorPos and acceptorPos classes. While Degroeve *et al.* obtained high recall values for both the positive classes, it resulted in deteriorating precision values. Their developed splice site recognition model will therefore be able to identify the true positive samples in a highly precise manner, yet the predicted positive samples will suffer from a high amount of incorrectly identified samples. In order to find a balance between precision and recall, a maximum for both can be derived. This is done by finding the intersect of the $y = x$ line and the PR-curve. In figure 6.1, the joint maximum is found at around 90% for both performance measures, with the donorPos class having a slightly higher rate than the acceptorPos class. The highest achievable F_1 for the SpliceRover models trained in this dissertation will therefore lie around 0.9, a result which is reflected in the values shown in table 5.2.

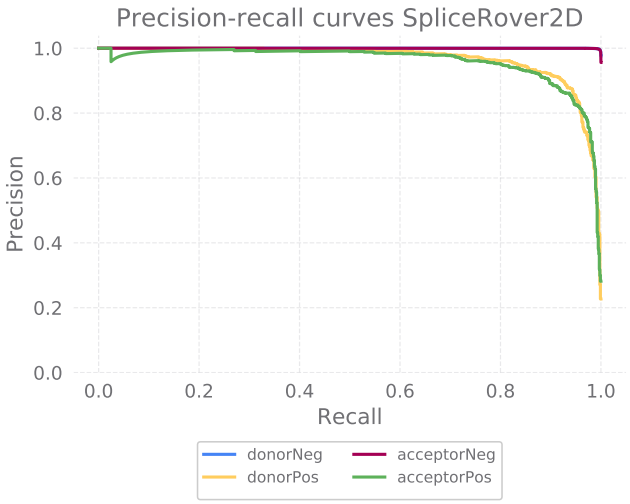


Figure 6.1: Class-specific PR-curves for the test set results from the SpliceRover2D model trained on the original ARAsplice set with a stratified split.

In both the original and augmented ARAsplice runs, a trend is seen where the precision and recall values - and therefore also the F_1 score - are worse in the positive acceptor class ($p = 0.0106$ for F1) than in the positive donor class. This is due to acceptor splice site prediction being a harder problem than donor splice site prediction, something which was also remarked by Hebsgaard *et al.* (1996). Inside the cell, different mechanisms are in place to recognise the specific splice sites. A donor site is recognised by one single substructure of the spliceosome, while a series of substructures are needed to identify the acceptor site (Matera and Wang, 2014). The recognition of the donor site is also the first step in initiating splicing and thus crucial for it to take place, while the acceptor site is only identified later on in the process. Due to these differences, a donor site will have a stronger conserved consensus sequence than an acceptor site, which leads to donors having more easily identifiable features than acceptors. While the FDR on the original ARAsplice set follows the same trend as the precision, recall, and F_1 scores, an opposite trend is noticed in the augmented ARAsplice set. This trend is however not found to be significant ($p = 0.433$).

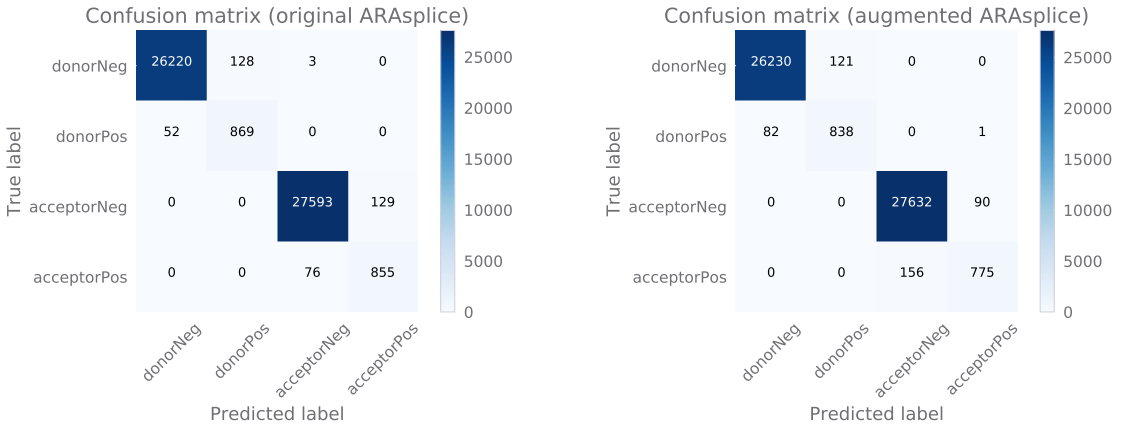
To derive if the different proposed strategies differ significantly from each other, t-tests are conducted between the original and augmented run, the stratified and grouped split, and the 2D and 1D model. The results from these tests are found in table 6.2. Significant differences are reported only for the first comparison between the original and augmented ARAsplice set, with the augmented run resulting in overall better performance measures. The statistically most powerful effects are seen in the precision value and FDR when all positive samples are evaluated together, with a relative increase of 2.7% in the former and a relative decrease of 16.3% in the latter. Slightly less significant effects are detected when

the positive classes are treated separately, with a precision gain of 1.6% and 3.4% and a FDR reduction of 10.6 and 21.7% for respectively the donorPos and acceptorPos class. The proposed augmentation method is thus a viable addition to splice site datasets that suffer from a large class imbalance, especially when it comes to decreasing the number of FPs in the predictions. Specifically interesting is that the decrease in FP results in the acceptor class is twice the value of the donor class. Seeing as acceptor sites pose a more difficult classification problem than donor sites, the reverse complement DNA augmentation can be used to even out the difference in performance measures between the two classes. Caution is however advised when augmenting the data as it can result in a worse recall rate for the positive acceptor class, where a significant relative decrease is seen of 5.9%. This recall drop is due to a smaller number of true positives (TPs) and larger number of FNs when SpliceRover is run with the augmented dataset. This is illustrated in the confusion matrices given in figure 6.2 (absolute numbers))and figure 6.3b (normalised). While the number of TPs drops (885 vs. 775), no decrease is seen in the precision due to compensation by the smaller number of FPs (129 vs. 91). However, recall suffers from the rise in FN predictions (76 vs. 156), resulting in a decreasing recall value compared to the original dataset run. This effect can again be contributed to the difference in recognition mechanisms between donor and acceptor splice sites. When the data are augmented, new 'consensus' sequences are introduced to the positive samples. As donors have stronger conserved consensus sequences, they have more robust features which the network can rely on to be consistently present within a positive donor sample. The newly added consensus sequences - which can potentially add noise - have thus little effect on the donor data due to their inherent robustness. This is not the case for the positive acceptor samples, where the data augmentation will introduce even more consensus sequences to an already great variety, leading to confusion of the network which features to focus on in order to correctly predict the acceptor class. Note that the recall drop is not seen when both positive classes are evaluated together, and thus the proposed reverse complement augmentation method only becomes a problem when one is specifically interested in a high recall for the true acceptor splice sites.

No significant differences are found between the runs with the SpliceRover2D model and the ones done with SpliceRover1D. This is no surprise as the first layer in the original SpliceRover network by Zuallaert *et al.* calculates the convolution over the full length of the one-hot encoding which results in one single value. This is essentially the same as what a 1D convolutional layer does. The first layer from both the SpliceRover2D and SpliceRover1D model is visualised in figure 6.4. Here it can be seen that the outputs of the two layers share the same dimensions, with the output from the 2D convolution having an extra dimension of length 1. This is however the extra dimension that was added to make the data 3D, which does not contain any additional information. While no significant differences are found in the predictions outputted by the two models, the SpliceRover1D model has the advantage that it takes less time to train and execute as a 1D convolution is an easier calculation than a 2D convolution and thus requires less computational resources. Due to this observation, the SpliceRover2D model is abandoned for further research purposes in this dissertation.
When the stratified and grouped split are compared, no significant variations are seen between the results from the two groups either. This indicates no data leakage takes place in any of the two different splits, and samples do not share underlying characteristics if they belong to the same gene. This is due to splicing being necessary for a gene to be expressed as a functional molecule. It is thus a crucial process in maintaining life, resulting in the splice patterns being highly conserved across genes. The grouped split is therefore discarded as well for use with other experiments in this dissertation.

Table 6.2: P-values and mean $\pm$ SD for the different ARAsplice SpliceRover runs comparisons. The first column indicates which two groups are compared to each other. The second column holds the performance measures (PM), and the third their accompanying p-value and the mean $\pm$ SD for the two groups within the comparison. Each comparison is done for both positive classes together (fourth column), and the two classes separately (fifth and sixth column). Significant p-values are highlighted in gray.

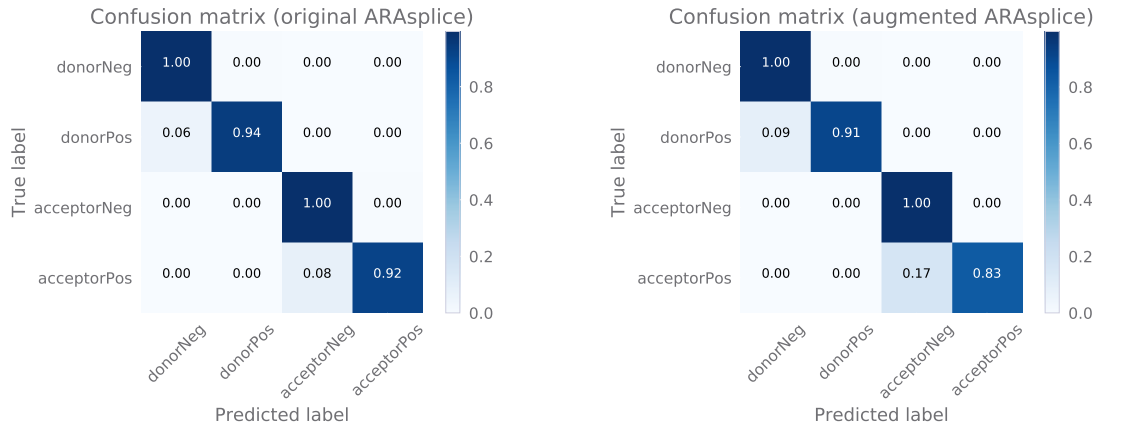
	PM		Donor + Acceptor	Donor	Acceptor
Original vs. augmented	**Pr**	*p*	0.00222	0.0304	0.0320
		original	$0.864 \pm 9.11 \cdot 10^{-3}$	$0.869 \pm 6.10 \cdot 10^{-3}$	$0.863 \pm 1.05 \cdot 10^{-2}$
		augmented	$0.887 \pm 1.24 \cdot 10^{-2}$	$0.883 \pm 6.61 \cdot 10^{-3}$	$0.892 \pm 1.51 \cdot 10^{-2}$
	Re	*p*	0.0923	0.983	0.0180
		original	$0.925 \pm 1.13 \cdot 10^{-2}$	$0.928 \pm 1.15 \cdot 10^{-2}$	$0.923 \pm 1.11 \cdot 10^{-2}$
		augmented	$0.898 \pm 3.60 \cdot 10^{-2}$	$0.927 \pm 1.18 \cdot 10^{-2}$	$0.869 \pm 2.73 \cdot 10^{-2}$
	F1	*p*	0.437	0.590	0.205
		original	$0.896 \pm 5.02 \cdot 10^{-3}$	$0.901 \pm 2.55 \cdot 10^{-3}$	$0.892 \pm 1.87 \cdot 10^{-3}$
		augmented	$0.891 \pm 1.87 \cdot 10^{-2}$	$0.904 \pm 8.77 \cdot 10^{-3}$	$0.877 \pm 1.16 \cdot 10^{-2}$
	FDR	*p*	0.00216	0.0304	0.0316
		original	$0.135 \pm 9.11 \cdot 10^{-3}$	$0.132 \pm 6.10 \cdot 10^{-3}$	$0.138 \pm 1.05 \cdot 10^{-2}$
		augmented	$0.113 \pm 1.23 \cdot 10^{-2}$	$0.118 \pm 6.06 \cdot 10^{-3}$	$0.108 \pm 1.49 \cdot 10^{-2}$
Stratified vs. grouped	**Pr**	*p*	0.788	0.223	0.787
		stratified	$0.875 \pm 1.11 \cdot 10^{-2}$	$0.874 \pm 8.06 \cdot 10^{-3}$	$0.880 \pm 1.26 \cdot 10^{-2}$
		grouped	$0.878 \pm 1.84 \cdot 10^{-2}$	$0.880 \pm 8.15 \cdot 10^{-3}$	$0.875 \pm 2.46 \cdot 10^{-2}$
	Re	*p*	0.416	0.914	0.312
		stratified	$0.928 \pm 1.20 \cdot 10^{-2}$	$0.928 \pm 1.20 \cdot 10^{-2}$	$0.883 \pm 3.67 \cdot 10^{-2}$
		grouped	$0.927 \pm 2.00 \cdot 10^{-2}$	$0.927 \pm 1.50 \cdot 10^{-2}$	$0.910 \pm 2.11 \cdot 10^{-2}$
	F1	*p*	0.115	0.087	0.221
		stratified	$0.888 \pm 1.60 \cdot 10^{-2}$	$0.898 \pm 4.87 \cdot 10^{-3}$	$0.877 \pm 1.63 \cdot 10^{-2}$
		grouped	$0.899 \pm 8.53 \cdot 10^{-3}$	$0.907 \pm 5.31 \cdot 10^{-3}$	$0.892 \pm 2.17 \cdot 10^{-3}$
	FDR	*p*	0.792	0.223	0.782
		stratified	$0.125 \pm 1.14 \cdot 10^{-2}$	$0.129 \pm 8.06 \cdot 10^{-3}$	$0.121 \pm 1.26 \cdot 10^{-2}$
		grouped	$0.123 \pm 1.84 \cdot 10^{-2}$	$0.120 \pm 8.15 \cdot 10^{-3}$	$0.125 \pm 2.44 \cdot 10^{-2}$
2D vs. 1D	**Pr**	*p*	0.632	0.849	0.696
		2D	$0.878 \pm 1.87 \cdot 10^{-2}$	$0.876 \pm 5.67 \cdot 10^{-3}$	$0.881 \pm 2.57 \cdot 10^{-2}$
		1D	$0.874 \pm 1.08 \cdot 10^{-2}$	$0.874 \pm 1.18 \cdot 10^{-2}$	$0.874 \pm 9.67 \cdot 10^{-3}$
	Re	*p*	0.958	0.615	0.796
		2D	$0.911 \pm 3.58 \cdot 10^{-2}$	$0.930 \pm 1.25 \cdot 10^{-2}$	$0.895 \pm 4.10 \cdot 10^{-2}$
		1D	$0.912 \pm 2.39 \cdot 10^{-2}$	$0.924 \pm 1.40 \cdot 10^{-2}$	$0.900 \pm 2.52 \cdot 10^{-2}$
	F1	*p*	0.948	0.929	0.898
		2D	$0.894 \pm 1.37 \cdot 10^{-2}$	$0.902 \pm 7.85 \cdot 10^{-3}$	$0.885 \pm 1.30 \cdot 10^{-2}$
		1D	$0.893 \pm 1.43 \cdot 10^{-2}$	$0.903 \pm 5.12 \cdot 10^{-3}$	$0.884 \pm 1.43 \cdot 10^{-2}$
	FDR	*p*	0.506	0.849	0.557
		2D	$0.118 \pm 2.59 \cdot 10^{-2}$	$0.124 \pm 5.67 \cdot 10^{-3}$	$0.113 \pm 3.54 \cdot 10^{-2}$
		1D	$0.126 \pm 1.08 \cdot 10^{-2}$	$0.125 \pm 1.18 \cdot 10^{-2}$	$0.126 \pm 9.67 \cdot 10^{-3}$



(a) Original ARAsplice

(b) Augmented ARAsplice

Figure 6.2: Confusion matrices of the SpliceRover2D model run on the stratified split of both the original and augmented ARAsplice set.



(a) Original ARAsplice

(b) Augmented ARAsplice

Figure 6.3: Normalised confusion matrices of the SpliceRover2D model run on the stratified split of both the original and augmented ARAsplice set.

conv2d_1: Conv2D	input:	(None, 1, 402, 4)
	output:	(None, 70, 394, 1)

(a) SpliceRover2D

conv1d_1: Conv1D	input:	(None, 402, 4)
	output:	(None, 394, 70)

(b) SpliceRover1D

Figure 6.4: First layer from both the SpliceRover2D and SpliceRover1D model, as visualised by Keras.

The negative classes - donorNeg and acceptorNeg - have similar performance results over all the different model runs, with $0.997 \pm 4.84 \cdot 10^{-4}$, $0.995 \pm 6.61 \cdot 10^{-4}$, and 0.996 ± 0 for respectively the precision, recall, and F_1 score for the donorNeg class, and $0.996 \pm 1.13 \cdot 10^{-3}$, $0.996 \pm 1.12 \cdot 10^{-3}$, and $0.996 \pm 4.84 \cdot 10^{-4}$ for respectively the precision, recall, and F_1 score for the acceptorNeg class. These values indicate that both SpliceRover models experience no problems in correctly predicting the negative classes. This was already remarked in section 5.1.1, where figure 5.2 shows significant differences in DNA sequence between the positive and negative classes, with no preservation of consensus sequences around the pseudo splice sites for the latter.
The obtained FDR for the donorNeg class is $0.00258 \pm 3.26 \cdot 10^{-4}$. For the FDR of the acceptorNeg class, a significant difference is found between the SpliceRover models run on the original ARAsplice set and the augmented one ($p = 0.0049$), with respectively FDRs of $0.00290 \pm 2.85 \cdot 10^{-4}$ and $0.00391 \pm 1.16 \cdot 10^{-3}$. The augmented ARAsplice dataset thus introduces more FPs in the acceptorNeg class than the original set, which is due to the rise in FNs in the acceptorPos class (see also figure 6.2b). The acceptorPos FN results are all wrongly categorised into the acceptorNeg class, thus inflating the number of FPs in this class. This rise in FDR is however no reason for concern, as the reported value is less than 1% and one will most commonly only be interested in correctly predicting the positive classes.

6.1.2. K-mer encoding

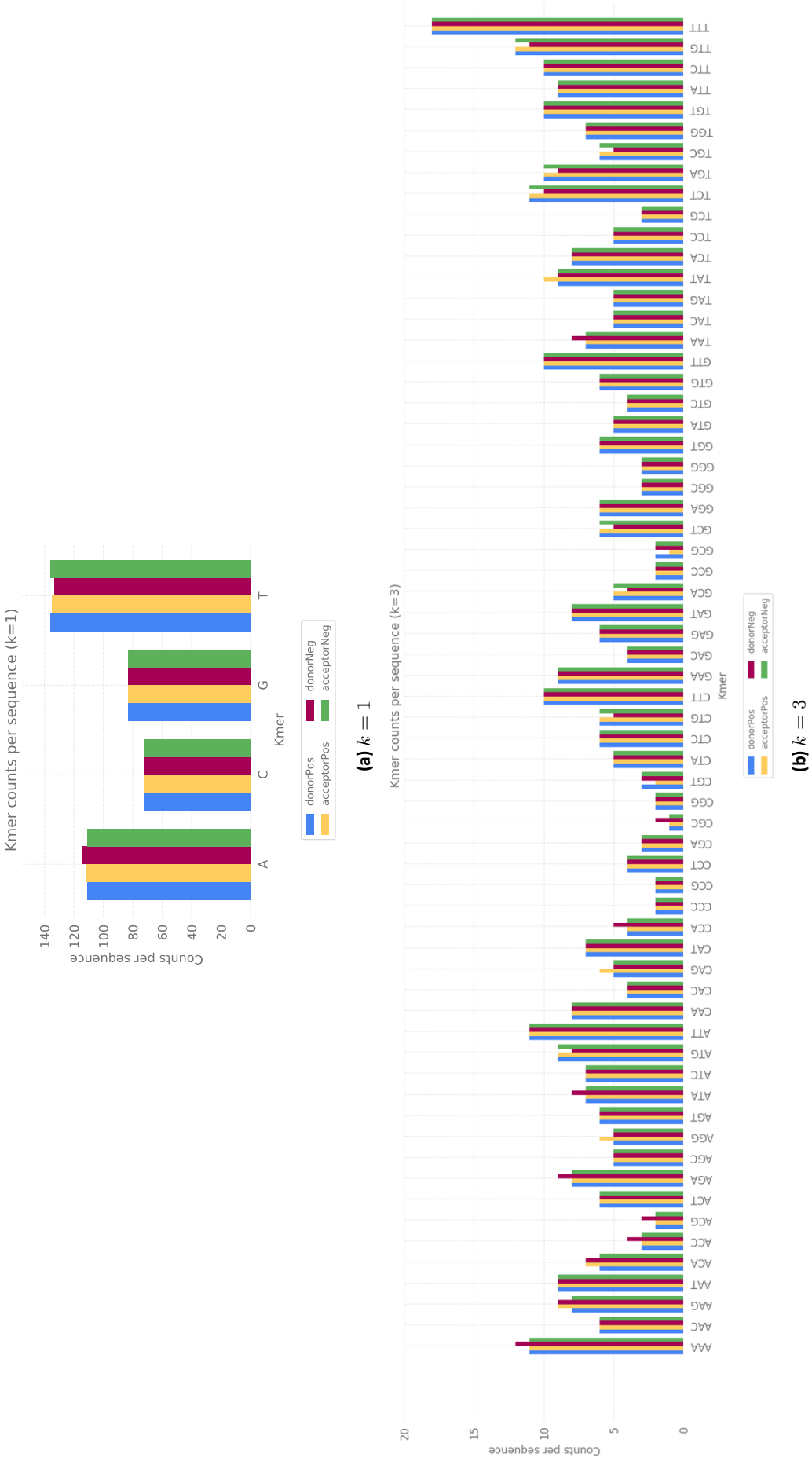
K-mer histograms

The histograms for $k = 1$ and $k = 3$ are plotted in figure 6.5. For $k = 1$ (figure 6.5a), it is seen that the count of DNA bases is independent from the ARAsplice class, and the GC-content is considerably lower than the number for A and T. This is a normal observation, as organisms typically have a GC-content inherent to their species. For *A. thaliana*, this value is determined to be around 36% (NCBI, 2011), which is in line with the results plotted in figure 6.5a. As the four classes share the same number of bases per single nucleotide, k-mers of length 1 will most likely not contribute towards producing meaningful features for splice site prediction. This is in contrast for k-mers of length 3 (figure 6.5b), although the differences are still small between the four classes for a certain k-mer. These effects become however more explicit when the k-mers of length 4 and 5 are plotted (plots not shown due to their large size). An algorithm could thus pick up on these subtle patterns, theoretically making k-mer encoding a valid alternative to other, more frequently used encodings.

K-mer network

A variety of different architectures consisting of only dense layers were tested for use with k-mer encoded splice data. None managed to capture the intricate differences between the four splice classes in the original nor the augmented ARAsplice set, with the best performing architecture stalling around a training and validation loss of 0.8. The SpliceRover1D model was also tested, with again the same performance issues as the dense architectures. The loss plots obtained with the SpliceRover1D model for both the original and augmented ARAsplice set are shown in figure 6.6 for a value of $k = 3$.

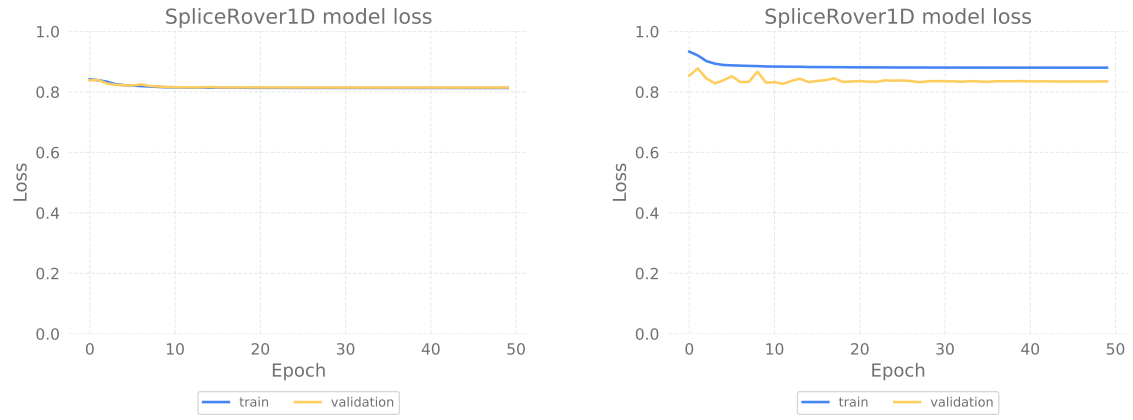
In the plot of the original ARAsplice run (figure 6.6a), a small decline in the beginning of the training process can be observed in the training and validation loss, meaning learning indeed takes place in the earliest epochs. However, the learning process stalls almost immediately after these few epochs, resulting in a loss that never declines anymore and stays around a value of 0.8. The same effect is seen when the augmented ARAsplice set is used (figure 6.6b), with only a few fluctuations in the validation loss in the beginning of the training process. All tested and optimised NNs stopped learning around a loss of 0.8 for both ARAsplice sets. This indicates that k-mer encoding cannot produce a meaningful data representation for splice site data, as even the deepest tested NN were not able to correctly predict the ARAsplice set. This is most likely due to the differences in k-mer counts being too small between the four classes, where not even a deep learning network can capture them.



(a) $k = 1$

(b) $k = 3$

Figure 6.5: K-mer counts per sequence for all classes within the ARAsplice dataset and for different values of k.



(a) Original ARAsplice

(b) Augmented ARAsplice

Figure 6.6: SpliceRover1D model loss plots after training on both the original and augmented k-mer encoded ARAsplice set, with $k = 3$.

Another observation is that the validation loss drops below the training loss for the augmented ARAsplice run (figure 6.6b). This is due to the effect of the dropout layers present in the SpliceRover1D model. Dropout injects a small amount of noise into the training set by setting different neurons to active during each pass of the data through the network. As dropout is not applied onto the validation set, the validation data lacks this noise which results in a slightly lower loss for the validation set than for the training set. This effect is bigger in the augmented dataset, as it contains more samples in both the training set. The effect of a model with and without dropout layers is visualised in appendix E.

6.2. Promoter prediction

6.2.1. K-mer histograms

In figure 6.7 the k-mers of length 1 are shown for the samples within the balanced ARAprom dataset. It is shown that the k-mer numbers between the classes differ for each of the kmers within the vocabularium. Organisms typically have a base distribution that is inherent to their species, which should only be differing in very small amounts when diverse genome sequences are analysed. As a difference is already seen in this base composition, it can be derived that the negative sample construction will provide no challenge for a machine learning algorithm and this regardless of the encoding used. When the histograms for higher k-mer counts are analysed for the balanced ARAprom dataset, the negative samples evolve into a uniform distribution with only a few outliers (figure 6.8b), whereas the positive samples are represented by a more sparse distribution (figure 6.8a). This uniform negative sample distribution is an artifact from the semi-random construction method that is used where every base has an equal chance of being chosen. Although 40% similarity is retained between the two samples, the remaining 60% inequality still has a large influence on the overall base composition of the negative samples. A possible solution to this could be to incorporate the species-

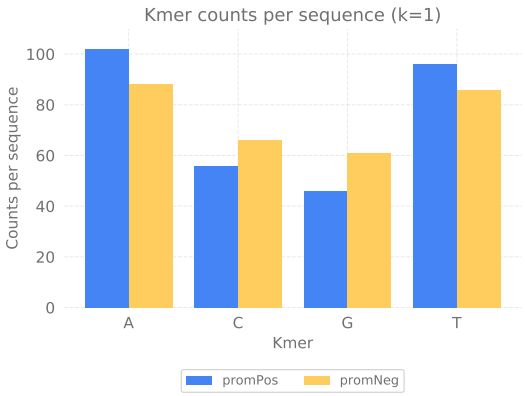
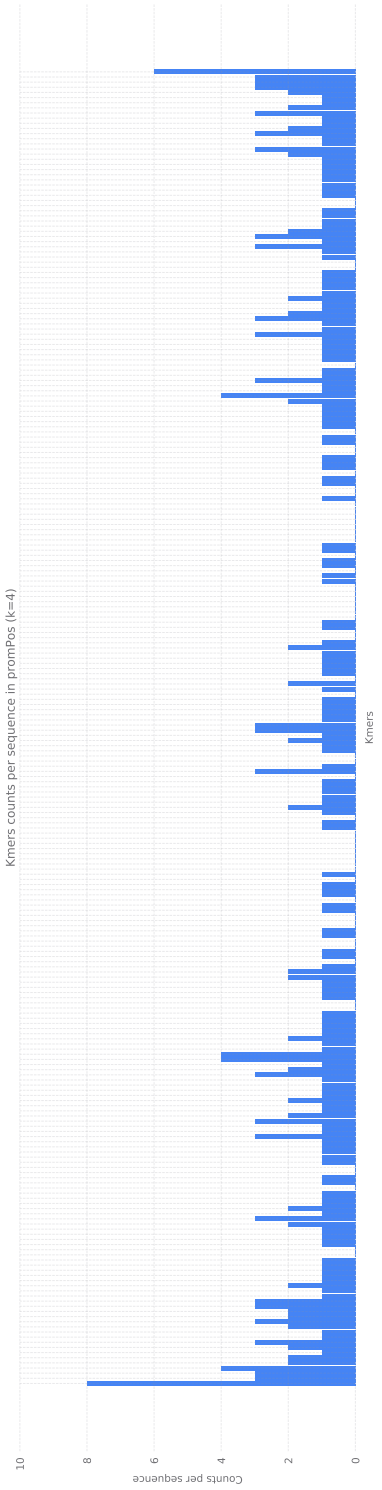
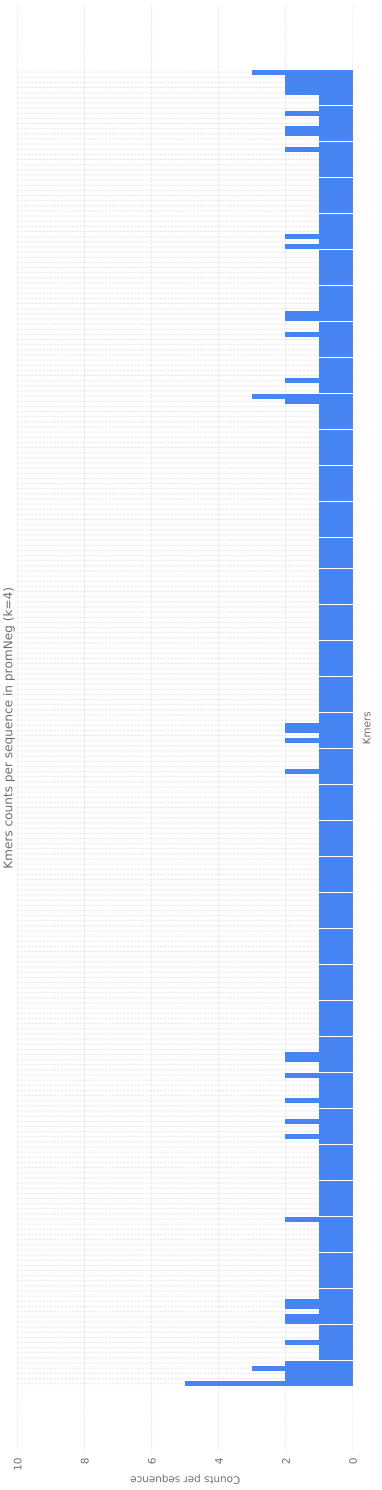


Figure 6.7: K-mer counts per sequence for both classes within the ARAprom-B dataset for $k = 1$.



(a) Positive promoters (promPos)



(b) Negative promoters (promNeg)

Figure 6.8: K-mer counts per sequence for both classes within the ARAprom-B dataset for $k = 4$.

specific base composition into the negative sampling method, so every one of the four DNA bases will have a different probability of being chosen as a substitute base. This will help in preserving the natural order of occuring nucleotides, resulting in a more similar distribution between the different k-mers in both classes.
The same observations are seen in the k-mer histograms of the conserved dataset approach for *A. thaliana*, which are visualised in appendix F.1. Again, the species-specific base composition is not retained over the two classes (figure F.1), and eventually evolves into a uniform distribution for the negative samples (figure F.2b). Little difference is also spotted in the outliers of the balanced and conserved approach, indicating that preserving the most conserved consensus sequences does not cancel out the influence of the 60% dissimilarity between samples. The same conclusions can be drawn for both the balanced and conserved balanced human dataset approach, which are respectively visualised in appendix F.2 and appendix F.3. In humans, the kmer representation ($k = 4$) of the positive samples is less sparse than the one for *A. thaliana*, resulting in a more uniform distribution that shows a higher resemblance to the negative samples k-mer distribution. However, the negative samples histogram lacks the frequent outliers that are seen in the positive sample histogram, indicating that the negative sample construction method cannot capture the underlying characteristics innate to human promoter samples.

Due to the large differences in k-mer composition, it is expected that the proposed DNN will have little trouble in predicting the correct class when k-mer encoding is used.

6.2.2. Performance of the ARAprom- and HOMpromnet models

The performance measures for the ARAprom- and HOMpromnet networks for all different encodings and the balanced and conserved ARAprom and HOMprom datasets are given in table 6.3, and this for both classes within the datasets.

A trend seen over the reported results is that the negative promoter class suffers from worse performance measures than the positive promoter class, except for the recall value where an opposite tendency is seen. A t-test is performed to detect if these trends are significant. These results can be found in table 6.4.

ARApromnet

The obtained results in table 6.3 with the one-hot encoding and a k-mer encoding of k equal to 5 are in line with state of the art results, where recall and specificity values up to 95% are obtained for TATA-box containing *A. thaliana* promoter datasets (Triska *et al.*, 2017). When compared to more conventional machine learning techniques, it is able to outperform the proposed models. Anwar *et al.* (2008) reports precision and recall values of respectively 86% and 89 % for the positive promoter class with the use of a SVM. Similar results are found by Azad *et al.* (2011), who also implemented a SVM to predict plant promoter sequences. Both extracted coding and non-coding sequences from elsewhere in the *A. thaliana* genome to construct their negative sample set. The highest reported precision and recall values are found with the one-hot balanced ARApromnet, respectively 94.7% and 94% when averaged over the two promoter classes. This is a relative increase of respectively 10.3% and 6.4% compared to the results obtained by Anwar *et al.* (2008) and Azad *et al.* (2011). When compared to other state of the art deep learning models for promoter prediction, it achieves comparable results. Umarov and Solovyev (2017) designed a CNN to predict *A. thaliana* promoter data, consisting of one convolutional layer with 200 filters of length 21, and one max-pooling layer of length 2 followed by one dense layer with 128 neurons. As negative construction approach, the intron sequences of each gene contained in the dataset are extracted. Umarov and Solovyev (2017) reported recall values of 94% and 95 % for respectively *A. thaliana* datasets with non-TATA and TATA promoters. This shows that the ARApromnet models implemented in this master dissertation can compete with and even outperform state of the art *A. thaliana* promoter prediction models.

In table 6.4, the p-values for the t-tests conducted between the negative and positive promoter classes

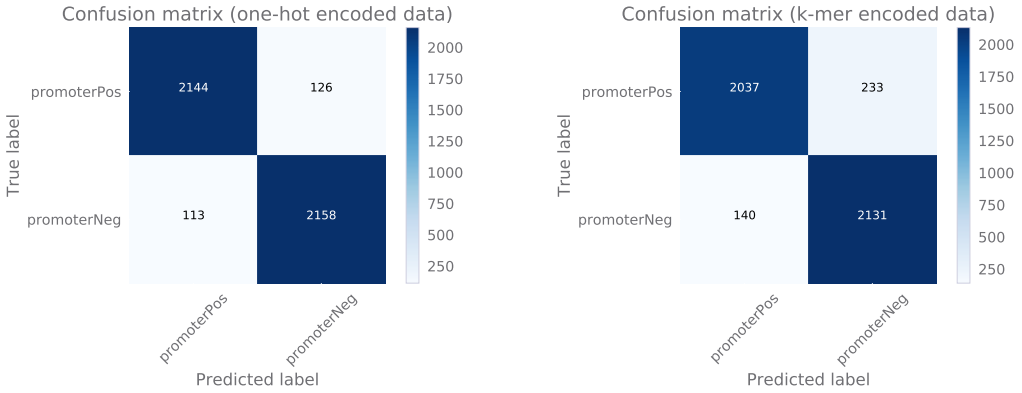
Table 6.3: PMs on the balanced and conserved ARAprom and HOMprom datasets for all different data encodings, and both classes within the datasets.

Encoding	Class	PM	*A. thaliana*		*H. sapiens*	
			Balanced	Conserved	Balanced	Conserved
Onehot	**promNeg**	**Pr**	0.945	0.932	0.930	0.928
		Re	0.950	0.952	0.949	0.945
		F1	0.948	0.942	0.939	0.936
		FDR	0.0552	0.0681	0.0702	0.0723
	promPos	**Pr**	0.950	0.951	0.948	0.944
		Re	0.944	0.930	0.928	0.926
		F1	0.947	0.941	0.938	0.935
		FDR	0.0501	0.0491	0.0518	0.0558
***k* = 3**	**promNeg**	**Pr**	0.900	0.907	0.895	0.888
		Re	0.936	0.923	0.926	0.926
		F1	0.918	0.915	0.910	0.907
		FDR	0.0995	0.0926	0.105	0.113
	promPos	**Pr**	0.933	0.922	0.923	0.923
		Re	0.896	0.906	0.892	0.883
		F1	0.915	0.914	0.907	0.903
		FDR	0.0665	0.0780	0.0769	0.0770
***k* = 4**	**promNeg**	**Pr**	0.921	0.922	0.912	0.909
		Re	0.950	0.944	0.928	0.928
		F1	0.935	0.933	0.920	0.918
		FDR	0.0794	0.0782	0.0877	0.0913
	promPos	**Pr**	0.948	0.943	0.926	0.926
		Re	0.918	0.920	0.911	0.907
		F1	0.933	0.931	0.919	0.917
		FDR	0.0519	0.0573	0.0735	0.0735
***k* = 5**	**promNeg**	**Pr**	0.929	0.930	0.904	0.910
		Re	0.961	0.959	0.953	0.938
		F1	0.945	0.945	0.928	0.924
		FDR	0.0715	0.0700	0.0959	0.0895
	promPos	**Pr**	0.960	0.958	0.950	0.936
		Re	0.926	0.928	0.899	0.908
		F1	0.943	0.943	0.924	0.922
		FDR	0.0402	0.0419	0.0500	0.0638

are shown for all performance values derived with the k-mer encoding. The promNeg class has a significantly overall lower precision and a higher FDR than the promPos class (p-values of respectively 0.00525 and 0.00476), and a significant better recall performance ($p = 0.00206$). When the one-hot encoded promNeg and promPos values are compared to each other, no significant differences are reported ($p > 0.207$). This is due to the different nature of the one-hot and k-mer encoding, and the conservation of consensus sequences within *A. thaliana* promoter sequences. It is seen in figure 5.3a that two major core promoter elements are preserved within *A. thaliana* promoters, namely the TATA-box and the inr. When one-hot encoding is applied, the model will learn to look for these specific sequences in the data. If a consensus sequence is accompanied by a lot of random noise, it is flagged as a negative sample, and if the bases are to some extent retained, a positive class is predicted. When k-mer encoding is applied, the conserved consensus sequences get split into smaller k-mers, transforming the once easily identifiable consensus sequences into much less detailed shorter k-mers. The network trained on k-mer encoded data will thus be more easily confused as to what constitutes a negative or positive promoter sample, resulting in larger differences of the FPs and FNs within a class. These effects are also seen in the confusion matrices for ARApromnet run on one-hot encoded data (figure 6.9a) and on k-mer

Table 6.4: Results of the t-tests performed on k-mer encoded data to derive if a difference can be seen in performance measures between the negative and positive promoter class. Significant p-values are highlighted in gray.

Class	PM	*p*-value	Mean $\pm$ SD	
			PromNeg	PromPos
A. thaliana	**Pr**	$5.25 \cdot 10^{-3}$	0.923 ± 0.0134	0.945 ± 0.0112
	Re	$2.06 \cdot 10^{-3}$	0.947 ± 0.0117	0.921 ± 0.0139
	F1	0.787	0.935 ± 0.0118	0.933 ± 0.0120
	FDR	$4.76 \cdot 10^{-3}$	0.0768 ± 0.0132	0.0544 ± 0.0132
H. sapiens	**Pr**	$1.97 \cdot 10^{-3}$	0.910 ± 0.0136	0.936 ± 0.0107
	Re	$5.75 \cdot 10^{-4}$	0.937 ± 0.0104	0.907 ± 0.0145
	F1	0.724	0.923 ± 0.0107	0.921 ± 0.0114
	FDR	$1.71 \cdot 10^{-3}$	0.0905 ± 0.0136	0.0653 ± 0.0107



(a) One-hot encoded ARAprom data

(b) K-mer encoded ARAprom data ($k = 3$)

Figure 6.9: Confusion matrices for the ARApromnet model fitted onto one-hot encoded and k-mer encoded balanced *A. thaliana* promoter data. No normalised confusion matrix is shown as the test data is completely balanced.

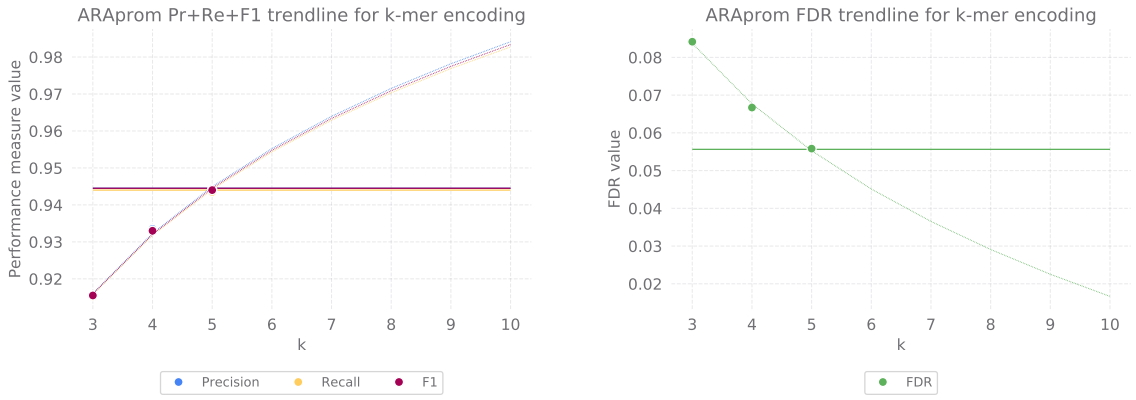
encoded data (figure 6.9b). Note that the comparison for the performance measures obtained with one-hot encoding is done using only two different values. The reported results could therefore be biased. To obtain statistically more robust results, these t-tests should ideally be conducted again over a series of repeated experiments. To this end, k-fold CV can be used. However, this approach is not feasible for deep learning approaches due to the amount of data and parameters inherent to this technique.

t-tests are also conducted to derive if differences exist between the balanced and conserved negative sample approach, and the one-hot and k-mer encoding for *A. thaliana* promoter prediction. Results from the conducted statistical tests are shown in table 6.5. These tests are again all conducted on only two performance measure values if the two promoter classes are tested separately and should therefore be treated with caution. For the balanced vs. conserved construction approaches comparison, all performance values obtained for the *A. thaliana* run are used, except for the one-hot-encoding ones. The analysis of the one-hot encoded balanced vs conserved negative sampling approach is not executed due to only having one value to perform a statistical test on.
No significant effects are seen between the application of the balanced and conserved negative sampling approach, showing that the conserved approach has no direct benefit over the balanced approach from Oubounyt *et al.*. While the conserved approach ensures that the most important features are shared between the negative and positives samples, it cannot cancel out the effect of the 60% random variety that is contained within the negative samples. This effect has previously been noted in section 6.2.1 during the observation of the k-mer histograms. The two sample classes still show too many differences

Table 6.5: P-values and mean $\pm$ SD for the different ARApromnet run comparisons. The first column indicates which two groups are compared to each other. The second column holds the performance measures (PM), and the third their accompanying p-value and the mean $\pm$ SD for the two groups within the comparison. Each comparison is done for both classes together (fourth column), and the two classes separately (fifth and sixth column). Significant p-values are highlighted in gray.

	PM		PromNeg + PromPos	PromNeg	PromPos
Balanced-conserved	**Pr**	*p*	0.775	0.930	0.673
		balanced	$0.936 \pm 1.80 \cdot 10^{-2}$	$0.924 \pm 1.62 \cdot 10^{-2}$	$0.948 \pm 9.65 \cdot 10^{-3}$
		conserved	$0.33 \pm 1.57 \cdot 10^{-2}$	$0.923 \pm 9.84 \cdot 10^{-3}$	$0.944 \pm 1.35 \cdot 10^{-2}$
	Re	*p*	0.658	0.507	0.348
		balanced	$0.936 \pm 1.98 \cdot 10^{-2}$	$0.949 \pm 8.87 \cdot 10^{-3}$	$0.921 \pm 1.72 \cdot 10^{-2}$
		conserved	$0.932 \pm 1.65 \cdot 10^{-2}$	$0.944 \pm 1.35 \cdot 10^{-3}$	$0.921 \pm 9.43 \cdot 10^{-3}$
	F1	*p*	0.699	0.783	0.825
		balanced	$0.935 \pm 1.20 \cdot 10^{-2}$	$0.936 \pm 1.17 \cdot 10^{-3}$	$0.934 \pm 1.23 \cdot 10^{-3}$
		conserved	$0.933 \pm 1.16 \cdot 10^{-3}$	$0.933 \pm 1.16 \cdot 10^{-3}$	$0.932 \pm 1.14 \cdot 10^{-3}$
	FDR	*p*	0.774	0.941	0.659
		balanced	$0.0643 \pm 1.78 \cdot 10^{-3}$	$0.0764 \pm 1.59 \cdot 10^{-3}$	$00.0522 \pm 9.39 \cdot 10^{-3}$
		conserved	$0.0669 \pm 1.56 \cdot 10^{-2}$	$0.0772 \pm 9.65 \cdot 10^{-3}$	$0.0566 \pm 1.35 \cdot 10^{-3}$
Onehot vs. *k* = 3	**Pr**	*p*	0.0150	0.0417	0.0531
		onehot	$0.944 \pm 7.56 \cdot 10^{-3}$	$0.938 \pm 6.50 \cdot 10^{-3}$	$0.950 \pm 5.00 \cdot 10^{-4}$
		***k* = 3**	$0.915 \pm 1.28 \cdot 10^{-2}$	$0.903 \pm 3.50 \cdot 10^{-3}$	$0.927 \pm 5.50 \cdot 10^{-3}$
	Re	*p*	0.0301	0.0821	0.0526
		onehot	$0.944 \pm 8.60 \cdot 10^{-2}$	$0.951 \pm 1.00 \cdot 10^{-3}$	$0.937 \pm 7.00 \cdot 10^{-3}$
		***k* = 3**	$0.915 \pm 8.60 \cdot 10^{-3}$	$0.929 \pm 6.50 \cdot 10^{-3}$	$0.901 \pm 5.00 \cdot 10^{-3}$
	F1	*p*	$1.00 \cdot 10^{-5}$	0.0135	0.01405
		onehot	$0.944 \pm 3.04 \cdot 10^{-3}$	$0.945 \pm 3.00 \cdot 10^{-3}$	$0.944 \pm 3.00 \cdot 10^{-3}$
		***k* = 3**	$0.915 \pm 1.50 \cdot 10^{-3}$	$0.916 \pm 1.50 \cdot 10^{-3}$	$0.914 \pm 5.00 \cdot 10^{-4}$
	FDR	*p*	0.0159	0.0423	0.0592
		onehot	$0.0556 \pm 8.94 \cdot 10^{-3}$	$0.0616 \pm 6.66 \cdot 10^{-3}$	$0.0496 \pm 2.56 \cdot 10^{-4}$
		***k* = 3**	$0.0842 \pm 1.52 \cdot 10^{-2}$	$0.0960 \pm 7.30 \cdot 10^{-3}$	$0.0723 \pm 4.70 \cdot 10^{-3}$
Onehot vs. *k* = 4	**Pr**	*p*	0.231	0.120	0.188
		onehot	$0.944 \pm 8.85 \cdot 10^{-3}$	$0.938 \pm 6.65 \cdot 10^{-3}$	$0.950 \pm 2.68 \cdot 10^{-4}$
		***k* = 4**	$0.933 \pm 1.27 \cdot 10^{-2}$	$0.921 \pm 5.61 \cdot 10^{-3}$	$0.945 \pm 9.91 \cdot 10^{-3}$
	Re	*p*	0.294	0.333	0.125
		onehot	$0.944 \pm 1.00 \cdot 10^{-3}$	$0.951 \pm 2.55 \cdot 10^{-3}$	$0.937 \pm 7.07 \cdot 10^{-3}$
		***k* = 4**	$0.933 \pm 1.41 \cdot 10^{-2}$	$0.947 \pm 9.73 \cdot 10^{-3}$	$0.919 \pm 5.24 \cdot 10^{-3}$
	F1	*p*	$1.02 \cdot 10^{-3}$	0.0736	0.0629
		onehot	$0.944 \pm 4.47 \cdot 10^{-3}$	$0.945 \pm 4.44 \cdot 10^{-3}$	$0.944 \pm 4.44 \cdot 10^{-3}$
		***k* = 4**	$0.933 \pm 7.36 \cdot 10^{-3}$	$0.934 \pm 7.57 \cdot 10^{-3}$	$0.932 \pm 7.07 \cdot 10^{-3}$
	FDR	*p*	0.231	0.117	0.210
		onehot	$0.0556 \pm 8.94 \cdot 10^{-3}$	$0.0616 \pm 6.66 \cdot 10^{-3}$	$0.0496 \pm 2.56 \cdot 10^{-4}$
		***k* = 4**	$0.0667 \pm 1.28 \cdot 10^{-2}$	$0.0788 \pm 5.52 \cdot 10^{-4}$	$0.0546 \pm 9.64 \cdot 10^{-3}$
Onehot vs. *k* = 5	**Pr**	*p*	0.980	0.301	0.0168
		onehot	$0.944 \pm 8.85 \cdot 10^{-3}$	$0.938 \pm 6.65 \cdot 10^{-3}$	$0.950 \pm 2.68 \cdot 10^{-4}$
		***k* = 5**	$0.944 \pm 1.94 \cdot 10^{-2}$	$0.929 \pm 1.15 \cdot 10^{-4}$	$0.959 \pm 9.43 \cdot 10^{-3}$
	Re	*p*	0.964	0.0238	0.292
		onehot	$0.944 \pm 0.100 \cdot 10^{-3}$	$0.951 \pm 2.55 \cdot 10^{-3}$	$0.937 \pm 7.07 \cdot 10^{-3}$
		***k* = 5**	$0.943 \pm 2.16 \cdot 10^{-2}$	$0.960 \pm 9.01 \cdot 10^{-3}$	$0.927 \pm 1.22 \cdot 10^{-2}$
	F1	*p*	0.785	1.00	0.771
		onehot	$0.944 \pm 4.47 \cdot 10^{-3}$	$0.945 \pm 4.43 \cdot 10^{-3}$	$0.944 \pm 4.44 \cdot 10^{-3}$
		***k* = 5**	$0.944 \pm 1.00 \cdot 10^{-3}$	0.945 ± 0.00	0.943 ± 0.00
	FDR	*p*	0.978	0.296	0.0134
		onehot	$0.0556 \pm 8.94 \cdot 10^{-3}$	$0.0616 \pm 6.66 \cdot 10^{-3}$	$0.0496 \pm 2.56 \cdot 10^{-4}$
		***k* = 5**	$0.0559 \pm 1.94 \cdot 10^{-2}$	$0.0708 \pm 1.12 \cdot 10^{-4}$	$0.0411 \pm 9.33 \cdot 10^{-4}$

between them, and thus do not provide a challenge for the network at hand. This results in performance measures equal to the one obtained with the balanced approach.
The next set of comparisons consist of the one-hot encoding versus the k-mer encoding, and this for different values of k. For $k = 3$ it is seen that the model performs significantly worse than when one-hot encoded data is used, and this for every performance measure outside of recall when the two classes are analysed separately. A tipping point is detected for $k = 4$, where no significant differences are reported except for the F1-value when both the promNeg and promPos classes are observed. However, as seen in table 6.4, a significant difference is found between the two classes and this obtained value could thus be an artifact of regarding the negative and positives classes as one group. When the value of k is set to 5, significant improvements are started to be seen in favour of the k-mer encoding, with an elevated precision and recall for respectively the promPos and promNeg class ($p = 0.0168$ and $p = 0.0238$). For the promNeg class, a decline is also detected in the FDR ($p = 0.0134$). In contrast to the k-mer encoding employed on splice site data, k-mer encoding is capable of capturing the intricate sequence details that prevail within *A. thaliana* promoters, yet only for a value of $k = 5$. K-mer encoding might be able to outperform one-hot encoding if larger values of k are used. To visualise this effect, a logarithmic trendline is fitted onto the performance measure results obtained with the three different values of k. The outcome can be seen in figure 6.10. The trendline is fitted onto the average taken over both the values for promPos and promNeg and over the conserved and balanced approach. Although significant differences are found between the positive and negative class, their average can still capture the general trend that is expected in the results for growing values of k.



(a) Trend for precision, recall, and F1

(b) Trend for FDR

Figure 6.10: Expectations in performance measures for higher values of k on the ARApromset. The dotted values are the performance results obtained with the tested values of k. A logarithmic trendline is fitted onto these values, yielding the dotted curved line. The horizontal solid line indicates the results obtained when one-hot encoding is used. The trendline is fitted onto the average over the promPos and promNeg performance measure results.

In the plots in figure 6.10 it can be seen that the k-mer encoding for $k = 5$ intercepts with the values obtained with one-hot encoded data, indicating both encodings are equal in performance. If k grows larger than 5, it immediately starts to outperform the one-hot encoded ARApromnet fits, with values as high as 98% for precision, recall and F1, and as low as 2% for FDR for a value of $k = 10$. These values are probably not realistic due to a too liberal fit onto the data for $k \leq 3$ and the lack of more data points. It can however be expected that an incline will still be seen if experiments are conduced for $k > 3$. K-mer encoding can thus provide a valid alternative to the more commonly used one-hot encoding when it is used with *A. thaliana* promoter data. This provides a benefit in the implementation of the NNs. As one-hot consists of two dimensions, a CNN is typically applied. In this architecture, a variety of hyperparameters are present that need to be tweaked in order to obtain good prediction performance. This tweaking requires a certain experience on how different hyperparameters have a different influence on the outcome. K-mer encoded data has only one dimension, so only a dense NN

can be applied to it. While still having a certain number of hyperparameters that need to be adjusted to obtain a good fit to the data, it is considerably easier to optimise due to its simpler architecture than a CNN.

HOMpromnet

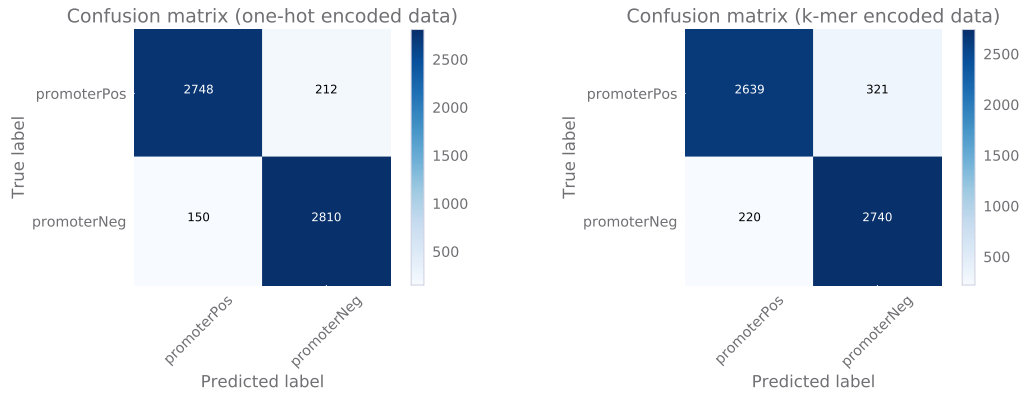
Research conducted on *H. sapiens* promoter prediction reports accuracies and sensitivities as high as 94% and 95%. These performances are respectively reached by Lai *et al.* (2019), who implemented a SVM, and by Umarov and Solovyev (2017). Umarov and Solovyev (2017) designed a series of CNN architectures for use with prokaryotic and eukaryotic TATA and non-TATA promoter data. The recall rate of 95% was found for both the human TATA and non-TATA datasets. Other reported results were 95% and 90% for the recall values of respectively human TATA and human non-TATA promoter sequences. The results obtained on the HOMprom set using the HOMpromnet model are equal to the state of the art results (table 6.3). In the research article by Oubounyt *et al.* (2019), the same human dataset is used as the one in this dissertation, with the difference that Oubounyt *et al.* split it into two new datasets yielding a TATA and a non-TATA promoter set. The scheme used to construct their negative samples is equal to the balanced strategy introduced in this thesis. For the TATA set, precision and recall values of respectively 93% and 95% are reported. For the non-TATA set, they obtain a precision of 97% and a recall of 95%. While the HOMpromnet is not able to achieve performance measures this high, this is no reason for concern as the HOMprom dataset in this dissertation was not split into two different ones. It can be assumed that their reported value for the entirety of the dataset will lie somewhere between a precision value of 93% and 97%. This is a result that the HOMpromnet model designed in this thesis is able to obtain. While these reported results make it seem that human promoter prediction has reached nearly perfect performance, this is hardly the case. The designed models are able to predict their training and test data in a highly reliable manner, yet fail when used in more realistic settings such as the human reference genome. Bajic *et al.* (2004) reviewed eight frequently used PPPs, and found that some of them perform even worse than when random guessing is used. The reported state of the art results should therefore not be seen as the golden standard to which a model can be compared, and models should ideally always be evaluated onto a different, unseen test set.

The same trend as with the ARAprom data is seen in the performances measures, where the promNeg class performs significantly worse than the promPos class, except for the recall value where the opposite effect is observed (table 6.4). If the confusion matrices are plotted for the one-hot and k-mer encoding ($k = 3$; figure 6.11), it is seen that the FPs and FNs results of the one-hot encoded HOMpromnet run also show rather large differences between them (figure 6.11a). When a t-test is conducted, significant differences are indeed detected between the precision, recall, and FDR values obtained with one-hot encoded data (p-values of respectively 0.0169, 0.0123, and 0.0164). This is because of the less conserved consensus sequences within human promoters. As a result, the model will also have a hard time learning meaningful representations on the one-hot encoded data for the different classes, resulting in elevated FP and FN results within a class.

In table 6.6 the p-values are found for the different group comparisons between the balanced vs. conserved negative sampling approach, and the one-hot encoding vs the k-mer encoding, and this for all three tested values of k. The differences in the balanced vs. conserved approach are here calculated using all different values from both encodings as the one-hot encoding showed the same significant differences between the negative and positive promoter class.
For the comparison of the balanced and the conserved negative sample set construction approach, again no differences are found between the two groups. This further proves that the conserved approach does not provide a significant benefit compared to the balanced approach by Oubounyt *et al.*. For the comparisons between the one-hot encoding and the three different k-mer encodings, the results differ from what was found with the ARApromnet model. For all k-mer values, significant worse results are seen compared to when one-hot encoded data is used. These effects are worst for an encoding with $k = 3$, and slowly wears of when k grows in value. However, at $k = 5$, significant worse performances are still seen for the k-mer encoding. A logarithmic trendline is again fitted to evaluate what effect growing

Table 6.6: P-values and mean $\pm$ SD for the different HOMpromnet run comparisons. The first column indicates which two groups are compared to each other. The second column holds the performance measures (PM), and the third their accompanying p-value and the mean $\pm$ SD for the two groups within the comparison. Each comparison is done for both classes together (fourth column), and the two classes separately (fifth and sixth column). Significant p-values are highlighted in gray.

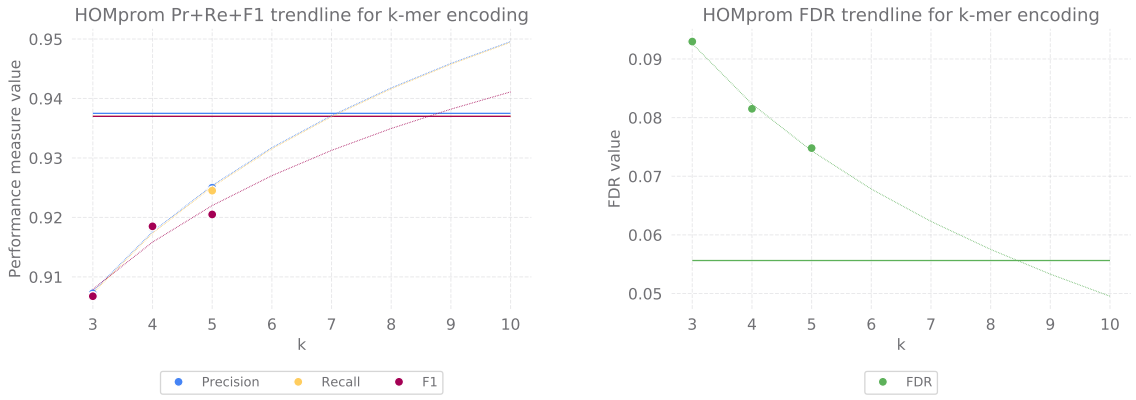
	PM		PromNeg + PromPos	PromNeg	PromPos
Balanced-conserved	**Pr**	*p*	0.752	0.896	0.618
		balanced	$0.923 \pm 1.82 \cdot 10^{-2}$	$0.910 \pm 0.128 \cdot 10^{-2}$	$0.936 \pm 1.32 \cdot 10^{-2}$
		conserved	$0.920 \pm 1.65 \cdot 10^{-2}$	$0.908 \pm 1.41 \cdot 10^{-2}$	$0.932 \pm 8.31 \cdot 10^{-3}$
	Re	*p*	0.768	0.587	0.903
		balanced	$0.923 \pm 20.3 \cdot 10^{-2}$	$0.939 \pm 7.96 \cdot 10^{-2}$	$0.907 \pm 1.52 \cdot 10^{-2}$
		conserved	$0.920 \pm 1.86 \cdot 10^{-2}$	$0.934 \pm 7.63 \cdot 10^{-3}$	$0.906 \pm 1.52 \cdot 10^{-2}$
	F1	*p*	0.631	0.739	0.775
		balanced	$0.923 \pm 1.09 \cdot 10^{-2}$	$0.924 \pm 1.06 \cdot 10^{-2}$	$0.922 \pm 1.11 \cdot 10^{-2}$
		conserved	$0.920 \pm 1.102 \cdot 10^{-2}$	$0.921 \pm 1.04 \cdot 10^{-2}$	$0.919 \pm 1.14 \cdot 10^{-2}$
	FDR	*p*	0.746	0.882	0.618
		balanced	$0.0764 \pm 1.82 \cdot 10^{-2}$	$0.0897 \pm 1.12 \cdot 10^{-2}$	$0.0630 \pm 1.22 \cdot 10^{-2}$
		conserved	$0.0795 \pm 1.67 \cdot 10^{-2}$	$0.0914 \pm 1.42 \cdot 10^{-2}$	$0.0675 \pm 9.31 \cdot 10^{-3}$
Onehot vs. *k* = 3	**Pr**	*p*	0.0277	$9.29 \cdot 10^{-3}$	$7.84 \cdot 10^{-3}$
		onehot	$0.937 \pm 8.64 \cdot 10^{-3}$	$0.929 \pm 1.00 \cdot 10^{-3}$	$0.946 \pm 2.00 \cdot 10^{-3}$
		***k* = 3**	$0.907 \pm 1.51 \cdot 10^{-2}$	$0.891 \pm 3.50 \cdot 10^{-3}$	0.923 ± 0.00
	Re	*p*	0.0545	$8.98 \cdot 10^{-3}$	0.0134
		onehot	$0.937 \pm 1.00 \cdot 10^{-2}$	$0.947 \pm 2.00 \cdot 10^{-3}$	$0.927 \pm 1.00 \cdot 10^{-3}$
		***k* = 3**	$0.906 \pm 1.95 \cdot 10^{-2}$	0.926 ± 0.00	$0.887 \pm 4.50 \cdot 10^{-3}$
	F1	*p*	0.00	$5.31 \cdot 10^{-3}$	$6.24 \cdot 10^{-3}$
		onehot	$0.937 \pm 1.58 \cdot 10^{-3}$	$0.937 \pm 1.50 \cdot 10^{-3}$	$0.936 \pm 1.50 \cdot 10^{-3}$
		***k* = 3**	$0.906 \pm 2.48 \cdot 10^{-3}$	$0.908 \pm 1.50 \cdot 10^{-3}$	$0.905 \pm 2.00 \cdot 10^{-3}$
	FDR	*p*	0.0290	0.0106	$7.39 \cdot 10^{-2}$
		onehot	$0.0625 \pm 8.94 \cdot 10^{-3}$	$0.0713 \pm 1.05 \cdot 10^{-3}$	$0.0538 \pm 2.00 \cdot 10^{-3}$
		***k* = 3**	$0.0929 \pm 1.61 \cdot 10^{-2}$	$0.108 \pm 3.75 \cdot 10^{-3}$	$0.0769 \pm 5.00 \cdot 10^{-5}$
Onehot vs. *k* = 4	**Pr**	*p*	0.0288	$9.36 \cdot 10^{-3}$	$9.58 \cdot 10^{-3}$
		onehot	$0.937 \pm 8.85 \cdot 10^{-3}$	$0.929 \pm 1.50 \cdot 10^{-3}$	$0.946 \pm 2.00 \cdot 10^{-3}$
		***k* = 4**	$0.918 \pm 7.88 \cdot 10^{-3}$	$0.910 \pm 1.50 \cdot 10^{-3}$	0.926 ± 0.00
	Re	*p*	0.0614	0.0109	0.0151
		onehot	$0.937 \pm 1.00 \cdot 10^{-2}$	$0.947 \pm 2.00 \cdot 10^{-3}$	$0.927 \pm 1.00 \cdot 10^{-3}$
		***k* = 4**	$0.918 \pm 9.60 \cdot 10^{-2}$	0.928 ± 0.00	$0.909 \pm 2.00 \cdot 10^{-3}$
	F1	*p*	0.00	$9.36 \cdot 10^{-3}$	$9.36 \cdot 10^{-3}$
		onehot	$0.937 \pm 1.58 \cdot 10^{-3}$	$0.937 \pm 1.50 \cdot 10^{-3}$	$0.936 \pm 1.50 \cdot 10^{-3}$
		***k* = 4**	$0.918 \pm 1.18 \cdot 10^{-3}$	$0.919 \pm 1.00 \cdot 10^{-3}$	$0.918 \pm 1.00 \cdot 10^{-3}$
	FDR	*p*	0.0339	0.0127	0.0101
		onehot	$0.0625 \pm 8.87 \cdot 10^{-3}$	$0.0713 \pm 1.05 \cdot 10^{-3}$	$0.0538 \pm 2.00 \cdot 10^{-3}$
		***k* = 4**	$0.0815 \pm 8.01 \cdot 10^{-3}$	$0.0895 \pm 1.80 \cdot 10^{-3}$	0.0735 ± 0.00
Onehot vs. *k* = 5	**Pr**	*p*	0.335	0.0200	0.720
		onehot	$0.937 \pm 8.85 \cdot 10^{-3}$	$0.929 \pm 1.00 \cdot 10^{-3}$	$0.946 \pm 2.00 \cdot 10^{-3}$
		***k* = 5**	$0.925 \pm 1.87 \cdot 10^{-2}$	$0.907 \pm 3.00 \cdot 10^{-3}$	$0.907 \pm 7.00 \cdot 10^{-3}$
	Re	*p*	0.404	0.846	0.036
		onehot	$0.937 \pm 1.12 \cdot 10^{-2}$	$0.947 \pm 2.00 \cdot 10^{-3}$	$0.927 \pm 1.00 \cdot 10^{-3}$
		***k* = 5**	$0.924 \pm 2.18 \cdot 10^{-2}$	$0.945 \pm 7.50 \cdot 10^{-3}$	$0.903 \pm 4.50 \cdot 10^{-3}$
	F1	*p*	$2.00 \cdot 10^{-4}$	0.0441	0.0172
		onehot	$0.937 \pm 1.58 \cdot 10^{-3}$	$0.937 \pm 1.50 \cdot 10^{-3}$	$0.936 \pm 1.50 \cdot 10^{-3}$
		***k* = 5**	$0.924 \pm 2.17 \cdot 10^{-3}$	$0.926 \pm 2.00 \cdot 10^{-3}$	$0.923 \pm 1.00 \cdot 10^{-2}$
	FDR	*p*	0.343	0.0237	0.708
		onehot	$0.0625 \pm 8.87 \cdot 10^{-3}$	$0.0713 \pm 1.05 \cdot 10^{-3}$	$0.0538 \pm 2.00 \cdot 10^{-3}$
		***k* = 5**	$0.0748 \pm 1.85 \cdot 10^{-2}$	$0.0927 \pm 3.20 \cdot 10^{-3}$	$0.0569 \pm 6.90 \cdot 10^{-3}$



(a) One-hot encoded HOMprom data

(b) K-mer encoded HOMprom data ($k = 3$)

Figure 6.11: Confusion matrices for the HOMpromnet model fitted onto one-hot encoded and k-mer encoded balanced *H. sapiens* promoter data. No normalised confusion matrix is shown as the test data is completely balanced due to the usage of a stratified split.



(a) Trend for precision, recall, and F1

(b) Trend for FDR

Figure 6.12: Expectations in performance measures for higher values of k on the HOMpromset. The dotted values are the performance results obtained with the tested values of k. A logarithmic trendline is fitted onto these values, yielding the dotted curved line. The horizontal solid line indicates the results obtained when one-hot encoding is used. The trendline is fitted onto the average over the promPos and promNeg performance measure results.

values of k will have on the performance measures, which is pictured in figure 6.12. Here it is observed that k-mer encoding only starts to outperform the one-hot encoding from a value of $k = 7$, and this for the precision and recall values. F1 and FDR values will only get significantly better if $k \geq 8$. While k-mer encoding was able to achieve results equal to the ones obtained with one-hot encoding in the case of *A. thaliana* promoter prediction, this is not the case anymore for human promoter prediction. This is because human promoter prediction is a problem that is much more difficult than for less complex species like *A. thaliana*. In the latter, a conservation is seen of both the TATA-box and inr, while in the HOMprom set only the inr is conserved. Human promoter sequences are thus prone to a larger difference in consensus sequences. Due to this variety, the distribution of the k-mers in the positive sample set evolves into a more uniform distribution, which is also found within the negative sample set of both the balanced and conserved approach. This has as a result that the differences between the two promoter classes get smaller. As was illustrated with the k-mer encoding applied onto the ARAsplice set, k-mer encoding will not work if no or only a little amount of dissimilarities are found between the data classes. The k-mer encoded HOMprom dataset seems to lie on a point where models are still able to learn meaningful features from it, yet also start suffering from the many similarities that are already seen between the two classes. K-mer encoding will thus only have an application in species where large

differences are examined in the k-mer composition between the different classes. This can be assessed by plotting the k-mer histograms as was done in section 6.2.1, and can be used as a tool in order to determine is k-mer encoding a certain dataset is worth the effort.

7. Conclusion

In this master dissertation, deep learning was applied to predict splice sites and promoter sequences in DNA data. These specific problems in genome annotation are still plagued by a fair amount of FP results. In splice site detection, this is due to the high class imbalance in favour of the negative samples. For promoter detection, no easily identifiable reason can be appointed, as it suffers from a variety of problems, such as the lack of easily identifiable consensus sequences and a straight-forward negative sample construction method. In addition, the most conserved consensus sequences in promoters stretch over tens or even twenty bp and are variable in length. This is in contrast with splice sites, of which the most conserved sequences are only two bp long. Due to this greater length and variable size, it is difficult to capture the details contained within promoter sequences in an encoding which uses a fixed length, as sequences have to be padded or truncated to this set length.

For one-hot encoded splice site data, the SpliceRover network by Zuallaert *et al.* (2018) was reimplemented. The network as designed by Zuallaert *et al.* employs 2D convolution onto the 2D one-hot encoded splice site data to which an extra dimension is added in order to make it 3D. In this thesis, a novel version of the SpliceRover was also implemented apart from the original, which utilises 1D convolutions that can directly work onto the 2D one-hot encoded dataset. Both models failed in reproducing the results obtained by Zuallaert *et al.*, reaching maximum values for precision, recall and FDR of 87, 94, 94, and 13 percent for the positive donor class, and 86, 93, 89, 14 percent for the positive acceptor class. This result is due to the slightly different architecture that is used in this dissertation for the SpliceRover2D model. The SpliceRover2D model as originally implemented by Zuallaert *et al.* has an output layer with only two neurons, and two different classifiers are trained for the donor and acceptor classes separately. The self-implemented SpliceRover2D network uses four neurons in its output layer. While this has the disadvantage of not reaching state of the art results in splice site prediction, it has the benefit that it can work on the dataset in its entirety and only one model has to be trained. Especially the latter is a considerable advantage, as training and tweaking of deep learning models can take a long time due to the amount of data they work on. It however does not outweigh the low performance results, as Zuallaert *et al.* obtained precision values and FDRs of respectively 96% and 4.4% for the positive donor class, and respectively 94% and 6.1% for the positive acceptor class. Nonetheless, both SpliceRover2D and SpliceRover1D are able to outperform state of the art methods such as SVMs (Degroeve *et al.*, 2005). Together with the results obtained by Zuallaert *et al.*, this indicates that deep learning has the ability to significantly outperform other machine learning methods.
No differences were found between the self-implemented SpliceRover2D and the SpliceRover1D performance results. This was expected, as the filter sizes of the convolution layers in the original SpliceRover2D network are initialised in such a way that they are essentialy equivalent to 1D convolutions. The SpliceRover1D network has the advantage that is takes less time to train as 1D convolutions need less computational resources than their 2D counterparts. No differences were found either between the results obtained with a stratified split and a grouped split. This indicates that no data leakage takes place in either of the splits, and that splice site patterns are conserved independently from their accompanying gene. On the contrary, better performance is reported for runs where the augmented *A. thaliana* splice site dataset was used, with the most significant gains found in the prediction of the positive acceptor class. Here, a relative decrease of 21.7% was noted in their FDR, which is twice the value of the decrease seen in the FDR of the positive donor class. The reverse complement augmentation technique can thus be a valuable tool in improving the results for positive acceptor sites. This is especially interesting as this class still suffers from worse prediction rates than the positive donor class. A downside to this augmentation method is however that it can also result in a lower recall rate for the positive acceptor class. An assessment therefore has to be made if higher precision or higher recall rates are preferred when this DNA augmentation technique is used.

Efforts made to develop a NN that works on k-mer encoded splice site data proved fruitless, as no proposed deep learning architecture managed to get the validation loss below 0.8. This is due to the very small differences that are observed between the k-mer counts for the four different splice classes. K-mer encoding is therefore not a valid encoding strategy for use with splice site data. When k-mer encoding is applied onto the promoter dataset of *A. thaliana*, it is able to reach results equivalent to the ones obtained with the one-hot encoded *A. thaliana* promoter data when a value of $k = 5$ is used. This effect however disappears again in the *H. sapiens* promoter dataset. Consequently, the performance of k-mer encoding relies heavily on the data it is applied on. When the data of interest is k-mer encoded, large enough differences should be seen between the different classes. Once this is not the case, the performance obtained with k-mer encoding starts deteriorating. In the worst case, a model trained on k-mer encoded data will fail to learn any meaningful data representations, as was the case for the *A. thaliana* splice site set. In order to assess if k-mer encoding will work for a certain dataset, k-mer histograms can be plotted. If these display too many similarities between them, kmer encoding will not work on the data at hand. This does not mean that the application of k-mer encoding is futile in all cases. When the right balance between similarity and dissimilarity is reached between the different classes, it can compete with the more frequently used one-hot encoding. K-mer encoding then has the advantage that it works with dense network architectures, as opposed to CNNs for one-hot encoded data. These are easier to tweak and demand less expert knowledge of the techniques involved in deep learning for them to be optimised.

7.1. Future perspectives

Improvements to the techniques presented in this master dissertation can be done in a variety of ways. The SpliceRover as originally implemented by Zuallaert *et al.* has the disadvantage that two different networks have to be trained, resulting in a longer training time and more effort spent on finding the most optimal hyperparameters for both networks. If one wants to use the online version of their network, a choice has to be made between which model to run on the provided sequences. This means the user has to make two prediction calls to the model, and also two different analyses of the results need to be conducted. This can be solved with the implementation of a branched network. Such a network consists of two smaller networks that learn different features parallel to each other. By constructing two branches, one branch can focus on correctly predicting if a sample is a negative or positive donor splice site, while the other branch learns how to detect the differences between negative and positive acceptor sites. This way, only one network has to be trained, and the user does not have to select their preferred model resulting in a more user-friendly prediction experience.
The grouped split can be revised to identify the function of the gene they belong to. Genes with similar functions often share the same expression patterns (Inoue and Horimoto, 2017; Hickman *et al.*, 2018). It is thus possible that, while still being highly conserved in general, splice site patterns differ slightly across the different gene groups. This function group can also be passed onto the network as an additional label, giving the network the ability to learn different patterns according to the gene function group at hand. Furthermore, this is especially interesting for application in promoter prediction, as promoters directly drive the gene expression process and are prone to much more variety in their consensus sequence than splice sites are.

The negative promoter dataset construction can also be enhanced further. Firstly, the balanced strategy by Oubounyt *et al.* and the novel proposed conserved strategy should be altered in such a way that the random choice of bases retains the species-specific DNA base composition. This will result in more shared features between the positive and negative promoter classes, providing the DNN with a bigger challenge than when the balanced and conserved approach are used. It will possibly have the advantage that the trained deep learning model will be more robust when used in real-life prediction predictions. Secondly, the negative sampling method can be adapted according to the conclusions found by Gusmao and de Souto (2014). They stated that negative sampling techniques could be improved when differ-

ent strategies are used together in order to construct the negative sample set. Another enhancement that can be made to ensure the artificially constructed dataset mimics real-life genome conditions in the best possible way is to introduce a class imbalance in favour of the negative samples. Utilising different negative sampling methods will produce more negative samples than there are positive ones, and thus consequently also results into the class imbalance approach. In appendix G, a small proposal is written on how to apply these strategies onto the promoter datasets of *A. thaliana* and *H. sapiens* used in this dissertation.
A whole new approach to the generation of negative samples can be done by applying a machine learning technique called generative adversarial networks (GANs, Goodfellow *et al.* (2014)). This approach is able to generate its own data with the same characteristics as its received input data. It consists of two neural networks, namely a generative one and a discriminative one. The discriminative network is a network such as the ones seen in this master thesis, which classifies data based on their characteristics and features. Another way of stating this is that a discriminative network maps features to labels. A generative network performs the opposite task by modeling features given a certain label. It will generate new, real-looking data instances, which the discriminator then evaluates to see if the generated sample is part of the original training set or not. The goal of the generator is to model data in such a way that the discriminator becomes confused in calling out the falsely provided data samples. This eventually leads to a discriminator network that will have learned to only model features that are present in the real data samples. The model can then be utilised to predict promoter sequences (Goodfellow *et al.*, 2014; Nicholson, 2019).

As the used promoter datasets of both *A. thaliana* and *H. sapiens* only covers Pol II promoters, future work consists of expanding these datasets by also including Pol I and Pol III promoter sequences. A deep learning network can then be constructed that learns to predict if a sample is a promoter or not (binary prediction). It is likely that such a model will produce poor prediction results due to the great variance seen in consensus sequences between these promoter classes. The different promoter classes can then be encoded as different labels instead of just the binary setting, resulting in a multiclass classification problem. This way, the network will be forced to learn meaningful data representations for each of the classes instead of trying to find - possibly non-existing - class-wide features.

As deep learning proved its merits in the application of automated DNA sequences annotation, it can also be adopted to tackle different DNA annotation problems. One of these is the prediction of enhancer regions. The identification of enhancers is especially challenging as they completely lack consensus sequences, can be found nearly anywhere in the genome, have a great variety in length, and can still function in a reverse orientation. A DNN could provide a significant boost to this problem, as the different characteristics and patterns that recede within enhancer data are extremely hard to determine.

References

Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, Corrado GS, Davis A, Dean J, Devin M, Ghemawat S, Goodfellow I, Harp A, Irving G, Isard M, Jozefowicz R, Jia Y, Kaiser L, Kudlur M, Levenberg J, *et al.* (2015) TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. http://tensorflow.org

Abeel T, Saeys Y, Bonnet E, Rouzé P, Van de Peer Y (2008) Generic eukaryotic core promoter prediction using structural features of DNA. *Genome research* **18**: 310–23

Adams C, Henke A, Gromoll J (2011) A novel two-promoter-one-gene system of the chorionic gonadotropin gene enables tissue-specific expression. *Journal of Molecular Endocrinology* **47**: 285–298

Akan P, Deloukas P (2008) DNA sequence and structural properties as predictors of human and mouse promoters. *Gene* **410**: 165–176

Al-Ajlan A, El Allali A (2018) CNN-MGP: Convolutional Neural Networks for Metagenomics Gene Prediction. *Interdisciplinary Sciences Computational Life Sciences* : 1–8

Alberts B, Johnson A, Lewis J, Raff M, Roberts K, Walter P (2014) *Molecular biology of the cell*. New York, USA: Garland Science

Ambros V (2004) The functions of animal microRNAs. *Nature* **431**: 350–355

Ambrosini G, Dreos R, Kumar S, Bucher P (2016) The ChIP-Seq tools and web server: a resource for analyzing ChIP-seq and other types of genomic data. *BMC Genomics* **17**: 938

Anderson S, Bankier AT, Barrell BG, de Bruijn MHL, Coulson AR, Drouin J, Eperon IC, Nierlich DP, Roe BA, Sanger F, Schreier PH, Smith AJH, Staden R, Young IG (1981) Sequence and organization of the human mitochondrial genome. *Nature* **290**: 457–465

Antequera F, Bird A (1993) Number of CpG islands and genes in human and mouse. *Proceedings of the National Academy of Sciences of the United States of America* **90**: 11995–9

Anwar F, Baker SM, Jabid T, Mehedi Hasan M, Shoyaib M, Khan H, Walshe R (2008) Pol II promoter prediction using characteristic 4-mer motifs: a machine learning approach. *BMC bioinformatics* **9**: 414

Ars E, Serra E, García J, Kruyer H, Gaona A, Lázaro C, Estivill X (2000) Mutations affecting mRNA splicing are the most common molecular defects in patients with neurofibromatosis type 1. *Human Molecular Genetics* **9**: 237–247

Azad AKM, Shahid S, Noman N, Lee H (2011) Prediction of plant promoters based on hexamers and random triplet pair analysis. *Algorithms for molecular biology AMB* **6**: 19

Bajic VB, Tan SL, Suzuki Y, Sugano S (2004) Promoter prediction analysis on the whole human genome. *Nature Biotechnology* **22**: 1467–1473

Ball P (2006) Smallest genome clocks in at 182 genes. *newsnature*

Bashirullah A, Cooperstock RL, Lipshitz HD (2001) Spatial and temporal control of RNA stability. *Proceedings of the National Academy of Sciences of the United States of America* **98**: 7025–8

Batut P, Dobin A, Plessy C, Carninci P, Gingeras TR (2013) High-fidelity promoter profiling reveals widespread alternative promoter usage and transposon-driven developmental gene expression. *Genome Research* **23**: 169–180

Beam AL (2017) Deep Learning 101 - Part 1: History and Background. https://beamandrew.github.io/deeplearning/2017/02/23/deep_learning_101_part1.html

Bengio Y (2009) Learning Deep Architectures for AI. Technical report

Berardini TZ, Reiser L, Li D, Mezheritsky Y, Muller R, Strait E, Huala E (2015) The arabidopsis information resource: Making and mining the "gold standard" annotated reference plant genome. *genesis* **53**: 474–485

Bharanikumar R, Premkumar KAR, Palaniappan A (2018) PromoterPredict: sequence-based modelling of Escherichia coli σ 70 promoter strength yields logarithmic dependence between promoter strength and sequence. *PeerJ* **6**: e5862

Bishop CM (2006) *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag Berlin

Breathnach R, Benoist C, O'Hare K, Gannon F, Chambon P (1978) Ovalbumin gene: evidence for a leader sequence in mRNA and DNA sequences at the exon-intron boundaries. *Proceedings of the National Academy of Sciences of the United States of America* **75**: 4853–7

Breathnach R, Chambon P (1981) Organization and Expression of Eucaryotic Split Genes Coding for Proteins. *Annual Review of Biochemistry* **50**: 349–383

Brendel V, Xing L, Zhu W (2004) Gene structure prediction from consensus spliced alignment of multiple ESTs matching the same genomic locus. *Bioinformatics* **20**: 1157–1169

Brenner S, Miller JH, Broughton WWJ (2002) *Encyclopedia of genetics*. Los Angeles, USA: Academic Press

Brownlee J (2018) A Gentle Introduction to k-fold Cross-Validation. https://machinelearningmastery.com/k-fold-cross-validation/

Brunak S, Engelbrecht J, Knudsen S (1991) Prediction of human mRNA donor and acceptor sites from the DNA sequence. *Journal of molecular biology* **220**: 49–65

Butler JEF, Kadonaga JT (2002) The RNA polymerase II core promoter: a key component in the regulation of gene expression. *Genes development* **16**: 2583–92

Canard B, Sarfati RS (1994) DNA polymerase fluorescent substrates with reversible 3'-tags. *Gene* **148**: 1–6

Carter R, Drouin G (2009) Structural differentiation of the three eukaryotic RNA polymerases. *Genomics* **94**: 388–396

Cavin Perier R, Junier T, Bucher P (1998) The Eukaryotic Promoter Database EPD. *Nucleic Acids Research* **26**: 353–357

Chollet F (2015) Keras. https://keras.io/

Chollet F (2017) *Deep learning with Python*

Cindy MC (2007) Gene Regulation in Eukaryotes. Technical report

Clancy S (2008a) DNA transcription. *Nature Education* **1**: 41

Clancy S (2008b) RNA splicing: introns, exons and spliceosome. *Nature Education* **1**: 31

Clancy S, Brown W (2008) Translation: DNA to mRNA to Protein. *Nature Education* **1**: 101

CLC bio (2019) Calculation of sequence logos. http://resources.qiagenbioinformatics.com/manuals/clcgenomicsworkbench/650/index.php?manual=Calculation_sequence_logos.html

Cong L, Ran FA, Cox D, Lin S, Barretto R, Habib N, Hsu PD, Wu X, Jiang W, Marraffini LA, Zhang F (2013) Multiplex genome engineering using CRISPR/Cas systems. *Science New York NY* **339**: 819–23

Conner JK, Hartl DL (2004) *A Primer of Ecological Genetics*. 1. Massachusetts, USA: Sinauer Associates

Crooks GE, Hon G, Chandonia JM, Brenner SE (2004) WebLogo: A Sequence Logo Generator. *Genome Research* **14**: 1188–1190

Cumbie JS, Ivanchenko MG, Megraw M (2015) NanoCAGE-XL and CapFilter: an approach to genome wide identification of high confidence transcription start sites. *BMC Genomics* **16**: 597

Daguenet E, Dujardin G, Valcárcel J (2015) The pathogenicity of splicing defects: mechanistic insights into pre-mRNA processing inform novel therapeutic approaches. *EMBO reports* **16**: 1640–55

Davis CA, Hitz BC, Sloan CA, Chan ET, Davidson JM, Gabdank I, Hilton JA, Jain K, Baymuradov UK, Narayanan AK, Onate KC, Graham K, Miyasato SR, Dreszer TR, Strattan JS, Jolanki O, Tanaka FY, Cherry JM (2018) The Encyclopedia of DNA elements (ENCODE): data portal update. *Nucleic acids research* **46**: D794–D801

Davuluri RV, Grosse I, Zhang MQ (2001) Computational identification of promoters and first exons in the human genome. *Nature Genetics* **29**: 412–417

de Avila e Silva S, Echeverrigaray S, Gerhardt GJ (2011) BacPP: Bacterial promoter prediction — A tool for accurate sigma-factor specific assignment in enterobacteria. *Journal of Theoretical Biology* **287**: 92–99

de Farias ST, do Rêgo TG, José MV (2014) Evolution of transfer RNA and the origin of the translation system. *Frontiers in genetics* **5**: 303

Degroeve S, Saeys Y, De Baets B, Rouzé P, Van de Peer Y (2005) SpliceMachine: predicting splice sites from high-dimensional local context representations. *Bioinformatics Oxford England* **21**: 1332–8

Demeler B, Zhou G (1991) Neural network optimization for E.coli promoter prediction. Technical Report 7

Deng W, Roberts SGE (2006) Core promoter elements recognized by transcription factor IIB. *Biochemical Society transactions* **34**: 1051–3

Desmet FO, Hamroun D, Lalande M, Collod-Béroud G, Claustres M, Béroud C (2009) Human Splicing Finder: an online bioinformatics tool to predict splicing signals. *Nucleic Acids Research* **37**: e67–e67

Djebali S, Davis CA, Merkel A, Dobin A, Lassmann T, Mortazavi A, Tanzer A, Lagarde J, Lin W, Schlesinger F, Xue C, Marinov GK, Khatun J, Williams BA, Zaleski C, Rozowsky J, Röder M, Kokocinski F, Abdelhamid RF, Alioto T, *et al.* (2012) Landscape of transcription in human cells. *Nature* **489**: 101–108

Dogan RI, Getoor L, Wilbur WJ, Mount SM (2007) SplicePort–an interactive splice-site analysis tool. *Nucleic acids research* **35**: 285–91

Dreos R, Ambrosini G, Cavin Périer R, Bucher P (2013) EPD and EPDnew, high-quality promoter resources in the next-generation sequencing era. *Nucleic Acids Research* **41**: D157–D164

Dror G, Sorek R, Shamir R (2005) Accurate identification of alternatively spliced exons using support vector machine. *Bioinformatics* **21**: 897–901

EnsemblPlants (2019) Arabidopsis thaliana. http://plants.ensembl.org/Arabidopsis_thaliana/Info/Index

European Bioinformatics Institute (2019) Gene Ontology and GO Annotations. https://www.ebi.ac.uk/QuickGO/

Everett RD, Baty D, Chambon P (1983) The repeated GC-rich motifs upstream from the TATA box are important elements of the SV40 early promoter. *Nucleic acids research* **11**: 2447–64

FANTOM Consortium and the RIKEN PMI and CLST (DGT), Forrest ARR, Kawaji H, Rehli M, Baillie JK, de Hoon MJL, Haberle V, Lassmann T, Kulakovskiy IV, Lizio M, Itoh M, Andersson R, Mungall CJ, Meehan TF, Schmeier S, Bertin N, Jørgensen M, Dimont E, Arner E, Schmidl C, *et al.* (2014) A promoter-level mammalian expression atlas. *Nature* **507**: 462–470

Ferrão LFV, Ferrão RG, Ferrão MAG, Fonseca A, Carbonetto P, Stephens M, Garcia AAF (2019) Accurate genomic prediction of Coffea canephora in multiple environments using whole-genome statistical models. *Heredity* **122**: 261–275

Fickett JW, Hatzigeorgiou AG (1997) Eukaryotic promoter recognition. *Genome research* **7**: 861–78

Florea L, Hartzell G, Zhang Z, Rubin GM, Miller W (1998) A computer program for aligning a cDNA sequence with a genomic DNA sequence. *Genome research* **8**: 967–74

Fredriksson NJ, Ny L, Nilsson JA, Larsson E (2014) Systematic analysis of noncoding somatic mutations and gene expression alterations across 14 tumor types. *Nature Genetics* **46**: 1258–1263

Genetic Science Learning Center (2016) RNA: The Versatile Molecule. http://learn.genetics.utah.edu/content/basics/rna/

Ghosh A, Bansal M (2003) A glossary of DNA structures from A to Z. *Acta crystallographica Section D Biological crystallography* **59**: 620–6

Goodfellow IJ, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A, Bengio Y (2014) Generative Adversarial Nets. Technical report

Google Developers (2018) Validation: Another Partition | Machine Learning Crash Course | Google Developers. https://developers.google.com/machine-learning/crash-course/validation/another-partition

Gusmao EG, de Souto MCP (2014) Issues on sampling negative examples for predicting prokaryotic promoters. In *2014 International Joint Conference on Neural Networks (IJCNN)*. IEEE

Harris NL, Senapathyl P (1990) Distribution and consensus of branch point signals in eukaryotic genes: a computerized statistical analysis. *Nucleic Acids Research* **18**

He W, Jia C, Duan Y, Zou Q (2018) 70ProPred: a predictor for discovering sigma70 promoters based on combining multiple features. *BMC Systems Biology* **12**: 44

Heather JM, Chain B (2016) The sequence of sequencers: The history of sequencing DNA. *Genomics* **107**: 1–8

Hebsgaard SM, Korning PG, Tolstrup N, Engelbrecht J, Rouzé P, Brunak S (1996) Splice site prediction in Arabidopsis thaliana pre-mRNA by combining local and global sequence information. *Nucleic acids research* **24**: 3439–52

Hickman MJ, Jackson A, Smith A, Thornton J, Tursi A (2018) Gene function contributes to gene expression levels in S. cerevisiae. *bioRxiv* : 330365

Ho S (2017) Interpretability of Neural Networks. https://datawarrior.wordpress.com/2017/10/31/interpretability-of-neural-networks/

Horton PB, Kanehisa M (1992) An assessment of neural network and statistical approaches for prediction of E.coli promoter sites. Technical Report 16

Hutchinson GB (1996) The prediction of vertebrate promoter regions using differential hexamer frequency analysis. *Computer applications in the biosciences CABIOS* **12**: 391–8

Inoue M, Horimoto K (2017) Relationship between regulatory pattern of gene expression level and gene function. *PLOS ONE* **12**: e0177430

IUPAC-IUB Joint Commission on Biochemical Nomenclature (1990) Designation of the two strands of DNA. *Biochemical Education* **18**: 135

Jain M, Olsen HE, Paten B, Akeson M (2016) The Oxford Nanopore MinION: delivery of nanopore sequencing to the genomics community. *Genome Biology* **17**: 239

Jang YJ, LaBella AL, Feeney TP, Braverman N, Tuchman M, Morizono H, Ah Mew N, Caldovic L (2018) Disease-causing mutations in the promoter and enhancer of the ornithine transcarbamylase gene. *Human Mutation* **39**: 527–536

Jones N (2014) Computer science: The learning machines. *Nature* **505**: 146–148

Juven-Gershon T, Kadonaga JT (2010) Regulation of gene expression via the core promoter and the basal transcriptional machinery. *Developmental Biology* **339**: 225–229

Kadonaga JT (2002) The DPE, a core promoter element for transcription by RNA polymerase II Regulation of Transcription by RNA Poly-merase II. Technical Report 4

Kalate RN, Tambe SS, Kulkarni BD (2003) Artificial neural networks for prediction of mycobacterial promoter sequences. *Computational Biology and Chemistry* **27**: 555–564

Karn U (2016) An Intuitive Explanation of Convolutional Neural Networks. https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

Karolchik D, Hinrichs AS, Furey TS, Roskin KM, Sugnet CW, Haussler D, Kent WJ (2004) The UCSC Table Browser data retrieval tool. *Nucleic Acids Research* **32**: 493D–496

Karpathy A, Li FF, Johnson J (2016) Stanford University CS231n: Convolutional Neural Networks for Visual Recognition. http://cs231n.stanford.edu/2016/

Kellis M, Wold B, Snyder MP, Bernstein BE, Kundaje A, Marinov GK, Ward LD, Birney E, Crawford GE, Dekker J, Dunham I, Elnitski LL, Farnham PJ, Feingold EA, Gerstein M, Giddings MC, Gilbert DM, Gingeras TR, Green ED, Guigo R, *et al.* (2014) Defining functional DNA elements in the human genome. *Proceedings of the National Academy of Sciences of the United States of America* **111**: 6131–8

Kent WJ, Sugnet CW, Furey TS, Roskin KM, Pringle TH, Zahler AM, Haussler D (2002) The human genome browser at UCSC. *Genome research* **12**: 996–1006

Kim TH, Barrera LO, Zheng M, Qu C, Singer MA, Richmond TA, Wu Y, Green RD, Ren B (2005) A high-resolution map of active promoters in the human genome. *Nature* **436**: 876–80

Knudsen S (1999) Promoter2.0: for the recognition of PolII promoter sequences. *Bioinformatics* **15**: 356–361

Kolovos P, Knoch TA, Grosveld FG, Cook PR, Papantonis A (2012) Enhancers and silencers: an integrated and simple model for their function. *Epigenetics chromatin* **5**: 1

Koza JR, Bennett FH, Andre D, Keane MA (1996) Automated Design of Both the Topology and Sizing of Analog Electrical Circuits Using Genetic Programming, In *Artificial Intelligence in Design '96*, Dordrecht: Springer Netherlands, pp. 151–170

Krizhevsky A, Sutskever I, Hinton GE (2012) ImageNet Classification with Deep Convolutional Neural Networks. Technical report

Kudo M, Iida Y, Shimbo M (1987) Syntactic pattern analysis of 5'-splice site sequences of mRNA precursors in higher eukaryote genes. *Computer applications in the biosciences CABIOS* **3**: 319–24

Lai HY, Zhang ZY, Su ZD, Su W, Ding H, Chen W, Lin H (2019) iProEP: A Computational Predictor for Predicting Promoter. *Molecular therapy Nucleic acids* **17**: 337–346

Lapedes A, Barnes C, Burks C, Farber R, Sirotkin K (1988) Application of neural networks and other machine learning algorithms to DNA sequence analysis. In *Interface between computational science and nucleic acid sequencing*

Larsen NI, Engelbrecht J, Brunak S (1995) Analysis of eukaryotic promoter sequences reveals a systematically occurring CT-signal. *Nucleic acids research* **23**: 1223–30

Lee MP, Howcroft K, Kotekar A, Yang HH, Buetow KH, Singer DS (2005) ATG deserts define a novel core promoter subclass. *Genome research* **15**: 1189–97

Lee TY, Chang WC, Hsu J, Chang TH, Shien DM (2012) GPMiner: an integrated system for mining combinatorial cis-regulatory elements in mammalian gene group. *BMC Genomics* **13**: S3

Lenhard B, Sandelin A, Carninci P (2012) Metazoan promoters: emerging characteristics and insights into transcriptional regulation. *Nature Reviews Genetics* **13**: 233–245

Li J, Wong L (2003) Using Rules to Analyse Bio-medical Data: A Comparison between C4.5 and PCL, Springer, Berlin, Heidelberg, pp. 254–265

Li L, Wunderlich Z (2017) An Enhancer's Length and Composition Are Shaped by Its Regulatory Task. *Frontiers in genetics* **8**: 63

Lim CY, Santoso B, Boulay T, Dong E, Ohler U, Kadonaga JT (2004) The MTE, a new core promoter element for transcription by RNA polymerase II. *Genes development* **18**: 1606–17

Liu B, Yang F, Huang DS, Chou KC (2018) iPromoter-2L: a two-layer predictor for identifying promoters and their types by multi-window-based PseKNC. *Bioinformatics* **34**: 33–40

Liu L, Ho YK, Yau S (2007) Prediction of Primate Splice Site Using Inhomogeneous Markov Chain and Neural Network. *DNA and Cell Biology* **26**: 477–483

Lizio M, Harshbarger J, Shimoji H, Severin J, Kasukawa T, Sahin S, Abugessaisa I, Fukuda S, Hori F, Ishikawa-Kato S, Mungall CJ, Arner E, Baillie J, Bertin N, Bono H, de Hoon M, Diehl AD, Dimont E, Freeman TC, Fujieda K, *et al.* (2015) Gateways to the FANTOM5 promoter level mammalian expression atlas. *Genome Biology* **16**: 22

Lodish H, Berk A, Zipursky S L, Matsudaira P, Baltimore D, Darnell J (2000) Regulatory Sequences in Eukaryotic Protein-Coding Genes, In *Molecular Cell Biology*, chapter 10.4, New York, USA: W.H. Freeman, 4 edition

Lodish HF (2008) *Molecular cell biology*. New York, USA: W.H. Freeman

Lynch M (2006) The Origins of Eukaryotic Gene Structure. *Molecular Biology and Evolution* **23**: 450–468

Marashi SA, Goodarzi H, Sadeghi M, Eslahchi C, Pezeshk H (2006) Importance of RNA secondary structure information for yeast donor and acceptor splice site predictions by neural networks. *Computational Biology and Chemistry* **30**: 50–57

Maston GA, Evans SK, Green MR (2006) Transcriptional Regulatory Elements in the Human Genome

Matera AG, Wang Z (2014) A day in the life of the spliceosome. *Nature Reviews Molecular Cell Biology* **15**: 108–121

Matis S, Xu Y, Shah M, Guan X, Einstein JR, Mural R, Uberbacher E (1996) Detection of RNA polymerase II promoters and polyadenylation sites in human DNA sequence. *Computers chemistry* **20**: 135–40

Meher PK, Sahu TK, Rao A, Wahi S (2016a) A computational approach for prediction of donor splice sites with improved accuracy. *Journal of Theoretical Biology* **404**: 285–294

Meher PK, Sahu TK, Rao AR (2016b) Prediction of donor splice sites using random forest with a new sequence encoding approach. *BioData Mining* **9**: 4

Meher PK, Sahu TK, Rao AR, Wahi SD (2016c) Identification of donor splice sites using support vector machine: a computational approach based on positional, compositional and dependency features. *Algorithms for molecular biology AMB* **11**: 16

Mignone F, Gissi C, Liuni S, Pesole G (2002) Untranslated regions of mRNAs. *Genome biology* **3**: REVIEWS0004

Morton T, Petricka J, Corcoran DL, Li S, Winter CM, Carda A, Benfey PN, Ohler U, Megraw M (2014) Paired-End Analysis of Transcription Start Sites in Arabidopsis Reveals Plant-Specific Promoter Signatures. *The Plant Cell* **26**: 2746–2760

Mulligan ME, Mcclure WR (1986) Analysis of the occurrence of promoter-sites in DNA. Technical report

Murakami T, Nishiyama T, Shirotani T, Shinohara Y, Kan M, Ishii K, Kanai F, Nakazuru S, Ebina Y (1992) Identification of two enhancer elements in the gene encoding the type 1 glucose transporter from the mouse which are responsive to serum, growth factor, and oncogenes. *The Journal of biological chemistry* **267**: 9300–6

Naito T (2018) Human Splice-Site Prediction with Deep Neural Networks. *Journal of Computational Biology* **25**: 954–961

National Center for Biotechnology Information (1993) Conserved Sequence. https://www.ncbi.nlm.nih.gov/mesh?Db=mesh&term=Conserved+Sequence

National Center for Biotechnology Information (2011) Arabidopsis thaliana (thale cress). *British journal of pharmacology*

National Center for Biotechnology Information (2018) Home - Gene - NCBI. https://www.ncbi.nlm.nih.gov/gene/

National Center for Biotechnology Information (2019) Homo sapiens (ID 51) - Genome. https://www.ncbi.nlm.nih.gov/genome/?term=homo+sapiens

National Health Service (2017) Reference Genome: Defining Human Difference - Genomics Education Programme. https://www.genomicseducation.hee.nhs.uk/news/item/328-reference-genome-defining-human-difference/

National Institutes of Health (2017) What is DNA? https://ghr.nlm.nih.gov/primer/basics/dna

National Public Radio (2019) In A 1st, Doctors In U.S. Use CRISPR Tool To Treat Patient With Genetic Disorder. https://www.npr.org/sections/health-shots/2019/07/29/744826505/sickle-cell-patient-reveals-why-she-is-volunteering-for-landmark-gene-editing-st?t=1565380027752

Nature Education (2014) Allele. https://www.nature.com/scitable/definition/allele-48

Ng A (2018) What is Machine Learning? - Introduction | Coursera. https://www.coursera.org/lecture/machine-learning/what-is-machine-learning-Ujm7v

Ng A (2019) Neural Networks and Deep Learning | Coursera. https://www.coursera.org/learn/neural-networks-deep-learning

Ng A, Katanforoosh K (2018) CS230 Deep Learning. http://cs230.stanford.edu/

Nguyen NG, Tran VA, Ngo DL, Phan D, Lumbanraja FR, Faisal MR, Abapihi B, Kubo M, Satou K (2016) DNA Sequence Classification by Convolutional Neural Network. *Journal of Biomedical Science and Engineering* **09**: 280–286

Nicholson C (2019) A.I. Wiki: A Beginner's Guide to Important Topics in AI, Machine Learning, and Deep Learning. https://skymind.ai/wiki/generative-adversarial-network-gan

Nomenclature Committee of the International Union of Biochemistry (1984) Nomenclature for Incompletely Specified Bases

in Nucleic Acid Sequences. http://www.sbcs.qmul.ac.uk/iubmb/misc/naseq.html

Ohler U (2000) Promoter prediction on a genomic scale–the Adh experience. *Genome research* **10**: 539–42

Ohshima Y, Gotoh Y (1987) Signals for the selection of a splice site in pre-mRNA: Computer analysis of splice junction sequences and like sequences. *Journal of molecular biology* **195**: 247–59

O'Leary NA, Wright MW, Brister JR, Ciufo S, Haddad D, McVeigh R, Rajput B, Robbertse B, Smith-White B, Ako-Adjei D, Astashyn A, Badretdin A, Bao Y, Blinkova O, Brover V, Chetvernin V, Choi J, Cox E, Ermolaeva O, Farrell CM, *et al.* (2016) Reference sequence (RefSeq) database at NCBI: current status, taxonomic expansion, and functional annotation. *Nucleic Acids Research* **44**: D733–D745

Oliphant T (2006) NumPy: A guide to NumPy. http://www.numpy.org

O'Neill MC, Chiafari F (1989) Escherichia coli promoters. II. A spacing class-dependent promoter search protocol. *The Journal of biological chemistry* **264**: 5531–4

Oubounyt M, Louadi Z, Tayara H, Chong KT (2019) DeePromoter: Robust Promoter Predictor Using Deep Learning. *Frontiers in Genetics* **10**: 286

Patel AA, Steitz JA (2003) Splicing double: insights from the second spliceosome. *Nature Reviews Molecular Cell Biology* **4**: 960–970

Paule MR, White RJ (2000) Transcription by RNA polymerases I and III. *Nucleic acids research* **28**: 1283–98

Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E (2011) Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* **12**: 2825–2830

Pellicer JF, Michael F. Leitch, Ilia J. (2010) The largest eukaryotic genome of them all? *Botanical Journal of the Linnean Society* **164**: 10–15

Pennacchio LA, Bickmore W, Dean A, Nobrega MA, Bejerano G (2013) Enhancers: five essential questions. *Nature reviews Genetics* **14**: 288–95

Pertea M, Lin X, Salzberg SL (2001) GeneSplicer: a new computational method for splice site prediction. Technical Report 5

Pierce BA (2012) *Genetics : a conceptual approach*. New York, USA: W.H. Freeman

Ponting CP, Hardison RC (2011) What fraction of the human genome is functional? *Genome research* **21**: 1769–76

Pray L (2008) Discovery of DNA structure and function: Watson and Crick. *Nature Education* **1**: 100

Prestridge DS (1995) Predicting Pol II Promoter Sequences using Transcription Factor Binding Sites. *Journal of Molecular Biology* **249**: 923–932

Ratcliffe P (2015) The DNA Double Helix. http://pratclif.com/biologie-moleculaire/dna/helix.html

Reed R, Maniatis T (1985) Intron sequences involved in lariat formation during pre-mRNA splicing. *Cell* **41**: 95–105

Reese MG (2001) Application of a time-delay neural network to promoter annotation in the Drosophila melanogaster genome. *Computers chemistry* **26**: 51–6

Reese MG, Eeckman FH, Kulp D, Haussler D (1997) Improved Splice Site Detection in Genie. *Journal of Computational Biology* **4**: 311–323

Roeder RG (1996) The role of general initiation factors in transcription by RNA polymerase II. *Trends in Biochemical Sciences* **21**: 327–335

Roy AL, Singer DS (2015) Core promoters in transcription: old problem, new insights. *Trends in biochemical sciences* **40**: 165–71

Saenger W (1984) *Principles of Nucleic Acid Structure*. Springer Advanced Texts in Chemistry. New York, USA: Springer New York

Sanger F, Nicklen S, Coulson AR (1977) DNA sequencing with chain-terminating inhibitors. *Proceedings of the National Academy of Sciences of the United States of America* **74**: 5463–7

Scherf M, Klingenhoff A, Werner T (2000) Highly specific localization of promoter regions in large genomic sequences by PromoterInspector: a novel context analysis approach. *Journal of Molecular Biology* **297**: 599–606

Schneider TD, Stephens RM (1990) Sequence logos: a new way to display consensus sequences. Technical Report 20

Shafee T, Lowe R, Lowe R (2017) Eukaryotic and prokaryotic gene structure. *WikiJournal of Medicine* **4**: 2

Shahmuradov IA, Solovyev VV, Gammerman AJ (2005) Plant promoter prediction with confidence estimation. *Nucleic acids research* **33**: 1069–76

Shahmuradov IA, Umarov RK, Solovyev VV (2017) TSSPlant: a new tool for prediction of plant Pol II promoters. *Nucleic Acids Research* **45**: gkw1353

Sievers F, Higgins DG (2018) Clustal Omega for making accurate alignments of many protein sequences. *Protein Science* **27**: 135–145

Sloutskin A, Danino YM, Orenstein Y, Zehavi Y, Doniger T, Shamir R, Juven-Gershon T (2015) ElemeNT: a computational tool for detecting core promoter elements. *Transcription* **6**: 41–50

Smale ST, Kadonaga JT (2003) The RNA Polymerase II Core Promoter. *Annual Review of Biochemistry* **72**: 449–479

Solovyev V (2008) Statistical Approaches in Eukaryotic Gene Prediction, In *Handbook of Statistical Genetics*, Chichester, UK: John Wiley & Sons, Ltd, pp. 97–159

Solovyev V, Salamov A (2010) *Automatic annotation of microbial genomes and metagenomic sequences*

Solovyev V, Shahmuradov IA (2003) PromH: promoters identification using orthologous genomic sequences. *Nucleic Acids Research* **31**: 3540–3545

Solovyev VV, Salamov AA, Lawrence CB (1994) Predicting internal exons by oligonucleotide composition and discriminant analysis of spliceable open reading frames. *Nucleic Acids Research* **22**: 5156–5163

Solovyev VV, Shahmuradov IA, Salamov AA (2010) Identification of Promoter Regions and Regulatory Sites, In *Methods in molecular biology (Clifton, N.J.)*, volume 674, pp. 57–83

Sonnenburg S, Schweikert G, Philips P, Behr J, Rätsch G (2007) Accurate splice site prediction using support vector machines. *BMC Bioinformatics* **8**: S7

Staden R (1984) Computer methods to locate signals in nucleic acid sequences. *Nucleic acids research* **12**: 505–19

Stein L (2001) Genome annotation: from sequence to biology. *Nature Reviews Genetics* **2**: 493–503

Talwalkar A (2016) CS190.1x | Scalable Machine Learning. https://courses.edx.org/courses/BerkeleyX/CS190.1x/1T2015/576601d4282341f99ac7956718cc2301/

Tokizawa M, Kusunoki K, Koyama H, Kurotani A, Sakurai T, Suzuki Y, Sakamoto T, Kurata T, Yamamoto YY (2017) Identification of Arabidopsis genic and non-genic promoters by paired-end sequencing of TSS tags. *The Plant Journal* **90**: 587–605

Towell GG (1993) Extracting Refined Rules from Knowledge-Based Neural Networks. Technical report

Triska M, Solovyev V, Baranova A, Kel A, Tatarinova TV (2017) Nucleotide patterns aiding in prediction of eukaryotic promoters. *PLOS ONE* **12**: e0187243

Turunen JJ, Niemelä EH, Verma B, Frilander MJ (2013) The significant other: splicing by the minor spliceosome. *Wiley interdisciplinary reviews RNA* **4**: 61–76

Twyman R (2003) Gene structure. https://web.archive.org/web/20070328214808/http://genome.wellcome.ac.uk/doc_WTD020755.html

Umarov RK, Solovyev VV (2017) Recognition of prokaryotic and eukaryotic promoters using convolutional deep learning neural networks. *PLOS ONE* **12**: e0171410

Ushijima T, Hanada K, Gotoh E, Yamori W, Kodama Y, Tanaka H, Kusano M, Fukushima A, Tokizawa M, Yamamoto YY, Tada Y, Suzuki Y, Matsushita T (2017) Light Controls Protein Localization through Phytochrome-Mediated Alternative Promoter Selection. *Cell* **171**: 1316–1325

van der Velden AW, Thomas AA (1999) The role of the 5' untranslated region of an mRNA in translation regulation during development. *The international journal of biochemistry cell biology* **31**: 87–106

Wang K, Ussery DW, Brunak S (2009) Analysis and prediction of gene splice sites in four Aspergillus genomes. *Fungal Genetics and Biology* **46**: S14–S18

Wang M, Marín A (2006) Characterization and prediction of alternative splice sites. *Gene* **366**: 219–227

Wang Z, Chen Y, Li Y (2004) A brief review of computational gene prediction methods. *Genomics proteomics bioinformatics* **2**: 216–21

Watson JD, Crick FHC (1953) Molecular Structure of Nucleic Acids: A Structure for Deoxyribose Nucleic Acid. *Nature* **171**: 737–738

West AG, Gaszner M, Felsenfeld G (2002) Insulators: many functions, many mechanisms. *Genes development* **16**: 271–88

Wright JC, Mudge J, Weisser H, Barzine MP, Gonzalez JM, Brazma A, Choudhary JS, Harrow J (2016) Improving GENCODE reference gene annotation using a high-stringency proteogenomics workflow. *Nature Communications* **7**: 11778

Zhang Y, Chu CH, Chen Y, Zha H, Ji X (2006) Splice site prediction using support vector machines with a Bayes kernel. *Expert Systems with Applications* **30**: 73–81

Zuallaert J, Godin F, Kim M, Soete A, Saeys Y, De Neve W (2018) SpliceRover: interpretable convolutional neural networks for improved splice site prediction. *Bioinformatics* **34**: 4180–4188

Appendices

A. ARAsplice assembly as described by Degroeve et al.

Publicly available mRNA sequences were downloaded from the EMBL database on 5 June 2000 and aligned to the *Arabidopsis* chromosome assembly bacterial artificial chromosome (BAC) sequences using the SIM4 program (Florea *et al.*, 1998). To remove redundant genes, neighbouring genes were removed from the dataset. Genes were seen as neighbours if they shared 80% or more of their nucleotides, and excluded by counting the neighbours of every gene and removing the one with the largest number. The removal process was repeated until no genes with neighbours were present in the dataset. This gene pruning reduced the number of genes in the dataset from 1812 to 1495. The positive donor and acceptor splice site classes consist out of intron parts which comply to the GT-AG consensus sequence for U2 class introns. The negative donor splice site class consist out of all sequences for which a GT dinucleotide is found between -300 bp and +300 bp of the TSS, yet lack the full GT-AG consensus sequence. Similarly, the negative acceptor site class contains all sequences for which an AG dinucleotide is found between -300 bp and +300 bp of the translation stop site near the stop codon, and which lack the full GT-AG consensus sequence.

B. ARAprom and HOMprom assembly pipelines by EPDnew

B.1. ARAprom dataset

The source data consists of the February 2011 araTha1 genome release from the publicly available TAIR10 database (Berardini *et al.*, 2015). Genome annotation is also downloaded from the TAIR10 website (1 February 2015 version), and linked with NCBI RefSeq files (O'Leary *et al.*, 2016) to obtain the RefSeq IDs. From the original source data, 31615 promoter sequences covering 27149 different genes are selected, annotated, and saved under the SGA format for further processing (step 1 and 2 in figure B.1). The genome release data is extended with 13 extra experimental cap analysis gene expression (CAGE) samples obtained from Morton *et al.* (2014), Cumbie *et al.* (2015), Tokizawa *et al.* (2017), and Ushijima *et al.* (2017). Peak calling to obtain the location of the TSS is performed on these 13 samples, using the ChIP-Peak Analysis Module (Ambrosini *et al.*, 2016) with a window width of 1 bp, a vicinity range of 200 bp, the refinement of peak position toggled to YES, a count cut-off of 9999999, and the peak threshold read counts and relative enrichment factor both to 5 (step 3, 4, and 5 in figure B.1).
The peak-called CAGE data is then combined with the TAIR10 source data to experimentally validate the TSSs and thus the promoter sequences. An annotated TSS is confirmed as a true TSS if an experimental CAGE peak can be found in the vicinity of 100 bp around it. The validated TSS is then shifted to the position of where the highest CAGE data peak can be found (step 6 in figure B.1). This results in a separate promoter collection for each TAIR10 sample, as multiple annotated TSSs can be linked to the same CAGE peak and thus result in a slightly different position of the TSS. Each sample promoter collection is scanned for the presence of core promoter elements TATA-box and inr (step 8 in figure B.1), and used as input data for the next step of the data assembly pipeline (part B in figure B.1). TSSs that were not experimentally validated by a CAGE peak are discarded (step 7 in figure B.1)
The separate promoter collections are merged into one file, and one position for each experimentally validated TSS is chosen by taking the position that was confirmed by the largest amount of samples (step 9, 10, and 11 in figure B.1). An additional filter is then applied for TSSs that are in the vicinity of 100 bp or less of each other, and that belong to the same gene. These TSSs are merged into the promoter validated by the highest number of samples, which resulted in the experimentally validated EPDnew *A. thaliana* promoter dataset containing 22703 promoter samples (step 12 and 13 in figure B.1).

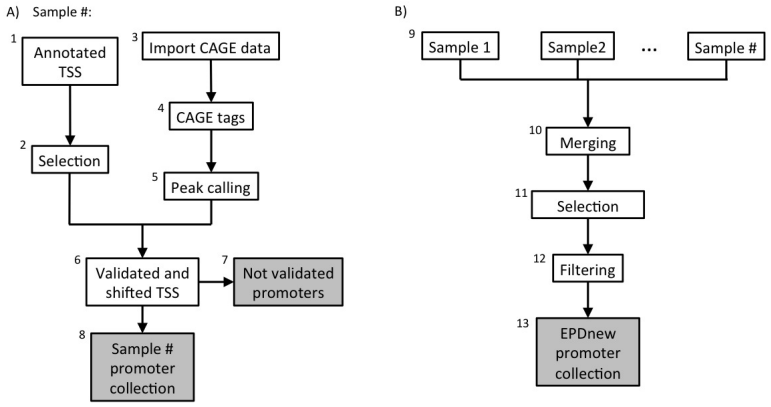


Figure B.1: Visualisation of the EPDnew assembly pipeline for the *A. thaliana* promoter dataset (Dreos *et al.*, 2013).

B.2. HOMprom dataset

Human ENCODE gene data is retrieved through the USCS Table Browser (Karolchik *et al.*, 2004) with the December 2013 (GRCh38/hg38) assembly toggled. Annotation data is downloaded from the EBI website, using the Human GENCODE v28/hg38 release (Wright *et al.*, 2016). Sequences are kept if they belong to genes coding for proteins and if all splice junctions of the gene are supported by at least one mRNA (transcript support level equal to 1). Sequences that share the same absolute TSS position are merged, resulting in 35320 extracted promoter sequences in 17056 genes which are saved in the SGA format (step 1 and 2 in figure B.2). The gene data is extended with experimental CAGE and RNA Annotation and Mapping of Promoters for the Analysis of Gene Expression (RAMPAGE) data from Djebali *et al.* (2012; 145 CAGE samples), Batut *et al.* (2013; 225 RAMPAGE samples), and FANTOM5 (2014; 941 CAGE samples). The CAGE source files in BAM format are downloaded from the USCS (Kent *et al.*, 2002) or FANTOM (Lizio *et al.*, 2015) website, and mapped onto the GRCh37/hg19 genome assembly. The liftOver tool (Kent *et al.*, 2002) is used to lift the samples over to the GRCh38/hg38 genome assembly. Peak calling with the ChIP-Peak Analysis Module (Ambrosini *et al.*, 2016) is performed on every CAGE sample to obtain the location of the TSS, with a window width of 1 bp, a vicinity range of 200 bp, the refinement of peak position toggled to NO, a count cut-off of 9999999, and the peak threshold read counts and relative enrichment factor both to 5 (step 3, 4, and 5 in figure B.2). The RAMPAGE data, obtained by paired-end sequencing, is used to identify TSS that map outside gene boundaries. The samples are downloaded through the ENCODE portal (Davis *et al.*, 2018), mapped as BAM files onto the GRCh38/hg38 genome assembly, and analysed separately from the CAGE data (step 6 in figure B.2). Samples are kept and annotated if a mapping score of 255 is achieved and if the second paired end maps inside a gene exon and has the same orientation as the first paired end. Peak calling is then performed in the same manner as for the CAGE samples, and peaks are selected for further analysis if they map outside their defined gene boundaries.

Since quality control for the extracted samples indicated poor quality, the RAMPAGE data is merged together with the ENCODE gene data for TSS validation by use of the CAGE data (step 7 in figure B.2). A ENCODE TSS is experimentally confirmed if a CAGE peak with at least 5 tags is found within a vicinity of 200 bp, while a RAMPAGE TSS is confirmed when the CAGE peak maps to the 5' UTR of the gene and has at least 50 tags. For both the ENCODE and RAMPAGE samples, the validated TSS is shifted onto the base where the highest CAGE peak was found. Secondary promoters for genes with multiple TSSs are discarded when their expression level is below 10% of the expression level of the strongest promoter (ENCODE data), or if the peak was less than 10 tags (RAMPAGE data). TSSs that failed to be experimentally validated were discarded from further analysis.

As a TSS can be validated by multiple CAGE peaks and thus have slightly different start positions, each sample in the ENCODE and RAMPAGE data generated its own promoter set. These are controlled for low quality by discarding samples with low frequencies for the TATA-box (< 5%) and the inr (< 10%) (step 8 in figure B.2). Each promoter set is then used as input for an additional processing step (part B in figure B.2), where they are all merged into one single file. A TSS is only retained when it was experimentally validated by at least three CAGE samples, and its position is fixed by taking the position that was confirmed by the largest number of samples (step 9,10 and 11 in figure B.2). Additional filtering is then done by merging promoters which share the same gene and from which the TSSs are within a 100 bp window of each other (promoter with largest sample confirmation is kept; step 12 in figure B.2). This eventually resulted in the experimentally validated EPDnew H. sapiens coding promoter collection, with 29598 promoter sequences (step 13 in figure B.2).

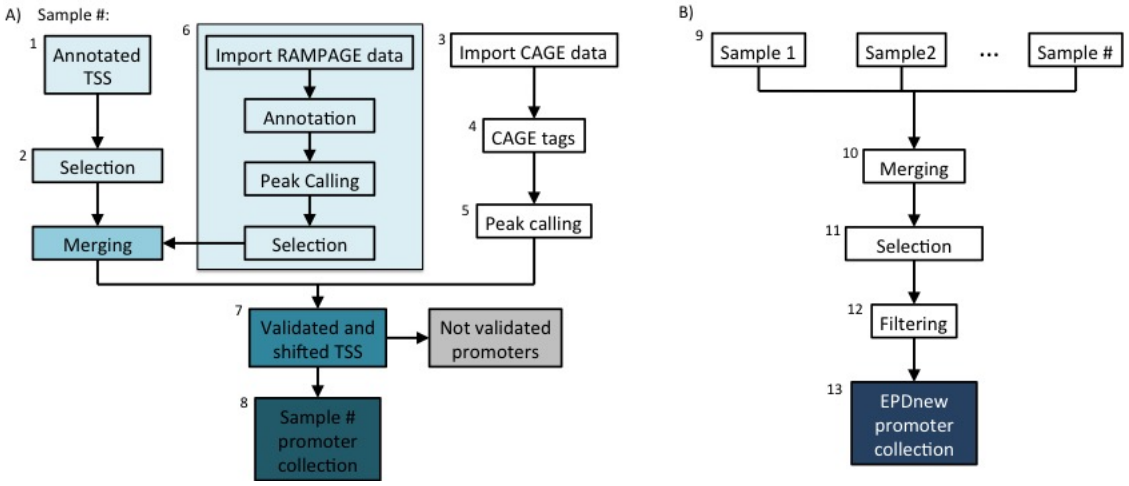


Figure B.2: Visualisation of the EPDnew assembly pipeline for the coding *H. sapiens* promoter dataset (Dreos *et al.*, 2013).

C. Python package versions

```
absl-py==0.6.1
astor==0.7.1
brewer2mpl==1.4.1
cycler==0.10.0
dill==0.2.8.2
gast==0.2.0
ggplot==0.11.5
grpcio==1.16.1
h5py==2.8.0
Keras==2.2.4
Keras-Applications==1.0.6
Keras-Preprocessing==1.0.5
kiwisolver==1.0.1
lml==0.0.7
Markdown==3.0.1
matplotlib==3.0.2
numexpr==2.6.8
numpy==1.15.4
odfpy==1.3.5
pandas==0.23.4
patsy==0.5.1
pkg-resources==0.0.0
protobuf==3.6.1
pydot==1.4.1
pyexcel-io==0.5.11
pyexcel-ods==0.5.4
pyparsing==2.3.0
python-dateutil==2.7.5
pytz==2018.7
PyYAML==3.13
scikit-learn==0.20.0
scipy==1.1.0
seaborn==0.9.0
six==1.11.0
sklearn==0.0
statsmodels==0.9.0
tables==3.4.4
tensorboard==1.12.0
tensorflow==1.12.0
termcolor==1.1.0
Werkzeug==0.14.1
```

Listing 1: Python packages and their versions used for the scripting and training of the DNNs.

D. Sequence logos and how to interpret them

A sequence logo is a visualisation of a multiple sequence alignment of proteins or DNA sequences which represents the nucleotide or amino acid conservation at each position in the provided sequences (Schneider and Stephens, 1990; Crooks *et al.*, 2004). For use with DNA sequences, a stack of bases is seen at each position where the total height R of the stack indicates how well the sequences are conserved at that location. This height is represented in bits of entropy, with the maximum value of 2 being found at positions where always the same base is encountered (CLC bio, 2019). Within a stack for a certain position, the heights of the letters relate to the relative frequency of each letter at that location. The stack height R_i at position i is calculated using:

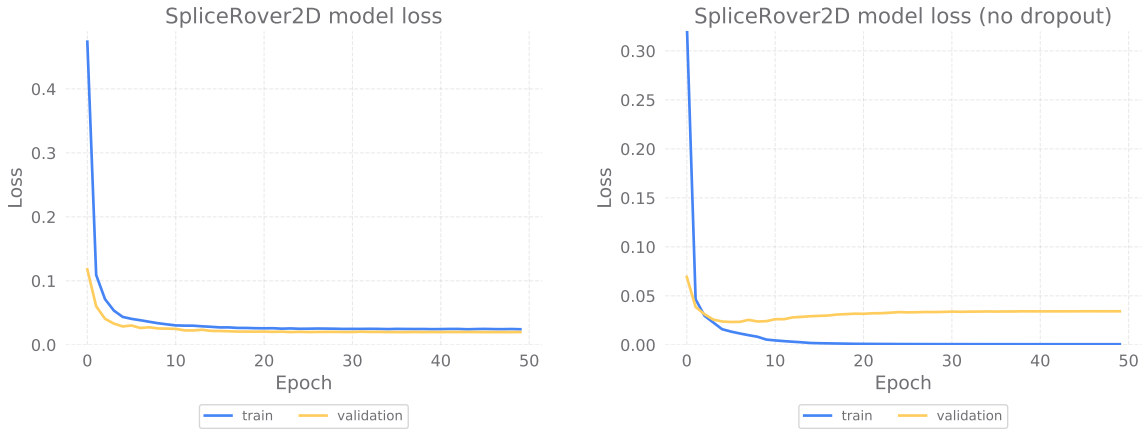
$$R_i = 2 - [H_i + e_n] \tag{1}$$

with H_i the Shannon entropy or uncertainty at position i and e_n a small-sample correction. They are determined by:

$$H_i = -\sum_{b \in \mathsf{B}} f_{b,i} \cdot log_2(f_{b,i}) \qquad e_n = \frac{1}{\ln(2)} \cdot \frac{3}{2n} \tag{2, 3}$$

where B = {A, C, G, T}, $f_{b,i}$ the relative frequency of base b at position i, and n the number of sequences in the alignment. Note that e_n can be discarded for large numbers of n.

E. Effect of dropout on the loss during training



(a) Original SpliceRover2D

(b) SpliceRover2D without dropout layers

Figure E.1: Loss plots for the SpliceRover2D model run with a stratified split on the original ARAsplice set.

F. K-mer histograms promoter datasets

F.1. ARAPROM: conserved

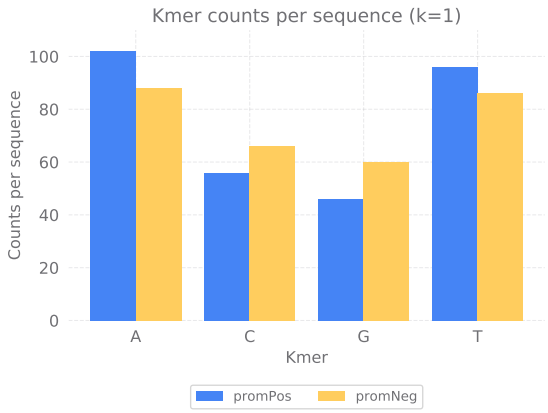
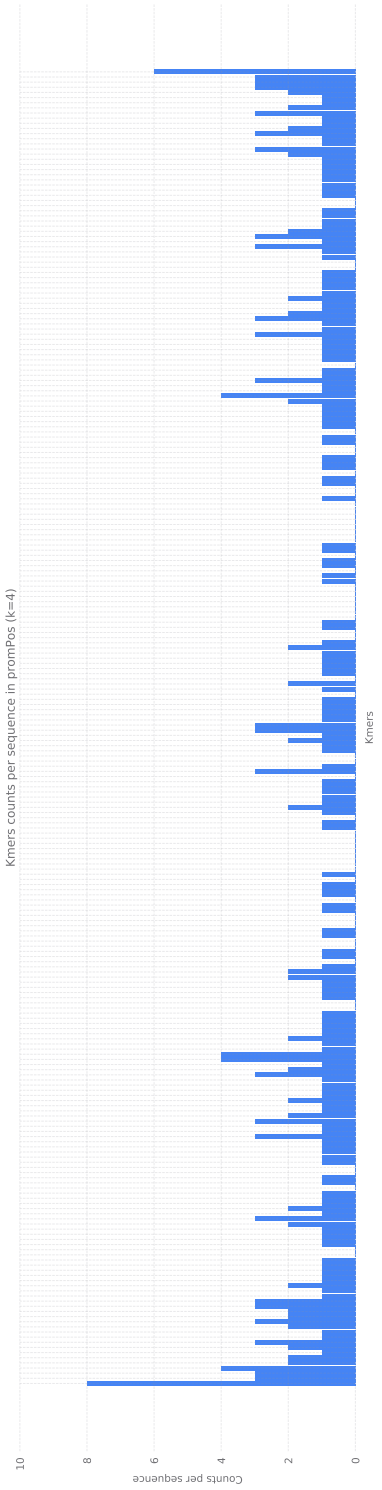
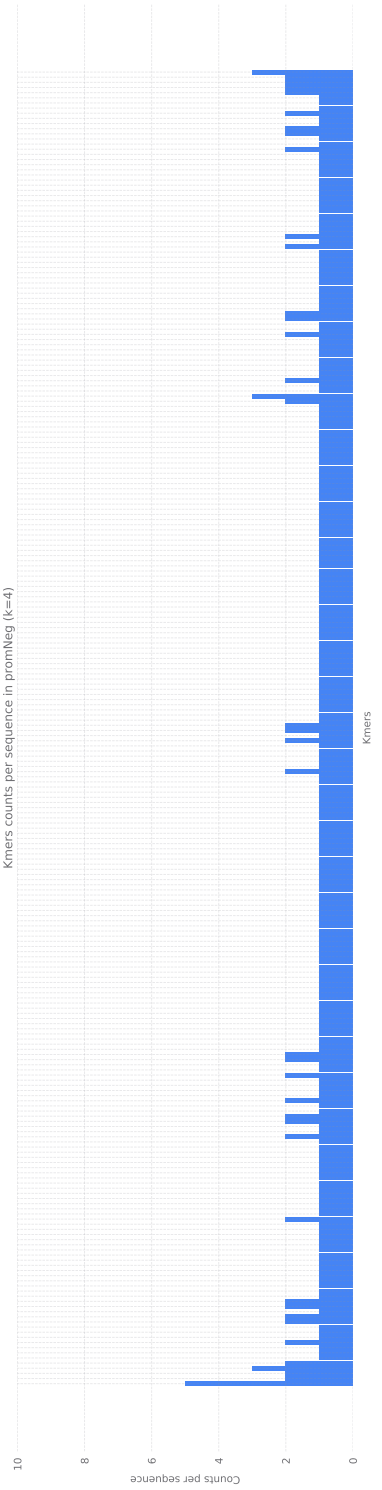


Figure F.1: K-mer counts per sequence for both classes within the ARAprom-C dataset for $k = 1$.



(a) Positive promoters (promPos)



(b) Negative promoters (promNeg)

Figure F.2: K-mer counts per sequence for both classes within the ARAprom-C dataset for $k = 4$.

F.2. HOMPROM: balanced

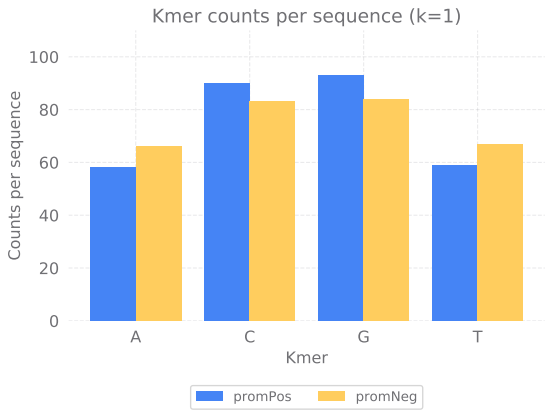
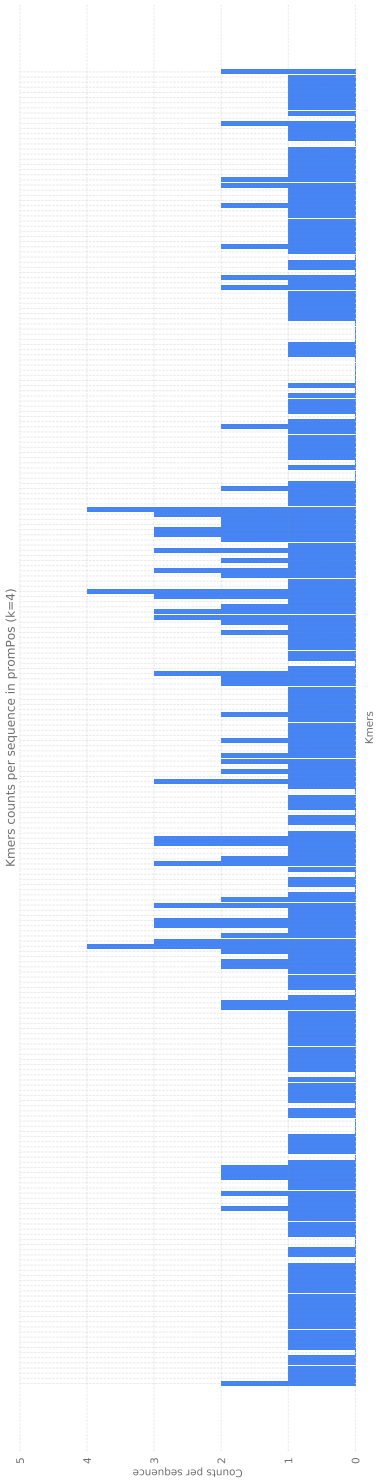
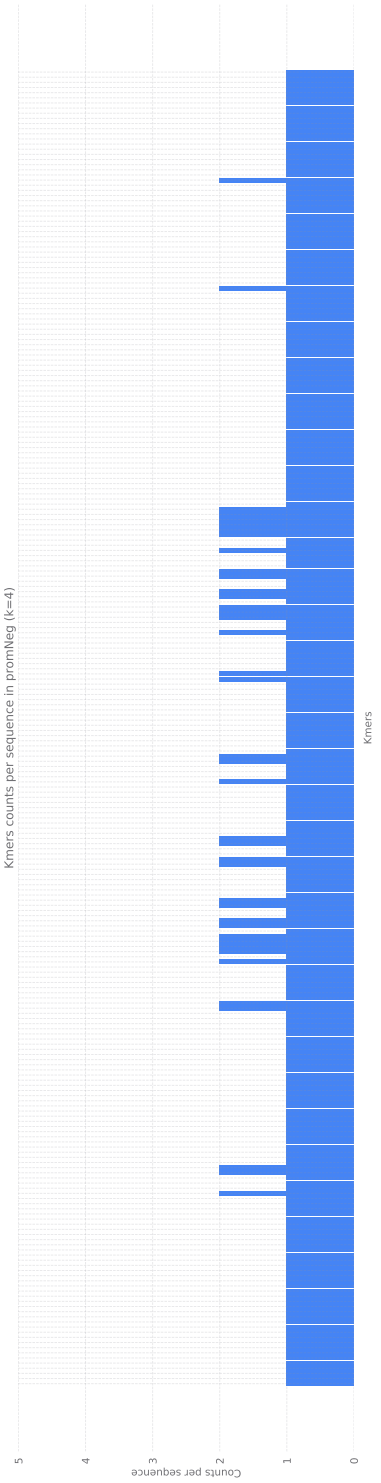


Figure F.3: K-mer counts per sequence for both classes within the HOMprom-B dataset for $k = 1$.



(a) Positive promoters (promPos)



(b) Negative promoters (promNeg)

Figure F.4: K-mer counts per sequence for both classes within the HOMprom-B dataset for $k = 4$.

F.3. HOMPROM: conserved

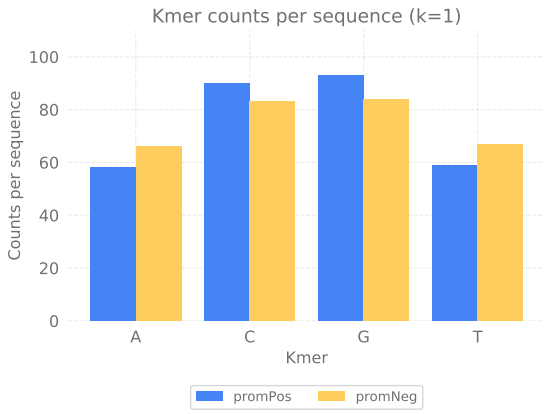
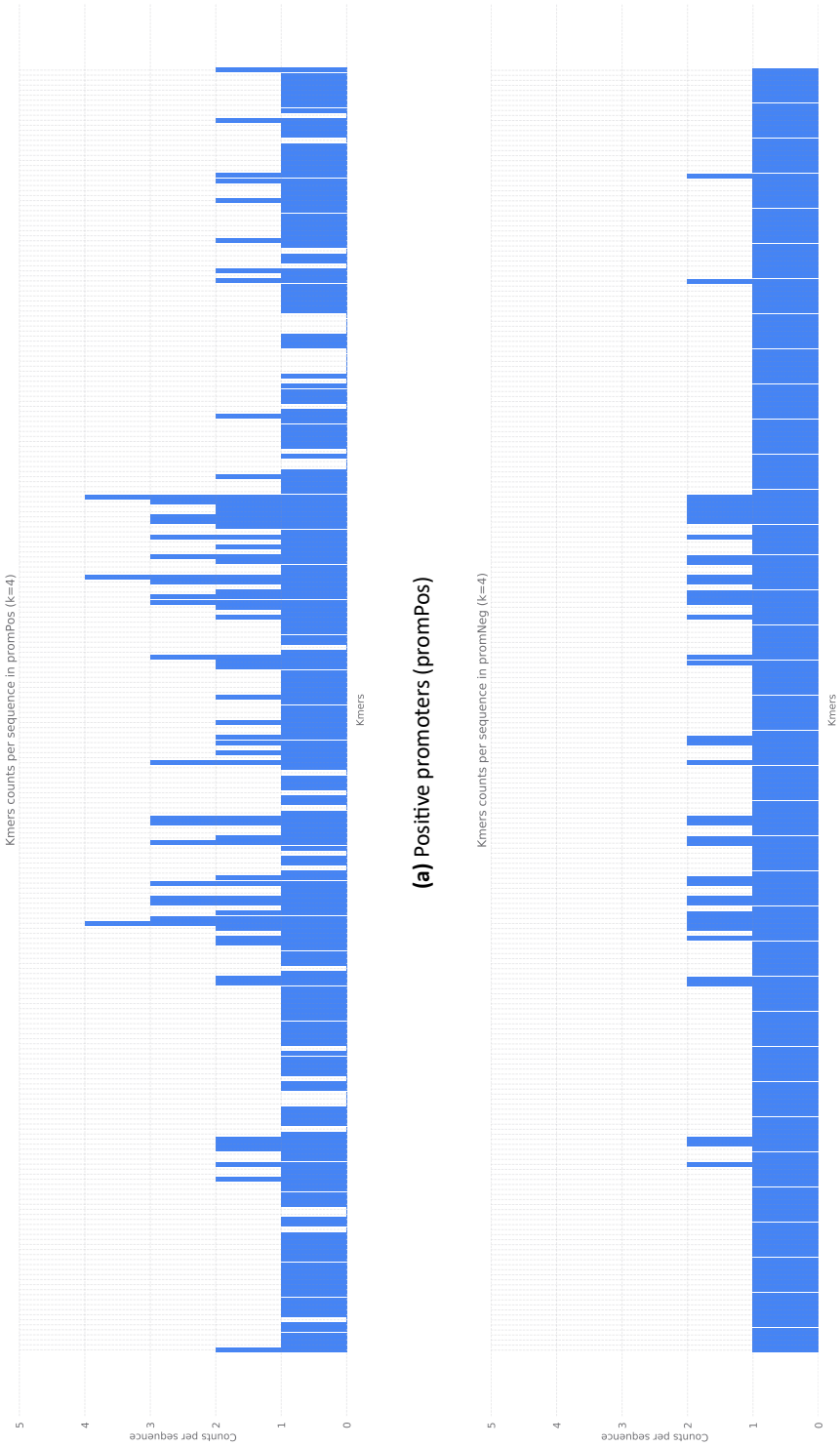


Figure F.5: K-mer counts per sequence for both classes within the HOMprom-C dataset for $k = 1$.



(a) Positive promoters (promPos)

(b) Negative promoters (promNeg)

Figure F.6: K-mer counts per sequence for both classes within the HOMprom-C dataset for $k = 4$.

G. Negative promoter construction: unbalanced approach

The unbalanced approach adopts several negative sample construction methods proposed by Gusmao and de Souto (2014). All methods are used on the same dataset at once, creating a dataset with a class imbalance in favour of the negative samples. This approach establishes a dataset that is more in line with the conditions in real-life genomes, where a promoter sequence is encountered every 30 to 40 kilobases (in humans; Antequera and Bird, 1993). This ratio roughly translates to 100 negative samples for every single positive one if sample lengths of 300 bp are used.

The first scheme is the nearly exactly the same as the conserved balanced approach described in section 5.2.3, with the alteration that now two negative samples are constructed for one positive sample. As 2, respectively 1, windows need to be conserved in the ARAprom and HOMprom set, 6 out of 18 remaining windows, respectively 7 out of 19, are randomly chosen to be conserved. This means for every dataset that, after the construction of the first negative sample, 12 windows were not used in the negative sample construction. From these 12 windows, 6, respectively 7, windows can again be randomly chosen to construct a new negative sample. In both negative samples the consensus sequences are then conserved, yet the other randomly chosen conserved sequences differ completely. The randomly chosen DNA bases are chosen in such a way that the species-specific base composition is retained.
The second approach is the addition of promoter sequences from related species as non-promoter samples. For *A. thaliana* the only other plant available in the EPDnew database is selected, namely *Zea mays* (corn), adding 17801 new samples to the negative ARAprom set. For *H. sapiens*, the most closely related mammal is chosen from EPDnew, resulting in the addition of 9575 samples from the *Macaca mulatta* (rhesus macaque) set. Note that this makes the trained network unusable for multispecies promoter prediction as the network will learn to focus on the promoters patterns unique to the species in the training set.
Third is the addition of sequences with an annotation other than 'promoter', such as introns and exons. Sequences can be extracted directly from the species reference genome, which is freely available for download on websites such as NCBI. Genome annotation can also be downloaded from the same website. Coding and non-coding sequences are then scanned for their length, and sequences smaller than 300 bp in length are discarded. For sequences up to 600 bp , a random starting point is chosen in order to obtain a 300 bp long sequences. For sequencing with a length between 600 and 900 bp , two sequences can be extracted. This is done in such a way that no overlap consists between the extracted sequences. This process is repeated for every multiplication of 300 bp , until no more genome sequences are left from which a negative sample can be obtained. Before the extracted sequences are added to the rest of the negative sample set, they are scanned to see if highly similar sequences are present to avoid biased prediction results. If two highly similar samples are divided over the training and test set, the algorithm will have no problem predicting the similar sample in the test set as it has already been seen during the training phase. This can lead to the algorithm having inflated prediction values for the test set, which do not reflect how well the algorithm would perform on an entirely unseen dataset. Samples are scanned with the Clustal Omega program (version 1.2.4; Sievers and Higgins (2018)) which outputs a full distance matrix, holding the similarity percentage between samples. If two samples share a similarity of 80% of higher, the sample which shares the highest overall similarity to all other samples is removed. This procedure is repeated until all samples have a similarity lower than 80% to each other.
Lastly, randomly generated DNA sequences are added until the desired class imbalance is reached. These random sequences are again constructed in such a way that the species-specific base composition is not altered. The samples generated with these four negative samples practices are then mixed together to yield one large negative sample dataset.