FACULTY OF SCIENCES

FACULTY OF BIOSCIENCE ENGINEERING

FACULTY OF ENGINEERING AND ARCHITECTURE

USING ARTIFICIAL NEURAL NETWORKS TO UNCOVER FEATURES IN PROMOTER SEQUENCES RESPONSIBLE FOR NON-ORTHOGONALITY IN E. COLI

Word count: 21.960

Lucas Davey
Student number: 01310378

Promoter: Prof. Dr. Willem Waegeman

Supervisor: Jim Clauwaert

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

Academic Year: 2019-2020

GHENT UNIVERSITY

Preface

After many long days and late nights, I am now on the home stretch. With the finish line in sight, I can reflect on what has been an exciting and rewarding journey. The results that I present here are the product of hard work and imagination, and while my computer monitor took the occasional beating, in the end it was well worth it. Finally, I can close this chapter of my life and I couldn't be happier.

First and foremost, I would like to thank my supervisor Jim Clauwaert for helping me with all my problems and questions, for the enthusiasm and creative thinking that was brought to discussions, and for being so contagiously optimistic. Without his guidance, my thesis would have been very different. Furthermore, I would like to thank my promoter Prof. Dr. Willem Waegeman for providing me with access to the facilities of the KERMIT research unit, and for being an outstanding teacher. And finally, Dr. Maarten Van Brempt for providing me with the necessary data.

At home, I want to thank my girlfriend Faezeh for being so caring and patient. My family, for being my biggest supporters and for providing me with all the tools I need to succeed. My father, especially, for proofreading my thesis and for encouraging me every step of the way, and last but not least, my cat Marmite for being absolutely adorable even though his advice was not very helpful.

Lucas Davey, Ghent, April 2020

List of Abbreviations

ANN	Artificial Neural Network
AUC	Area Under Curve
CNN	Convolutional Neural Network
FACS	Fluorescence Activated Cell Sorting
fcANN	fully-connected Artificial Neural Network
FN	False Negatives
FP	False Positives
ML	Machine Learning
NGS	Next Generation Sequencing
RNAP	RNA Polymerase
ROC	Receiver Operating Characteristic
PR	Precision Recall
TN	True Negatives
TP	True Positives
WT	Wild Type

Contents

1 Introduction **1**

1.1 Initiation of gene transcription in Bacteria 1

1.1.1 Sigma factors 1

1.1.2 Promoters 2

1.2 Synthetic biology 3

1.2.1 Engineering bacterial promoters with reliable strength 3

1.3 An introduction to Machine Learning 4

1.3.1 Supervised Learning 5

1.3.2 The bias-variance trade-off 5

1.3.3 Performance Evaluation 7

1.4 Artificial neural networks 10

1.4.1 The Perceptron 10

1.4.2 Layers of perceptrons 11

1.4.3 Gradient Descent 12

1.4.4 Regularisation 13

1.4.5 Convolutional Neural Networks 14

1.5 The classification of DNA 16

2 Aims **17**

2.1 Introduction 17

2.2 Aims 18

3 Results **19**

3.1 Data Generation 19

3.2 Choosing the performance metrics 21

3.3 Comparison of fcANN and CNN architectures 21

3.4 Investigating a subset of the data that is supported by more observations . . . 22

3.5 Improving performance with multi-task learning 26

3.5.1 Creating multi-label datasets 26

3.5.2 Training multi-task ANNs . . . 27
3.5.3 One multi-task model for all datasets . . . 28
3.6 Analysing the full multi-task model . . . 30
3.6.1 Visualising the output probabilities . . . 30
3.6.2 Extracting features . . . 30
3.6.3 Investigating the effect of Multi-task Learning . . . 34

4 Discussion **35**
4.1 Recap . . . 35
4.2 fcANNs and CNNs have similar performance . . . 36
4.3 High support samples are more reliable . . . 36
4.4 The advantages of multi-task learning . . . 38
4.5 The datasets are imbalanced and need more positive samples . . . 38
4.6 The extracted features might be motifs for interaction with σ^{70} . . . 39
4.7 Overall conclusions and future perspectives . . . 40

5 Materials and Methods **42**
5.1 Parameter optimisation and model evaluation . . . 42
5.2 Data Cleaning . . . 44
5.3 Data Generation . . . 45
5.4 Implementation . . . 46

6 References **47**

Abstract

English

With the explosion of biological data that has become available in recent years, there has been an increasing interest in developing new tools to mine this data for new features and patterns. Artificial neural networks (ANN) have the capacity to handle huge amounts of data and to uncover novel information that humans cannot identify. In this thesis, we used the results from the wet-lab screening of hundreds of thousands of *Escherichia coli* (*E. coli*) colonies to evaluate and optimise various ANN architectures for the classification of orthogonal and non-orthogonal promoters.

Orthogonal promoters are heterologous promoter sequences that only interact with an RNA Polymerase holoenzyme containing a specific, matched heterologous sigma factor. They are widely used in Synthetic Biology to allow individual regulation of genes to, for example, fine-tune the different steps of a new metabolic pathway.

In this thesis, we used ANNs to specifically identify motifs in the spacers of three promoters P_B, P_F and P_W of *Bacillus subtilis* that interact with sigma factors σ^B, σ^F and σ^W respectively. These promoters are orthogonal in *E. coli*, but through random mutagenesis of the spacer sequence, a library of orthogonal and non-orthogonal promoter sequences was created. We then trained and evaluated ANNs to classify the orthogonal and non-orthogonal promoter sequences and extracted learnt motifs that are available for further validation.

Specifically, our results suggest that a TATANT and TGN motif in the spacer is associated with non-orthogonality of the P_B and P_W promoters, and a CWWT motif with non-orthogonality of the P_F promoter. Furthermore, despite the challenges of working with inherently noisy and severely imbalanced datasets, we found that convolutional neural networks and fully-connected artificial neural networks have comparable overall performance with our datasets, and that an ANN that uses multi-task learning has many advantages.

Nederlands

Door de explosie van biologische data in recente jaren, is er een groeiende interesse in het ontwikkelen van nieuwe tools die patronen kunnen vinden in deze data. Artificial neural networks (ANN) kunnen uit grote hoeveelheden data specifieke informatie halen die men anders niet zou vinden. In deze thesis, gebruiken we de resultaten van de wet-lab screening van honderden duizenden *Escherichia coli* (*E. coli*) kolonies voor het evalueren en optimalizeren van verschillende ANN architecturen voor de classificatie van orthogonale en niet-orthogonale promotoren.

Orthogonale promotoren zijn heterologe promotorsequenties die enkel interageren met een RNA Polymerase holoenzyme dat een specifieke sigma factor bevat die past bij de heterologe promotor. Orthogonale promotoren worden veel gebruikt in de Synthetische Biologie omdat ze de individuele regulatie van genen toelaten, bijvoorbeeld wanneer men specifiek de verschillende stappen in een metabolische pathway wil fijnstellen.

In deze thesis gebruiken we ANNs om motieven te identificeren in de spacers van drie promotoren P_B, P_F en P_W afkomstig uit *Bacillus subtilis*. Deze promotoren zijn orthogonaal in *E. coli*, maar door random mutagenese van de spacer sequentie werd een verzameling van orthogonale en niet-orthogonale promotor sequenties bekomen. Vervolgens trainden en evalueerden we verschillende ANNs voor het classificeren van deze promotor sequenties en extraheerden we motieven uit de ANNs die beschikbaar zijn voor verdere wet-lab validatie.

Onze resultaten zijn, specifiek, dat een TATANT en TGN motief in de spacer van de P_B en P_W promotoren geassocieerd zijn met niet-orthogonaliteit. Eveneens vonden we dat een CWWT motief in de spacer van de P_F promotor geassocieerd is met niet-orthogonaliteit. Verder, ondanks de uitdagingen die komen kijken bij het werken met noisy en zeer ongebalanceerde data, stelden we vast dat convolutional neural networks en fully-connected artificial neural networks een vergelijkbare performantie hebben en dat een ANN die multi-task learning gebruikt veel voordelen heeft.

Introduction

In the first part of this introduction, the biological context of this thesis is outlined, which primarily concerns the initiation of gene transcription in bacteria, with a particular focus on the role of sigma factors and promoters. Next, a brief overview is given of methodologies used in synthetic biology to produce reliable promoter strengths in bacteria. Then, basic Machine Learning (ML) concepts are introduced together with the principles of artificial neural networks before finally ending with a brief overview on the ML algorithms that have been used to classify different DNA sequences.

1.1 Initiation of gene transcription in Bacteria

Transcription is the process by which RNA transcripts are produced from a DNA template. Central in this process is the RNA Polymerase (RNAP) holoenzyme, which is composed of a multi-subunit core enzyme ($\alpha_2\beta\beta^{'}\omega$) and a sigma factor. To initiate transcription, the core enzyme first binds a sigma factor, which enables it to recognise and interact with a specific promoter sequence (Figure 1.1).

1.1.1 Sigma factors

Sigma factors are proteins that are essential for the initiation of transcription in bacteria (Burgess et al. 1969). This is because they dock the RNAP holoenzyme near genes by binding critical promoter elements (Section 1.1.2). In addition, sigma factors facilitate the unwinding of double stranded DNA around the Transcription Start Site (TSS) – the position where transcription starts at the 5' end of a DNA sequence. In this way, a so-called "transcription bubble" of unwound DNA is formed that allows the DNA template to become accessible to the RNAP holoenzyme (Browning et al. 2004; Paget 2015).

Bacterial sigma factors regulate many genes, and based on the type of genes that are regulated, two types of sigma factors can be distinguished: primary and alternative sigma factors. All bacteria have one essential primary sigma factor, which is responsible for the bulk of transcription during normal growth. But many bacteria also contain multiple non-essential alternative sigma factors (Wösten 1998). *Escherichia coli* (*E. coli*), for instance, has 1 primary sigma factor σ^{70} and 6 alternative sigma factors (Keseler et al. 2016).

The number of alternative sigma factors varies between bacterial species, but it generally corresponds to the complexity of the environment. Bacteria living in a complex habitat, such as soil, tend to have more alternative sigma factors than bacteria that are obligate pathogens

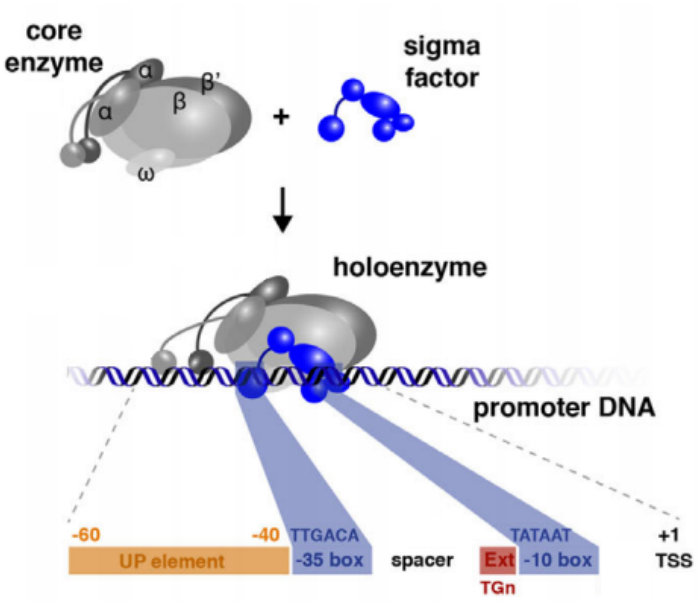

Figure 1.1: Schematic representation of the bacterial RNAP holoenzyme consisting of a core enzyme ($\alpha_2\beta\beta'\omega$) and a sigma factor that is responsible for promoter recognition. The core enzyme can dock at a specific promoter sequence after binding with a sigma factor. For this, the RNAP holoenzyme interacts with the -35 box and -10 box promoter elements with consensus sequence TTGACA and TATAAT respectively. In between these elements lies the spacer sequence. Ext represents the extended -10 region with consensus sequence TGN, and UP represents the promoter element upstream from the -35 box. The Transcription Start Site (TSS) marks the start of the region that is transcribed by the RNAP holoenzyme. Figure adapted from Bervoets and Charlier (2019).

(Roberts et al. 2017). This is probably related to the role of sigma factors in the overall adaptability of bacteria (Browning et al. 2004). In response to stress and environmental signals, sigma factors act as global switches, enabling the cell to quickly change its global gene expression program (Bervoets, Van Brempt, et al. 2018). For example, when *E. coli* is exposed to higher temperatures, σ^{32} mediates the synthesis of heat-shock proteins and DNA-repair enzymes that ultimately lead to the cell's survival (Yura et al. 1999; Nonaka et al. 2006). In *Bacillus subtilis* (*B. subtilis*), a more extreme response may be observed where a lack of nutrients triggers sporulation and the creation of an environmentally-resistant spore in a process regulated by a cascade of sigma factors (Haldenwang et al. 1981; Kroos et al. 1999).

1.1.2 Promoters

Bacterial promoters are regulatory regions of DNA situated immediately upstream of the TSS of a gene (Figure 1.1). Their function is to bind the RNAP holoenzyme and to regulate initiation of gene transcription. One of the defining characteristics of a promoter is its "strength". Promoter strength is a measure for the rate of transcription initiation of the associated gene. A strong promoter produces many RNA transcripts, while a weak promoter produces few. Bacterial promoter strength is mainly influenced by the DNA sequence (Li et al. 2014). Two promoter elements, the -35 and -10 box, are especially critical in this aspect (De Mey et al. 2007). These 6-nucleotide sequences are located upstream (towards the 5' end of the DNA sequence) from the TSS and interact with the sigma factor of the RNAP holoenzyme. Inefficient binding of the sigma factor to the -35 and -10 box has a significant effect on promoter strength for most bacteria

(Jensen et al. 1998). For that reason, the sequence of these promoter elements is conserved.

For *E. coli* promoters binding the primary sigma factor, the -35 box and the -10 box have the consensus sequence TTGACA and TATAAT respectively (Hawley et al. 1983). In general, promoters that have near-consensus sequences for the -35 and -10 box are stronger, while promoters that deviate significantly from this consensus are weaker (Browning et al. 2004).

The spacer sequence separates the -35 and -10 box, and eventhough it has no consensus sequence, its length and sequence still influence the promoter strength. However, this is to a lesser extent than the -35 and -10 box (Li et al. 2014; Jensen et al. 1998).

Some bacterial promoters have two additional elements that can also interact with the RNAP holoenzyme: the extended -10 region – a 3 bp motif with consensus sequence TGN (N being any nucleotide) located immediately upstream of the -10 element, and the UP element – a 20 bp region located upstream of the -35 box (Figure 1.1).

In conclusion, sigma factors and promoters play a critical role in the initation of gene transcription in bacteria. Both promoters and sigma factors determine which genes are transcribed, and promoters specifically also determine how high the rate of initiation of gene transcription is. This is largely determined by the -35 box and -10 box, but the spacer also has some influence.

1.2 Synthetic biology

Synthetic biology is an interdisciplinary field that combines elements of engineering and molecular biology to (re)design biological components that do not already exist in the natural world. Improvements in the speed and cost of DNA synthesis enable researchers to create novel biological components such as promoters, ribosome binding sites or gene-encoding DNA sequences. The individual components are then combined with increasing complexity to form circuits, pathways and systems that have real world applications including for example the creation of a renewable source of plastic (Yim et al. 2011), the production of biofuels (Atsumi et al. 2008), the degradation of pollutants (Petänen et al. 2001; Cases et al. 2005), and the production of medium-chain fatty acids (Liu et al. 2020).

1.2.1 Engineering bacterial promoters with reliable strength

Many synthetic biology applications require a specific level of gene expression. For example, in the overexpression of a particular gene, there is a careful balance between too high expression so that resources for growth are used, and too low expression so that the effect of overexpression is not visible. Since bacterial gene expression starts at the transcriptional level, precise gene expression is most often managed using synthetic or heterologous promoters. These promoters are then rationally designed to produce the desired level of gene expression. However, unlike endogenous promoters, synthetic promoters have not had millions of years of co-evolution with other cellular processes, and as a result, spurious interactions can occur between the promoter and the RNAP holoenzyme of the bacterial host (Cardinale et al. 2012). This crosstalk can lead to aberrant gene expression levels.

To design promoter sequences with reliable strength, three main strategies have been developed. The first strategy is to use robust promoters that are well-characterised and standardised (Mutalik et al. 2013) i.e. sequences that have been validated in many settings and conform to specific standards.

The second strategy is to use orthogonal promoters, which by definition are less dependent of the host's regulatory system (Lorenzo 2011). More specifically, in bacteria, orthogonal promoters are promoters that do not bind with the host RNAP holoenzyme. They either bind with an RNAP holoenzyme that contains a specific heterologous core enzyme such as the bacteriophage T7 core enzyme, or with an RNAP holoenzyme that contains a specific heterologous sigma factor (Rong et al. 1998; Bervoets, Van Brempt, et al. 2018). In the latter case, bacterial strains can then be engineered to express that heterologous sigma factor so that the gene controlled by the orthogonal promoter is expressed independently.

Both standardised and orthogonal promoters can be found in the registry of standard biological parts, a collection which counts over twenty thousand standardised and validated DNA sequences (Knight 2003). To date, the registry has many contributors consisting of both established labs and student teams participating in the annual International Genetically Engineered Machine competition.

The third strategy to engineering reliable promoter strength is more akin to a brute force method where a large part of the sequence space of a promoter is systematically searched. Rather than elucidating the relationship between the sequence and function, a collection or "library" of randomised promoter sequences is created and screened for promoters yielding the desired level of gene expression. As is often the case with brute force methods, this method suffers from combinatorial explosion: the longer the promoter sequence, the more possible sequences there are. Therefore, rather than randomising the entire promoter sequence, randomising particular regions can prove just as effective.

As described earlier, the -35 and -10 promoter elements are essential for promoter strength, which makes these sequences an attractive starting point for randomisation. However, promoter strength for some bacteria – most notably *E. coli* – is much less dependent on these conserved sequences than other prokaryotes (Jensen et al. 1998; De Mey et al. 2007). Moreover, since the -35 and -10 promoter elements bind the sigma factor, randomising these sequences can cause the promoter to lose specificity towards its native sigma factor (Bervoets, Van Brempt, et al. 2018). Therefore, the spacer sequence – being of lesser importance to sigma factor specificity – can be randomised and still produce a large variation in promoter strength (Jensen et al. 1998).

In conclusion, standardised and orthogonal promoters are feasible options to attain reliable promoter strength, provided they are available. Randomised promoter libraries can also be used, but the design and screening of promoter libraries still requires considerable effort.

1.3 An introduction to Machine Learning

Machine Learning (ML) is a discipline that involves the study of algorithms that have the ability to learn without following explicitly programmed instructions (Samuel 1959). Rather than being told what to learn, an ML algorithm learns by finding patterns and inferring relationships from examples. Once available, a trained ML algorithm can be used for future prediction using novel datasets.

The applications of ML algorithms are very diverse, ranging from simply detecting if an email is spam to diagnosing patients (Christina et al. 2010). Diseases such as Alzheimer's disease and breast cancer – to name just a few – have been predicted with state-of-the-art performance (Sarraf et al. 2016; Rakhlin et al. 2018). Through the use of medical records, it has even been

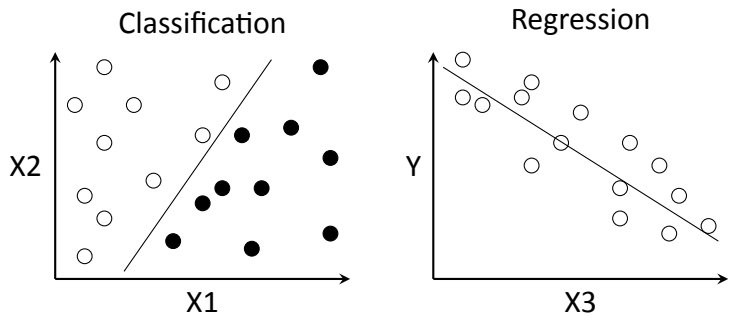

Figure 1.2: Difference between a classification and a regression task. For the classification task, a model (black line) is fitted that can seperate the labels (black, white) using samples containing the features X_1 and X_2. For the regression task, a model (black line) is fitted that approximates the values of the labels Y using X_3.

possible to reliably assess the suicide risk of patients (T. Tran et al. 2014). Other applications include the design of self-driving cars and improved quality control on assembly lines (Bojarski et al. 2016).

Despite many good uses, ML also has the potential to be misused. Facial recognition provides an easy-to-use biological pin-code, but it also poses a genuine threat to privacy as it further enables the unsolicited monitoring of people. Additionally, the rise of synthetic videos generated by ML models – so called "deepfakes" – add new meaning to the term fake news (Suwajanakorn et al. 2017). In the following Sections, the basic concepts of ML are introduced.

1.3.1 Supervised Learning

Supervised learning is the ML task of learning a function that maps an input to an output from data consisting of example input-output pairs (Russell et al. 2010). In supervised learning, each sample of a dataset is accompanied by a label and the goal is to predict the labels using the samples.

Within the group of supervised learning algorithms, further distinction is made based on the label (Figure 1.2). If the label is a continuous numerical variable, e.g. the price of a house, then the supervised learning task is regression. If it is a discrete categorical variable, e.g. the colour of a car, then the supervised learning task is classification. In this thesis, the labels are binary (samples are either a positive sample or a negative sample) so the supervised learning task is classification.

1.3.2 The bias-variance trade-off

In supervised learning, a model is created that estimates the true relationship between the samples and the labels. The estimations or predictions of the model are dependent of how accurate the model is: the closer the model is to the true relationship, the closer the predictions will be to the true labels. The difference between the model and the true relationship is therefore called the prediction error, and using the appropriate supervised learning algorithm can reduce it. The prediction error determines largely* how close the predictions will be to the

*The predictions are also dependent on a noise term that is irreducible.

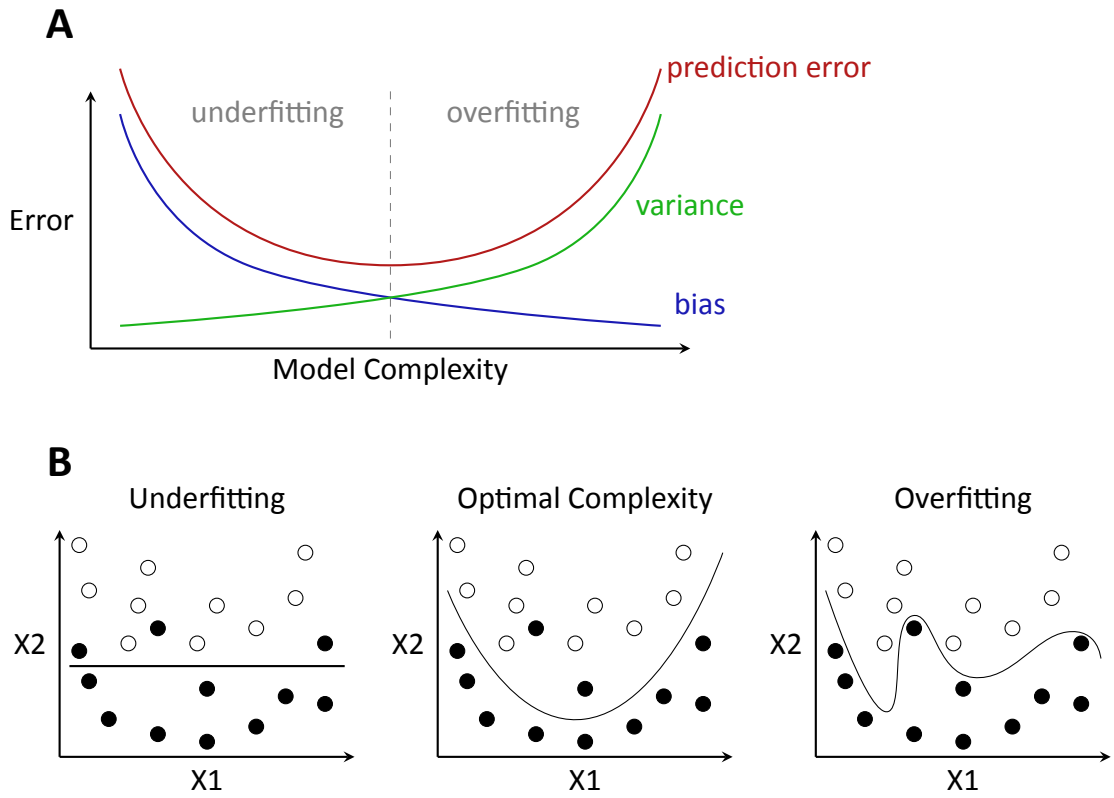

Figure 1.3: Overfitting and underfitting as a function of model complexity. (A) An increase in model complexity causes an increase in variance and a decrease in bias. The dashed line represents the complexity of the model with the lowest prediction error. Deviation from this optimum causes underfitting or overfitting. **(B)** Theoretical dataset consisting of two variables (X1, X2) and two labels (black, white) is fitted with a theoretical model with increasing complexity from left to right. The left model is too simple, and can never capture the non-linear relationship between the two labels, while the right model is too complex and captures the noise in the training set.

true labels and it can be decomposed into two sources of error: the bias and the variance.

But before we can elaborate on this, an important distinction has to be made. In any supervised learning task, the available dataset is always split into a training set (e.g. 80% of the available data) and a testing set (e.g. 20% of the available data). The training set is used to build a model, while the testing set is used to estimate its performance. The reason for this is further explained in Section 1.3.3.

The bias of a model refers to how well a model is able to predict the labels from the training set. A model with high bias pays very little attention to the training set and fails to capture meaningful patterns. Such a model is unable to generalise to other samples. This phenomenon is called underfitting.

The variance of a model refers to the amount by which a model changes if a different training set is used (James et al. 2013). If a model is sensitive to small changes in the training set, then it has a high variance. A model with high variance is prone to capturing patterns and intricacies – in essence noise – that is unique to that particular training set. This phenomenon is called overfitting.

High bias is caused by a model that is too simple, while high variance is caused by a model that is too complex (Figure 1.3). Reducing one error, will therefore cause the other one to increase.

This tension is called the bias-variance trade-off. One of the main challenges in supervised learning is finding a model that has both low bias and low variance. Essential to this, is how the model is evaluated.

1.3.3 Performance Evaluation

Earlier, it was stated that the available data is split into a training set that is used to build a model, and a testing set that is used to estimate its performance. Intuitively, it might seem weird to make this split, considering that in doing so the model can not learn meaningful relations that are present only in the testing set. Nonetheless, this split is necessary as the performance on the testing set provides an unbiased estimate of the generalisation capability of the model towards unseen samples. If the model were to be evaluated on the same data that was used to build it, the performance would be overestimated. For example, imagine that the ML model is a student taking an exam: if questions on the exam (the testing set) are the same as the questions in the textbook (the training set), then the student could just remember every answer in the textbook and get a perfect score without truly learning anything.

Aside from a train-test split, often, an additional split of the available data is made to create a validation set. The validation set is used to monitor the performance of the model during training. Based on the performance on the validation set, certain parameters of the model can be adjusted.

One downside from randomly splitting the dataset is that it can lead to unstable performance on the testing set due to the absence or presence of certain samples in the training set. To combat this effect, cross-validation can be used (Kohavi et al. 1995). Cross-validation is a performance evaluation method that calculates a more reliable performance by averaging the results of multiple models trained and tested on different subsets of the data. In k-fold cross-validation, the available data is split into a number of groups (k), and in an iterative process, each group is used to test a model that was trained on the remaining groups until all groups have been used as testing set.

Additionally, if the dataset is imbalanced, stratified k-fold cross validation splits in a stratified manner so that each group has a similar ratio of labels. For example, if a dataset has 20% positive samples and 80% negative samples, then stratified k-fold cross validation will split the data into training and testing sets that each also have 20% positive samples and 80% negative samples.

When evaluating the performance of a model on the testing set, different metrics can be used. Depending on the dataset, some evaluation metrics are more appropriate than others. The confusion matrix is a table that enables the visualisation of the performance of a supervised learning algorithm on a classification task by showing the amount of correct and incorrect predictions for each class (Stehman 1997). In Figure 1.4, a confusion matrix is shown for a binary classification problem. True positives (TP) and true negatives (TN) are samples that were correctly classified, while the false positives (FP) and false negatives (FN) are samples that were incorrectly classified as the opposite class. Using the confusion matrix, different evaluation metrics can be defined.

Accuracy, perhaps the most common evaluation metric for classification, can be defined as the proportion of correct predictions among all predictions (Equation 1.1). However, accuracy makes the assumption that false negatives and false positives are equally important, while in many scenarios, it can be more important to minimise the number of false negatives than to

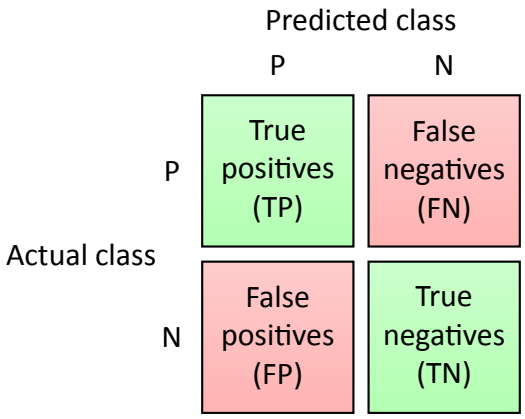

Figure 1.4: Confusion matrix for binary classification. P is the positive class, N the negative class.

minimise the number of false positives. For example, when diagnosing patients, not classifying a sick patient with a rare disease (FN), is much worse than classifying a healthy patient with a hypothetical rare disease (FP). After all, the patient with the rare disease might need treatment.

Moreover, for imbalanced datasets, i.e. datasets in which the number of positive samples is much lower than the number of negative samples, accuracy can be misleading. For example, if a disease is assumed to have a prevalence of 1%, a model that classifies every patient as healthy would achieve 99% accuracy even though it has no predictive power.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1.1}$$

The Receiver Operating Characteristic (ROC) curve is a plot of the True Positive Rate (TPR) (Equation 1.2) in function of the False Positive Rate (FPR) (Equation 1.3) at different threshold settings of a binary classification algorithm (Figure 1.5). The TPR defines the amount of correctly classified positive samples among all positive samples, while the FPR defines the amount of incorrectly classified positive samples among all negative samples.

$$TPR = \frac{TP}{TP + FN} \tag{1.2}$$

$$FPR = \frac{FP}{TN + FP} \tag{1.3}$$

Suppose that a binary classification model outputs a probability that a sample is positive. Intuitively, a value of 0.5 seems the most appropriate threshold for the probability that decides whether or not a sample should be classified as a positive sample. Decreasing the threshold of 0.5 to 0.4 increases the chance that a sample is classified as positive. In turn, less samples will incorrectly be classified as negatives (FN), but more samples will be incorrectly classified as positives (FP). Choosing a different threshold for classification thus enables more importance to be assigned to either the false positives or false negatives. In a ROC plot, all the possible values for the TPR and FPR of a model are plotted by shifting the probability threshold. Therefore, in contrast to accuracy, the ROC plot gives an overall view of the predictive capability of a model regardless of which threshold for classification is chosen.

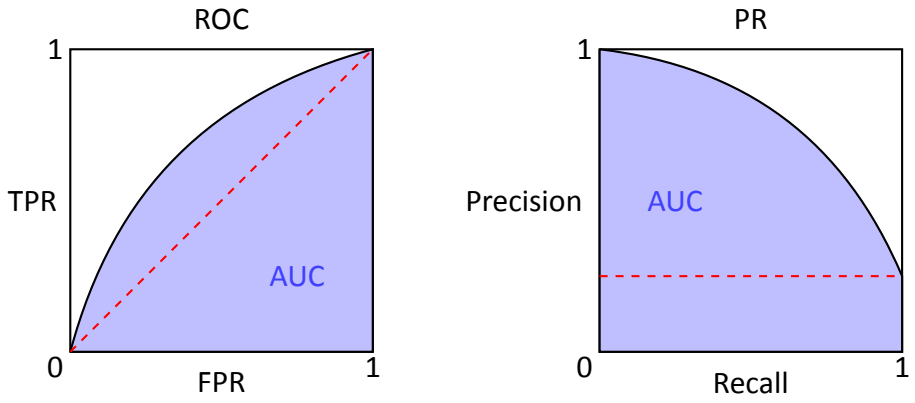

Figure 1.5: Theoretical Receiver Operating Characteristic (ROC) plot and Precision Recall (PR) plot. The red dashed lines represent the ROC and PR curve for a model that classifies samples randomly. The ratio of positive samples is 0.25, which serves as the baseline for the PR plot.

To compare the performance between two models using the ROC curve, the area under the curve (AUC) of the ROC curve is used. The ROC AUC is a metric which indicates how well the positive samples are separated from the negative samples across all thresholds. A model that perfectly classifies all samples has a ROC AUC of 1, while a model that randomly classifies all samples has a ROC AUC of 0.5.

In an imbalanced dataset with a minority of positive samples, the Precision Recall (PR) curve can be more informative than the ROC curve (Saito et al. 2015). The PR curve is a plot of the precision as a function of the recall. The recall is exactly the same as the TPR, and the precision is the amount of correctly classified positive samples among all predicted positive samples (Equation 1.4).

$$Precision = \frac{TP}{TP + FP} \tag{1.4}$$

Similar to the ROC curve, the precision and recall of a model are plotted for all possible thresholds and to compare the performance between models. Likewise, the AUC is also used and a perfect model has a PR AUC of 1. Since the TPR and the recall are the same metric, the difference between the ROC and the PR is the precision and the FPR. The precision is informative about the positive class, while the FPR is informative about the negative class. In a dataset with a minority of positive samples, the FPR stays misleadingly small for more thresholds since the amount of TN in the denominator is likely much higher than the FP. For this reason, the PR AUC is preferred over the ROC AUC in imbalanced datasets.

One downside of this metric though is that it should always be compared with its “baseline”, which is the ratio of positive samples. Therefore, it is difficult to compare PR AUC values between different datasets.

In conclusion, there are many metrics that can be used to define how good the predictions of a model are. In this thesis, both the PR AUC and the ROC AUC are used. The PR AUC is used because the data is imbalanced, and the ROC AUC is used to give an indication of the performance of a model regardless of the ratio of positive samples.

1.4 Artificial neural networks

Artificial neural networks (ANNs) are simultaneously one of the oldest and newest domains in ML. Research began mid-twentieth century when psychologist Frank Rosenblatt introduced the perceptron, a binary classification algorithm based on the biological neuron (Rosenblatt 1957).

1.4.1 The Perceptron

Summarised in Equation 1.5 and Figure 1.6, the perceptron algorithm starts by initialising a vector of weights $\vec{w} = (w_1 \ w_2 \ \dots \ w_p)^T$ and a bias* b with random values. Then, iteratively for each sample $\vec{x} = (x_1 \ x_2 \ \dots \ x_p)$ of the training set X (with $x_1 \ x_2 \ \dots \ x_p$ the features of $\vec{x}$), a weighted sum is calculated by multiplying every feature from $\vec{x}$ with the corresponding weight from $\vec{w}$ and summing the weighted inputs together. The weighted sum is then offset by the bias b and this output is passed through an activation function that binarises the output $\hat{y}$.

$$\hat{y} = \begin{cases} 1 \text{ if } \sum_{j=1}^{p} w_j x_j + b \geq 0 \\ 0 \text{ if } \sum_{j=1}^{p} w_j x_j + b < 0 \end{cases} \tag{1.5}$$

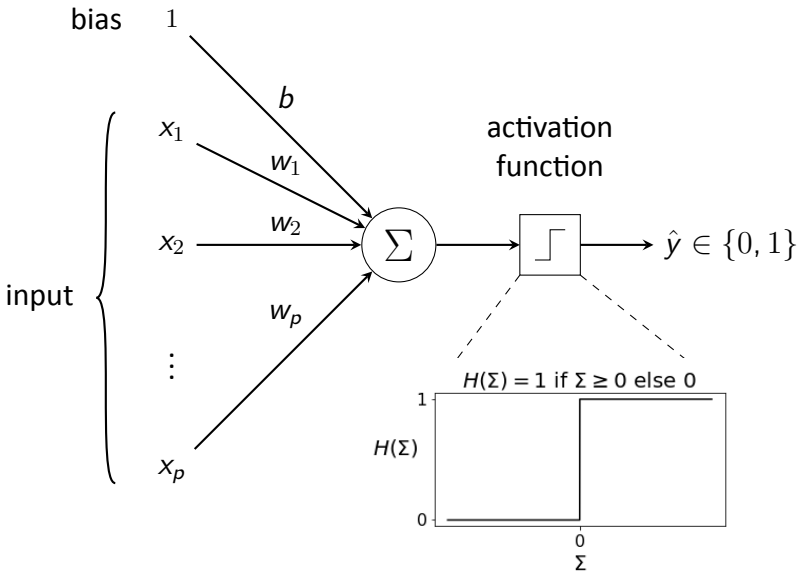

Figure 1.6: Diagram of the perceptron. Each sample $\vec{x} = (x_1 \ x_2 \ \dots \ x_p)$ is multiplied by the corresponding observation from weight vector $\vec{w} = (w_1 \ w_2 \ \dots \ w_p)^T$. Then, the weighted inputs are summed and offset by the bias b. Lastly, the weighted sum (Σ) is transformed to a binary output $\hat{y}$ using the step function H.

The weights of the perceptron are a measure of the relevance of each input variable to the classification of the training data – a weight with a large absolute value will have a bigger impact on the weighted sum (and the eventual label) than a weight with a smaller absolute value. The activation function, as the name suggests, is a function that determines when the perceptron

*Not to be confused with the bias from the bias-variance trade-off

activates – in this case, when the output is 1. If the input to this function is higher than the threshold 0, then the perceptron activates, otherwise it does not (Figure 1.6). Before being transformed by the activation function, the weighted sum is offset by the bias. The addition of the bias to the weighted sum means that a higher bias value makes it more likely for the perceptron to activate. The bias thus shifts the threshold for activation, and can be seen as a parameter for the overall sensitivity to the input.

The perceptron algorithm updates the weights and the bias for every sample from the training data based on whether or not the predicted label was true. After multiple passes of the entire training set and many updates, the weights and the bias have converged to values that yield an optimal classification of the labels of the training data. The perceptron thus learns by updating its own parameters.

Initially, this algorithm seemed very promising: the New York Times even reported the perceptron as "The embryo of an electronic computer that the navy expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence" (Olazaran 1996). These high expectations were however never realised as it was proven that the perceptron can only classify data that is linearly separable (Minsky et al. 2017). Interest was later renewed when multilayer perceptrons were introduced together with the backpropagation algorithm as an efficient learning algorithm (Werbos 1974). This event marks the birth of modern ANNs, which are essentially layers of perceptrons that feed into each other.

1.4.2 Layers of perceptrons

A biological neuron by itself does not provide the means for intelligence. Similarly, the perceptron by itself is too simple a model. However, when interconnected as neurons in a brain, a network of perceptrons is capable of complex classification.

To do this, the nodes of an ANN are organised in distinct layers: there is one input layer, one or more hidden layers and one output layer (Figure 1.7). These layers are also said to be fully connectedi, that is, the output of each node is part of the input of all nodes of the next layer.

For the sake of clarity, in Figure 1.7, an ANN architecture is shown with only two input nodes, three hidden nodes and two output nodes. However, in practice, it is not uncommon to see multiple hidden layers with hundreds of nodes.

The output of a single node – from now on called the activation value – is still computed in the same way as the perceptron, but for computational efficiency, the activation values are computed per layer instead of per node. Weight vectors and biases are stacked vertically and using matrix algebra, the activation values for every node in a layer are computed in one calculation. This can be seen in Figure 1.7, where the activation values $a^{(2)}$ for the hidden layer are calculated. First the dot product of the activation values of the previous layer x and the weight matrix of the current layer $W^{(2)}$ is calculated. This operation yields a vector of the weighted sum for every node, which is then offset with the bias vector $b^{(2)}$ and transformed using the activation function g.

The activation functions in an ANN differ from the one used in the perceptron. In the hidden layers, a rectified linear unit function (ReLU) is typically used that sets negative input values to zero while retaining positive input values. In the output layer, the softmax function is used that transforms real values into probabilities that sum to one.

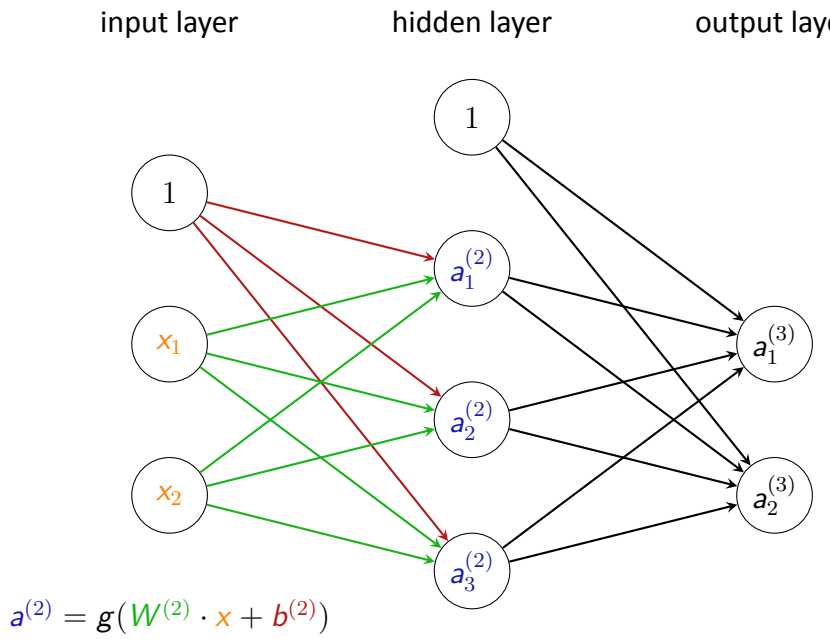

Figure 1.7: Diagram of an artificial neural network. Samples are propagated through the network from the input layer to the output layer. Connections coming from a 1 node are bias values since they are independent from any activation or input. Activation values are calculated per layer instead of per node, as shown by the equation for calculating the activation values $a^{(2)}$ of the hidden layer. Elements of this equation are color coded using the corresponding elements of the diagram.

1.4.3 Gradient Descent

Similar to the perceptron, an ANN learns to classify a sample by updating its weights and biases. However, unlike the perceptron with only a single weight vector and one bias value, an ANN has many more parameters that need to be optimised. To calculate the change that needs to occur for every weight and bias in the network, a loss function is defined that quantifies the distance between a prediction and its label. A higher loss equals a worse prediction.

For a network with two output nodes that outputs classification probabilities in the output layer, typically the log loss is used. As the predicted label moves further from the true label, the log loss increases exponentially.

To minimise the loss function and thus increase the performance of the model, the partial derivatives of the loss function are calculated with respect to all the weights and biases of the ANN. This is because the derivative of a single parameter of the loss function describes the slope of the loss function for that parameter. In turn, the slope is an indication of how a parameter needs to change to minimise the loss function. So in other words, the partial derivatives of the loss function with respect to all the weights and the biases are an indication of how these parameters need to change so as to decrease the loss most rapidly. And since the loss has to be minimised for all the training samples, the average of those partial derivatives is calculated.

Gradient descent is the optimisation algorithm that does the aforementioned. It is so named because the vector containing the partial derivatives for every parameter of an ANN is called the gradient vector. Using that gradient vector, the loss of an ANN is minimised. Intuitively, the process of updating the parameters of the ANN by calculating the gradient of the loss function is analogous to a ball rolling down a valley (Figure 1.8). In this example, the value of θ that yields

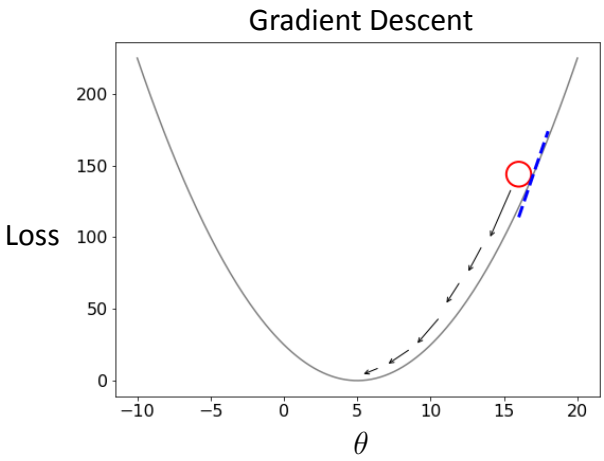

Figure 1.8: Gradient descent as a ball rolling down a valley. The red ball represents a value of θ with a loss as described by the function $\theta^2 - 10\theta + 25$. At the current position of the ball, the slope (dashed blue line) is calculated. The ball then rolls down the valley according to the slope. At that new position again the slope is calculated and after multiple epochs (arrows) the ball stops at $\theta = 5$, with a loss of 0.

the smallest loss is found by calculating the slope of the loss function for the current value of θ (the position of the ball). According to the slope, the value of θ is adjusted and this process continues until eventually a minimum loss is reached.

For a loss function with only one parameter, such as the one in Figure 1.8, calculating the derivative is straightforward. In reality, the loss function of a small ANN can have thousands of weights and biases per layer. Therefore, gradient descent uses the backpropagation algorithm to efficiently calculate the gradient vector of all the ANN parameters.

Normal gradient descent uses all samples to calculate the average gradient vector. However, since ANN applications often require many samples, calculating the average gradient vector over all samples can be computationally expensive. Therefore, modern alterations of gradient descent such as mini-batch stochastic gradient descent and Adaptive Moment Estimation calculate the average gradient vector with small batches of samples (Bottou et al. 2008; Kingma et al. 2014). In this way, accuracy of the average gradient vector is traded for computational efficiency and speed.

1.4.4 Regularisation

To capture complex relationships, ANNs often have thousands of parameters and, as mentioned in Section 1.3.2, overly complex models can result in overfitting. To prevent overfitting for ANNs, several specific regularisation methods have been developed. In this thesis, all of the regularisation methods discussed below were used to combat overfitting.

The first, called early stopping, entails stopping the training process in time. During training, the entire training set is propagated through the network multiple times in what is termed epochs. The more epochs an ANN is trained for, the better it learns to classify the samples, but conversely, the chance of overfitting is also increased. Therefore, the loss of an ANN is monitored during training with a separate validation set and once the loss on the validation set ceases to decrease, training is stopped.

The second method is called dropout. Nodes in an ANN learn based on the activations from

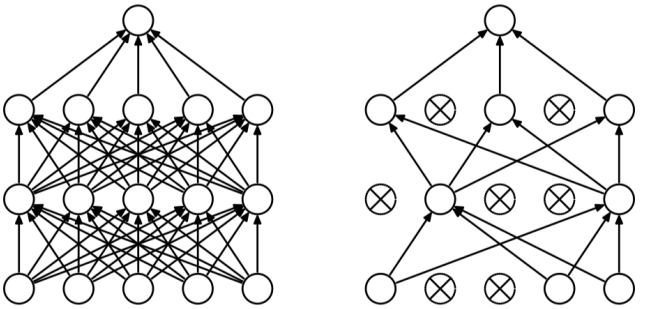

Figure 1.9: Dropout removes a random fraction of nodes during training. Figure adapted from Srivastava et al. (2014).

other nodes because samples are propagated from one layer to the next. This means that it is possible for some nodes to learn to fix mistakes from nodes in prior layers. This effect is called co-adaptation and it is computationally wasteful and often leads to overfitting. Dropout is a regularisation method that, during training, temporarily removes a random fraction of nodes along with their connections (Figure 1.9) (Srivastava et al. 2014). This has the effect of reducing co-adaptation between nodes by making the presence of other nodes unreliable.

Finally, batch normalisation, which was originally developed to speed up training, also has a regularisation effect (Ioffe et al. 2015). Batch normalisation is a transformation that is applied to the activations (and inputs) of an ANN. In batch normalisation, each activation of a layer $a^{(l)}$ is transformed to $a_t^{(l)}$ per mini-batch by subtracting its mini-batch mean μ_B and dividing by the mini-batch standard deviation $\sqrt{\sigma_B^2}$ (1.6).

$$a_t^{(l)} = \gamma\left(\frac{a^{(l)} - \mu_B}{\sqrt{\sigma_B^2}}\right) + \beta \tag{1.6}$$

In essence, this means that the input distribution of activations to a layer is scaled and shifted to have mean 0 and variance 1 per mini-batch in a process which is also called whitening of data.

Batch normalisation also provides two learnable parameters γ and β per layer that enable mini-batch stochastic gradient descent to transform the normalised data into a more optimal distribution.

1.4.5 Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a type of regularised ANN that is specialised in classifying images (LeCun et al. 1995). In a standard fully connected ANN, images are flattened to an array of pixels before being passed to the input layer. However, in a CNN, the input and the first hidden layers of a CNN are 3-dimensional: they have a height, a width and a depth. Images can thus be passed to the input layer as is without losing any spatial information. Apart from this attribute, the image-classifying properties of a CNN can be attributed to its use of convolutional layers.

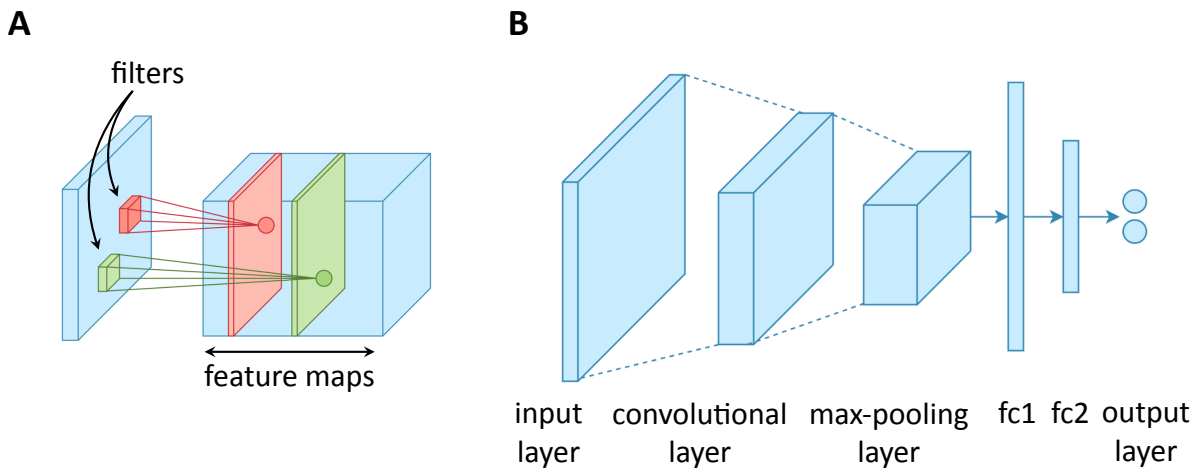

Figure 1.10: Diagram of a convolutional layer and the architecture of a Convolutional Neural Network (CNN). (A) The feature maps of a convolutional layer are constructed by convolving filters of weights over the previous layer. **(B)** A CNN consists of an input layer followed by convolutional and pooling layers and ends in fully-connected hidden layers (fc1, fc2) and one output layer. Figure adapted from https://towards-datascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

Convolutional layers are hidden layers that are connected to the previous layer by convolutions. A convolution is an operation where a 3-dimensional volume of weights, called a filter, slides over the nodes of the previous layer. For each volume of nodes that the filter slides across, one activation value is calculated in the convolutional layer. After a complete convolution, one 2-dimensional depth slice of nodes has been added to the convolutional layer. The convolution of multiple filters ultimately results in a 3-dimensional volume of nodes.

A depth slice of the convolutional layer is called a feature map (Figure 1.10 A). This is because the convolution of a filter corresponds to the detection of a feature. Filters are, in essence, feature detectors which by the convolution operation are used to scan an input image. Like the weights in a standard ANN, filters are not meaningful before a CNN has been trained. However, through gradient descent, filters are tuned so that they become meaningful feature detectors such as edge or texture detectors. The feature maps of a convolutional layer are thus a representation of those features that are important for classifying the input images. The more filters that are used, the more features that can be detected.

Convolutional layers also generally have fewer weights than a fully-connected layer. Through convolution, each node in a feature map of the convolutional layer is only connected to the previous layer via one filter of weights. Additionally, since the same filter is used to produce all the activations in a feature map, the weights for an entire feature map are identical. This drastically lowers the total number of weights, which in turn decreases the chance of a CNN to overfit.

After a convolutional layer, a pooling layer is sometimes used to aggregate the information contained in the feature maps. The pooling layer further downsizes the feature maps and reduces the number of weights. After a certain amount of convolutional and pooling layers, the resulting feature maps are flattened and fed into one or more fully-connected layers, which are connected to one output layer (Figure 1.10 B).

To make the distinction between a standard ANN and a CNN more clear, standard ANNs from now on will be called fully-connected ANNs (fcANN).

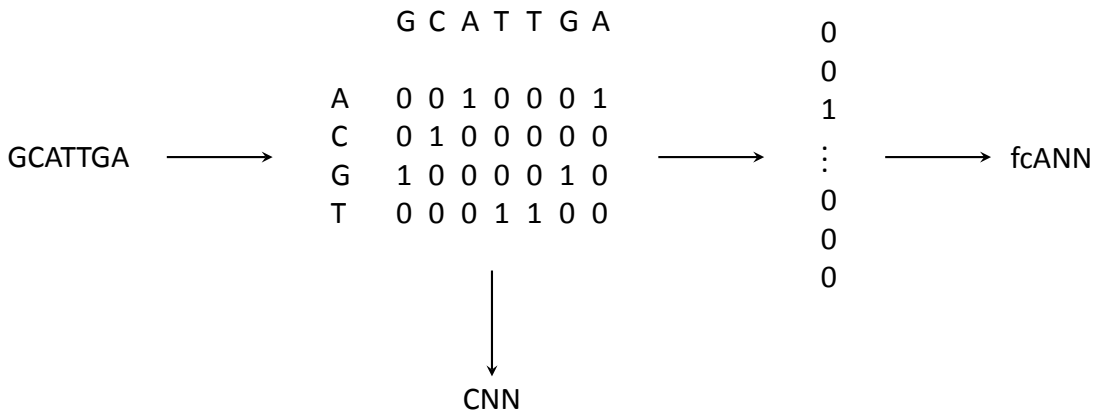

Figure 1.11: Example of one-hot encoding of a DNA sequence as input to a CNN or a fcANN. The one-hot encoded sequence matrix is flattened before being passed as input to the fcANN.

1.5 The classification of DNA

The classification of DNA is a problem in computational biology that has been approached using a variety of methods. Initial DNA classification algorithms were based on intrinsic features, specifically using the frequency of nucleotides per position in a DNA sequence to classify sequences (Pribnow 1975; Stormo et al. 1982).

Later, more sophisticated algorithms also used physicochemical and conformational features such as DNA stability, the DNA major groove depth/width and the DNA propeller twist with the aim of capturing more complex patterns (Bauer et al. 2010; Sonnenburg et al. 2006; T. Zhou et al. 2015; Ghandi et al. 2014).

In recent years, however, through technological advances in hardware and the use of Graphical Processing Units, fcANNs and CNNs have been applied successfully to sequence-based genomics problems (Cireşan et al. 2010; Yu et al. 2019). In particular, CNNs that were originally used in computer vision, are now applied in state-of-the-art methods to classify sequences or find motifs (J. Zhou et al. 2015; Alipanahi et al. 2015). This is because of their ability to extract features that are encapsulated in the "raw" DNA sequence such as the presence of motifs (Zeng et al. 2016). This is especially relevant in long DNA sequences where making a fully-connected ANN would not be feasible due to the number of weights. CNNs thus reduce the chance of overfitting by transforming the input DNA sequence to a collection of motifs.

To input a DNA sequence to an ANN, the sequence has to first be converted into numerical values. This is typically done with one-hot encoding (Figure 1.11). Here, each position in the DNA sequence is converted into a vector of length four that indicates which nucleotide is present at a particular position. For example, in Figure 1.11, the first position is represented by the vector (0, 0, 1, 0). The 1 at the third position represents a G nucleotide. The matrix that is obtained by stacking all these vectors can then be used as the input to a CNN or to a fcANN once it has been flattened.

Aims

2.1 Introduction

Genetically-engineered bacteria are used on an industrial scale for a wide range of purposes. They have for example been used to sense and degrade pollutants (Petänen et al. 2001; Cases et al. 2005), to create a renewable source of plastics (Yim et al. 2011), to produce biofuels (Atsumi et al. 2008), to invade cancerous cells (Zu et al. 2014) and to cost-effectively synthesise pharmaceuticals such as human insulin (Goeddel et al. 1979), ethionamin, a compound cytotoxic to *Mycobacterium tuberculosis* (Weber et al. 2008), and artemisinin, an anti-malarial drug (Martin et al. 2003; Ro et al. 2006).

Despite an overwhelming number of success stories, the engineering of novel functions in bacteria is hampered by an inability to ensure reliable (heterologous) gene expression. As discussed earlier in Section 1.2.1, synthetic promoters can have unwanted crosstalk with the bacterial RNAP holenzyme, which can lead to unexprected promoter strength and subsequently aberrant gene expression levels (Cardinale et al. 2012). Orthogonal promoters, instead, provide more reliable gene expression as they only interact with an RNAP holoenzyme containing a particular heterologous sigma factor.

To enable the use of different orthogonal promoters simultaneously and independently from each other, orthogonal promoters can be engineered to ensure that they do not interact with other heterologous sigma factors. Using such promoters, complex pathways can be uncoupled in separate modules that are then regulated independently of each other by different sigma factor-promoter pairs. This can, for example, give enhanced control over the synthesis of pathway intermediates, which may increase overall production yield (Bervoets and Charlier 2019).

2.2 Aims

In this thesis, we aim to provide insight in the optimal design of orthogonal promoters by uncovering motifs in the spacers of synthetic promoters – derived from *B. subtilis* – that underlie orthogonality or non-orthogonality in *E. coli*. For this, we build and evaluate ML models to carry out two classification tasks.

1. To distinguish non-orthogonal promoter sequences from orthogonal promoter sequences in *E. coli* (Figure 2.1 A).
2. To distinguish non-orthogonal promoter sequences from orthogonal promoter sequences that are independent as individual sigma factor-promoter pairs in *E. coli* . These orthogonal promoters only interact with one particular heterologous sigma factor (Figure 2.1 B).

Using feature visualisation methods, learnt features can then be extracted from these ML models.

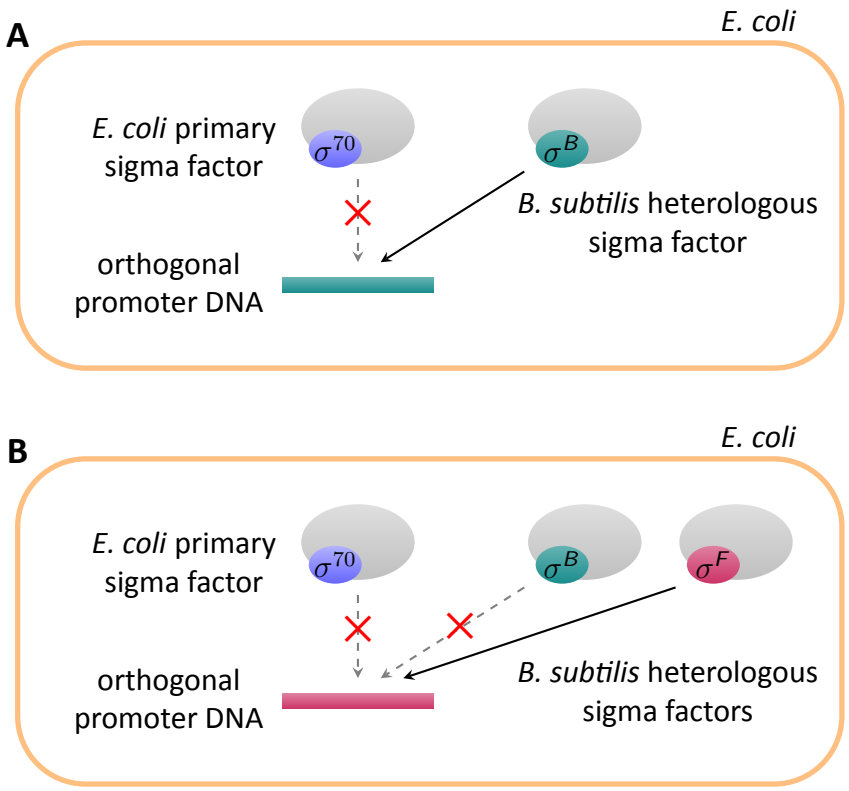

Figure 2.1: Schematic representation of the orthogonal promoters that we aim to classify. In light gray is the RNAP core enzyme, σ^B and σ^F are heterologous sigma factors from *B. subtilis*. **(A)** The orthogonal promoter does not interact with the host primary sigma factor σ^{70}, but it interacts with factor σ^B. **(B)** The orthogonal promoter does not interact with the host sigma factor nor does it interact with an σ^B. It only interacts with σ^F. In this example, σ^F and the orthogonal promoter are an independent sigma factor-promoter pair that can be present in the cell together with σ^{70} and σ^B without any crosstalk.

Results

In this Section we describe the construction and evaluation of artificial neural networks (ANN) to classify orthogonal promoters from non-orthogonal promoters in *E. coli*. This chapter starts with an overview of how the nine datasets for this thesis were created. Then, the performance of two types of ANNs: fully-connected ANNs (fcANN) and Convolution Neural Networks (CNN) are compared. Next, some space is dedicated to exploring a specific subset of each dataset that is supported by multiple observations, we investigate the use of multi-task learning to improve performance, and finally an analysis of the best model is made, from which we extract the features that the model has learnt.

3.1 Data Generation

To train a classification algorithm, a dataset with labeled samples is needed. In this case, that is a set of orthogonal and non-orthogonal promoters in *E. coli*. For this, three promoters P_B, P_F and P_W were selected from *B. subtilis*. Each of the aforementioned promoters interacts with its respective sigma factor σ^B, σ^F or σ^W, native to *B. subtilis*. However, P_B, P_F and P_W do not interact with σ^{70}, the primary sigma factor of *E. coli* – and so, by definition, these promoters are orthogonal in *E. coli*. The orthogonal nature of these *B. subtilis* promoters in *E. coli* can be attributed mainly to their consensus sequence, which is significantly different from the consensus sequence of σ^{70} (Bervoets, Van Brempt, et al. 2018).

For each of the orthogonal promoters, three synthetic promoter libraries were created with Polymerase Chain Reaction to create a total of nine libraries (Figure 3.1). Using random primers, the spacer sequence in between the -35 and -10 box of the promoter was randomised. By only randomising the spacer region and not the -35 and -10 box of the orthogonal promoters, the specificity of the synthetic promoters towards their native *B. subtilis* sigma factor was conserved whilst simultaneously allowing these synthetic promoters to lose their orthogonality (Bervoets, Van Brempt, et al. 2018). This means that the randomised synthetic promoters might interact with σ^{70} in *E. coli* even though the original promoters were orthogonal.

Next, the sequences were tested for orthogonality. For this, all nine synthetic promoter libraries were cloned in fluorescent reporter plasmids and transformed into *E. coli* K12 MG1655 strains expressing either the genes for σ^B, σ^F, σ^W or into the Wild Type (WT) strain. Red fluorescent protein expression was used as a reporter to characterise library promoter expression, while a constitutively expressed green fluorescent protein was used to correct for extrinsic factors. The *E. coli* cells containing the different promoter sequences were then sorted into two bins (positives and negatives) via Fluorescence Activated Cell Sorting (FACS) and sequenced using Next Generation Sequencing (NGS) on an Illumina MiSeq system.

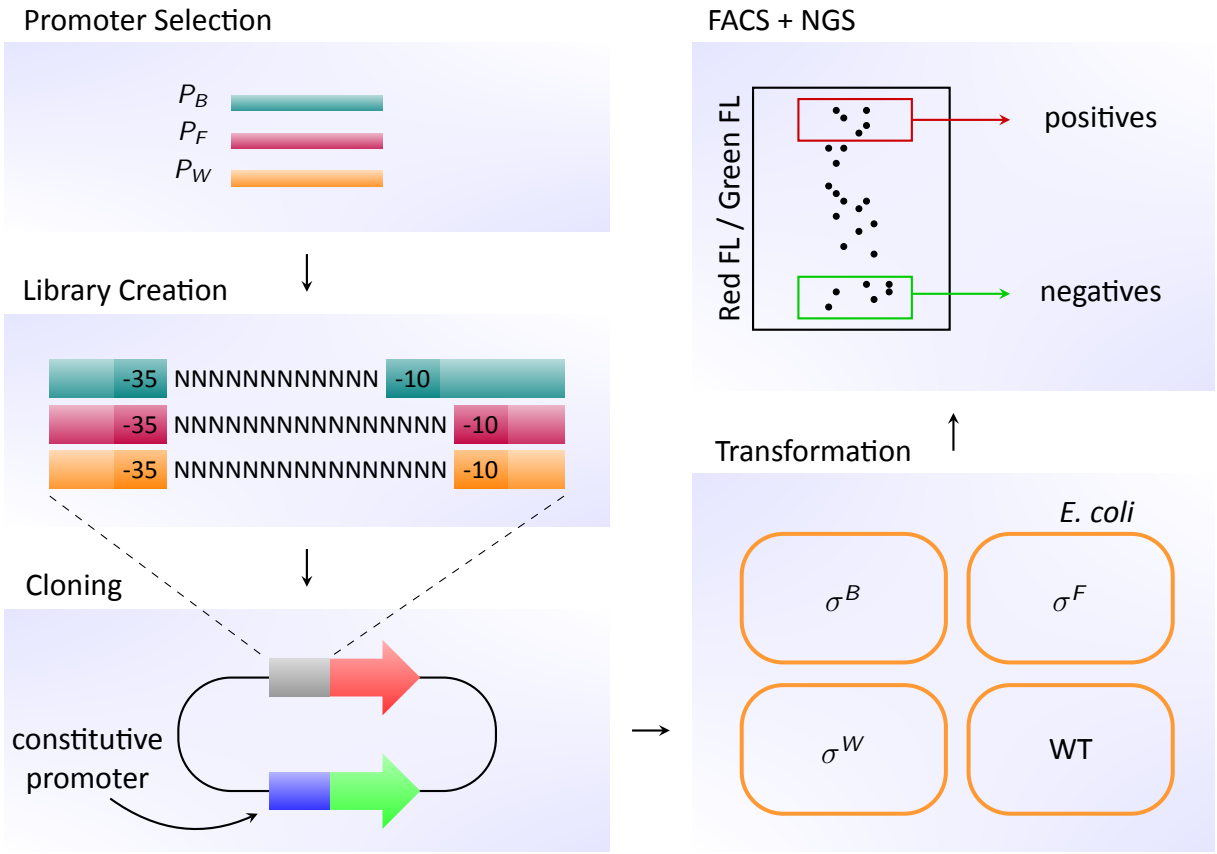

Figure 3.1: Schematic overview of the experimental setup. Promoter libraries were made from promoters P_B, P_F and P_W. These promoter libraries were cloned in fluorescent reporter plasmids and transformed in *E. coli* strains containing either the genes to express the *B. subtilis* sigma factors σ^B, σ^F, σ^W or in Wild Type (WT) *E. coli* expressing only the primary sigma factor σ^{70}. Lastly, the samples were sorted in a positive and a negative bin based on the fluorescence from the red and green fluorescent proteins in the reporter plasmid using FACS.

Ultimately, nine datasets were created. Three datasets containing synthetic promoter sequences that were tested for orthogonality in WT *E. coli*: B_{WT}, F_{WT}, W_{WT} and six datasets with synthetic promoter sequences that were tested for orthogonality in an *E. coli* strain that expresses both the genes encoding for the WT primary sigma factor σ^{70} as well as the genes encoding for σ^B, σ^F or σ^W: B_F, B_W, F_B, F_W, W_B, W_F. Each dataset is denoted by a capital letter that refers to the type of promoter library that was tested, and a subscript that refers to the *E. coli* strain into which it was transformed.

For example, the B_F dataset contains the synthetic promoter sequences that were initially derived from the *B. subtilis* promoter P_B and that were transformed in *E. coli* cells expressing the genes encoding for σ^F. A positive sample in this dataset is thus a promoter sequence that interacts with either σ^{70} and/or σ^F. A negative sample in this datasets is a promoter sequence that interacts with neither of the two sigma factors. Negative samples from this dataset are promoters that could be used together with σ^B as a separate sigma-factor promoter pair, even in the presence of σ^F. *E. coli* naturally also has other alternative sigma factors, however the genes encoding for these sigma factors were not expressed. This only occurs in response to stress such as higher temperatures or malnutrition.

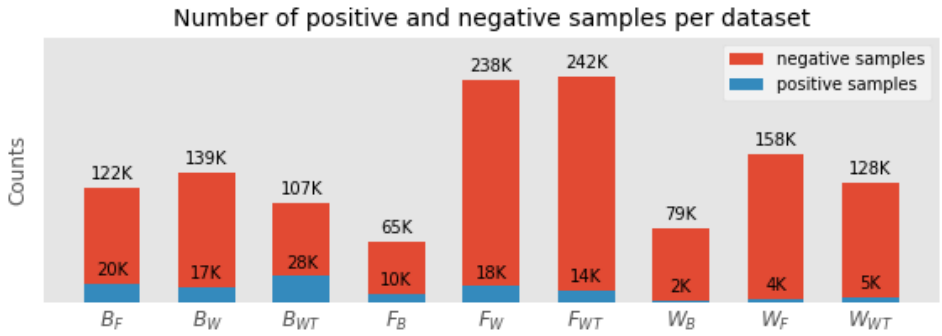

Figure 3.2: The datasets are imbalancead. Counts are the number of samples rounded to the nearest 1000 (K).

3.2 Choosing the performance metrics

Before training any ML algorithms, an appropriate performance metric has to be selected. Since the classificiation of orthogonal promoters from non-orthogonal promoters is a binary prediction problem, binary performance metrics such as the Receiver Operating Characteristic Area Under Curve (ROC AUC) and Precision Recall Area Under Curve (PR AUC) can be used. In contrast with accuracy, these performance metrics give an overall view of the predictive capability of a model that is independent of which probability threshold is chosen to distinguish positive from negative samples.

The ROC AUC can be too optimistic if there are more negative than positive samples, but it has the advantage that the baseline (i.e. the ROC AUC of a model that has no predictive capability) is always 0.5. Conversely, the PR AUC is more informative in case the dataset is imbalanced, but its baseline is the ratio of positive samples in the dataset – and thus the baseline varies between datasets. Therefore, in this thesis we opt to use both performance metrics: the PR AUC because all nine datasets are imbalanced (Figure 3.2) and the ROC AUC because it gives an overall view of the predictive capability of a model regardless of the ratio of positive samples in the dataset. A detailed comparison of the ROC AUC and PR AUC is provided in Section 1.3.3.

3.3 Comparison of fcANN and CNN architectures

ANNs have been used in many genomics classification problems, and in recent years, CNNs specifically have proven to be very effective learning algorithms (Yu et al. 2019; Oubounyt et al. 2019; Alipanahi et al. 2015; Zeng et al. 2016). In this thesis, both fcANNs and CNNs were implemented, and in this Section they are compared to each other.

For both the fcANNs and the CNNs, every DNA sequence was one-hot encoded. An example of this is provided in Figure 1.11 of the Introduction. The samples were also trimmed to only contain the spacer sequence since the promoter sequence surrounding the spacer was identical for all samples within a dataset (Section 5.2).

After the samples were prepared, the parameters of the fcANNs and CNNs were optimised. These parameters include the number of hidden layers, the size of each hidden layer, the number of convolutional filters and the presence of a pooling layer. A detailed overview of the pa-

Table 3.1: Description of the ANN and CNN architectures that were explored during the parameter tuning step. fc: fully-connected.

architecture	description
fcANN_1HiddenLayer	1 fc hidden layer
fcANN_2HiddenLayer	2 fc hidden layers
fcANN_3HiddenLayer	3 fc hidden layers
CNN_NoPooling_1HiddenLayer	1 convolutional layer, 1 fc hidden layer
CNN_1HiddenLayer	1 convolutional layer, 1 pooling layer, 1 fc hidden layer

rameter optimisation strategy is provided in Section 5.1. In short, each datasets was first split into a training set and a separate testing set. Then, a random combination of parameters was selected and the performance of a model with those parameters was estimated using stratified 5-fold cross validation. Depending on which parameters were randomly selected, the model was grouped into one of five different architectures to facilitate the practical implementation (Table 3.1). The models that had the combination of parameters with the best mean performance were then trained on the entire training set and tested on the separate testing set of each dataset. The results for this analysis are laid out later in Section 3.5.3.

During the parameter optimisation step, two notable observations were made concerning the different architectures.

Firstly, fcANNs and CNNs generally had the same mean performance within a dataset. Neither of the two types of ANNs performed specifically better, suggesting that the use of a convolutional layer does not have any benefits.

Secondly, some CNNs with the CNN_1HiddenLayer architecture (Table 3.1) showed exceptionally low mean performance compared to other architectures. Specifically those CNNs with a small filter had less than half the mean performance of the model with the best performing parameters. CNNs with a small filter size but a different architecture did not show this discrepancy in performance. Since the CNN_1HiddenLayer architecture is the only architecture that contains a pooling layer, this suggest that pooling the feature maps obtained from a small filter is detrimental to the performance.

3.4 Investigating a subset of the data that is supported by more observations

Each sample in each dataset consists of a DNA sequence and a label, either positive or negative. The label is, however, derived from two different features of a sample: observations clustered in the positive bin and observations clustered in the negative bin. As it turns out, some sequences in a promoter library occurred more than once in the data generation step, and they were sorted in the two bins more than once by FACS resulting in multiple observations per sample.

Table 3.2: Example of samples with different support values. The label of a sample corresponds to the bin with the most observations. Label 0, sample is a negative. Label 1, sample is a positive. Sample 1 is a low support sample because it has support $= 1$. Sample 2 is a high support sample because it has support ≥ 2. Sample 3 is not reliable because it has observations in both the negative and positive bin. This sample is removed from the dataset.

sample	spacer	label	positive bin	negative bin	support
1	AAGTC...	0	0	1	1
2	TAGCT...	1	2	0	2
3	AGTTA...	0	1	2	3

In the data cleaning step (Section 5.2), the samples that had observations in both the positive bin and negative bin were removed. After all, such samples are not trustworthy since they were sorted in two bins and thus have conflicting evidence for a label. Naturally, it can then also be said that samples with multiple observations in only one of the two bins are more reliable and trustworthy than samples with just one observation since the labels are supported by more evidence.

In this Section, we investigate whether or not samples with labels supported by more observations constitute a subset of samples that is more reliable. To do this, a new feature is defined called the "support", which is the total number of observations per sample. The majority of the samples for all datasets have a support of 1 and are called low support samples. A minority of samples has a support of 2 or higher and are called high support samples. Figure 3.2 shows an example of the different types of samples that were present in the datasets.

Initial tests showed that subsetting the datasets to only contain high support samples (samples supported by more observations) increased the performance on the testing sets. Moreover, the performance increased when the threshold for high support samples was also increased (i.e. high support samples have support ≥ 3). This suggests that by subsetting the data, a better model can be made.

However, one important aspect that was overlooked in these initial tests was that aside from the training sets, the testing sets were also subsetted to only contain high support samples. If the models are both tested on different testing sets, then it is not possible to conclude which model is better. For that reason, a more complicated setup was required.

The setup for this analysis involved models* that were trained and evaluated with different subsets of the data – as explained in Figure 3.3. Because a distinction is made in the train and validation set versus the testing set, it is possible to evaluate if training only on high support samples improves performance.

Figure 3.4 shows the results of this analysis for each of the nine datasets. The ROC AUC was used instead of the PR AUC since the PR AUC would have to be reported together with the ratio of positive samples for every subset. Since in this analysis the difference between the test performances is more important than the value of the performance itself, it suffices to use the ROC AUC.

The ROC AUC of models that were tested on the same subset of data are grouped by colour corresponding with the colours used in Figure 3.3. Generally, the ROC AUC of the models with the same colour was very similar, which means that the most important factor contributing to

*These models are the same models from earlier

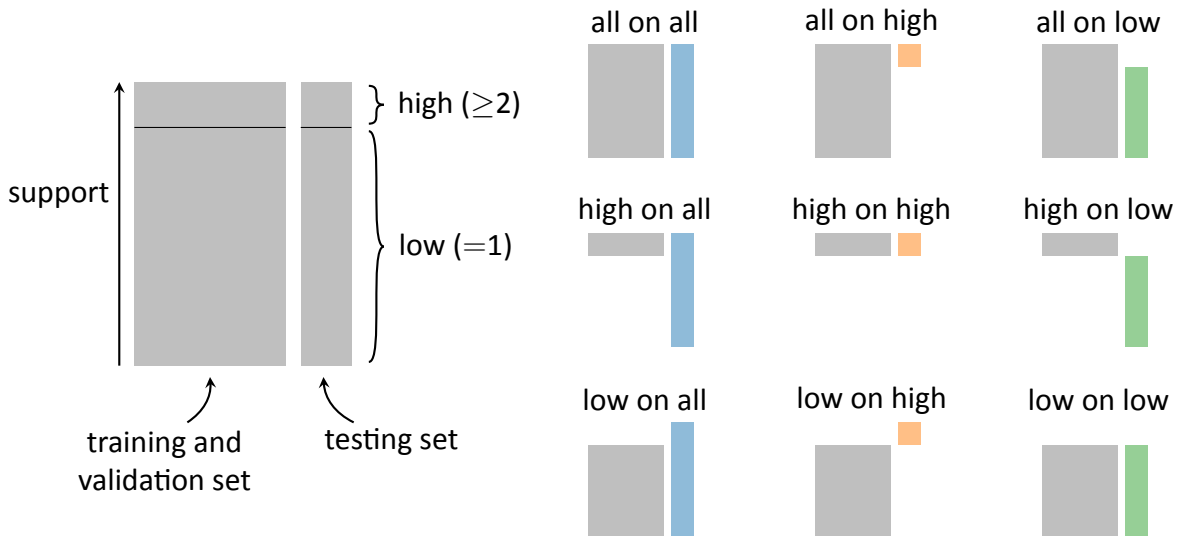

Figure 3.3: Schematic depiction of dataset subsets used to investigate if high support samples are more reliable. Left: a dataset can be presented as a box containing all the samples. Inside the box, samples are sorted according to their support value: a small subset of samples with a high support (two or higher) are grouped at the top of the box, while the majority of the samples with a low support (one) are grouped at the bottom. The box is split in two, which represents the split made when creating a train and validation set and a testing set. Though this is not depicted for the sake of clarity, the train and validation set is further split in a separate training set and a separate validation set. **Right:** Data subsets that were used to train and evaluate the models. Subsets contain either all support samples (all), only high support samples (high) or only low support samples (low). For example, the "high on low" model is a model trained and validated on the high support train and validation set, and tested on the low support testing set.

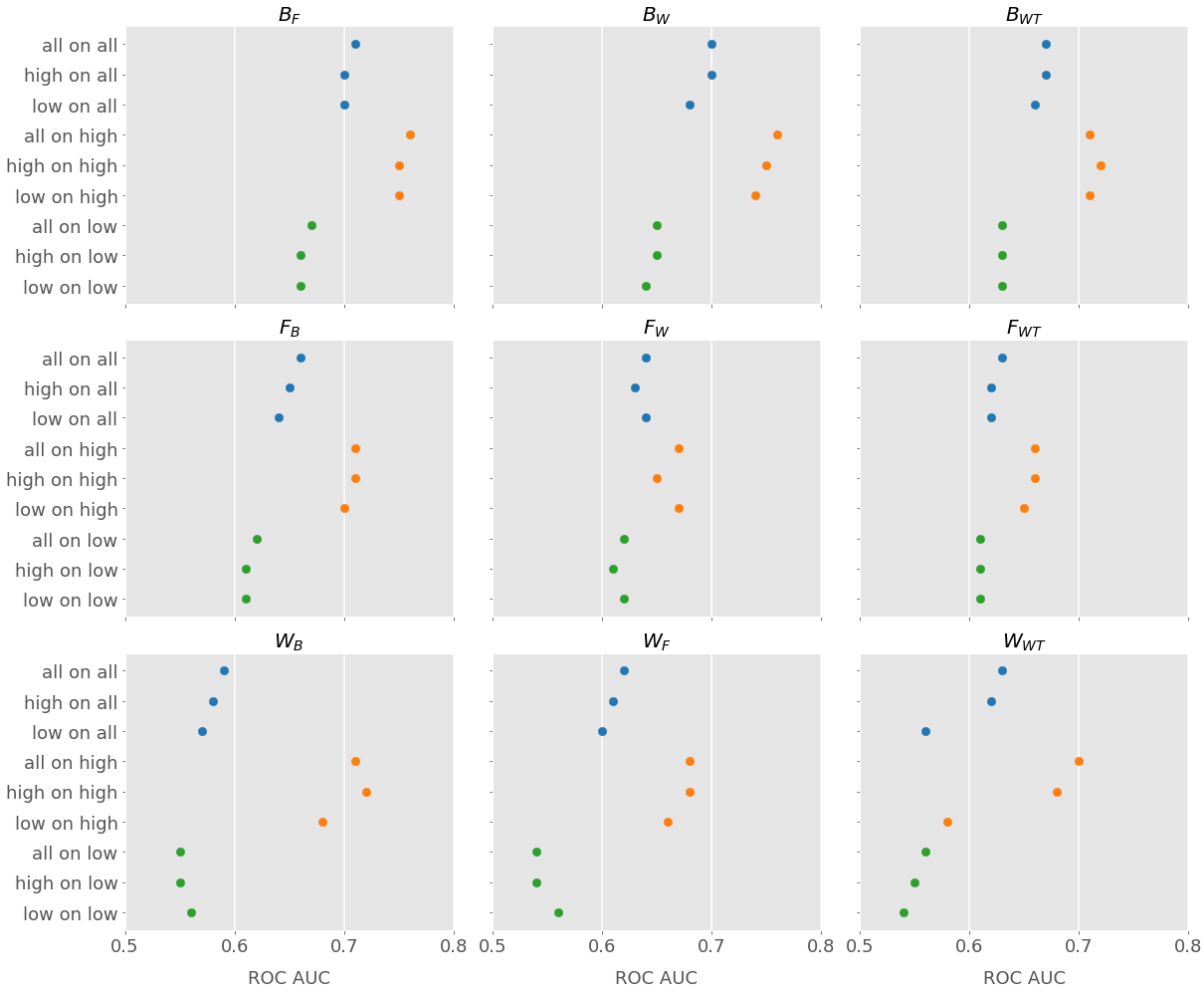

Figure 3.4: Performance in ROC AUC of models trained and evaluated with different subsets of data for each dataset. On the x-axis is the performance in ROC AUC, on the y-axis are the models trained and tested with different subsets of a dataset. Figure 3.3 explains the naming of the models on the y-axis. It can be seen that across all datasets, the ROC AUC (in orange) of the models tested on high support samples (all/high/low on high) is significantly higher than the performance of the other models. This shows that the high support samples are less noisy. Similarly, it can be seen that the ROC AUC (in green) of the models tested on low support samples (all/high/low on low) is significantly lower than the performance of the other models, indicating that low support samples are more noisy.

the differences in performance was the testing set and not the training and validation set. This observation reiterates our earlier notion that to compare models, it is essential that they are tested on the same data.

If high support samples are more reliable, it can be expected that the performance of models tested on high support samples (in orange) is higher than the performance of models tested on all the samples (in blue) or the performance of models tested on low support samples (in green) . Indeed, across all datasets, the performance of models tested on high support samples was significantly higher than the performance of models tested on all samples or low support samples.

Averaged across all datasets, the ROC AUC of a model trained on all support samples and tested on high support samples (all on high), was 0.06 higher than the same model tested on all samples (all on all) and 0.10 higher than a model tested on low support samples (all on low) (Figure 3.4). This suggests that the high support samples are less noisy, and that the low support samples are more noisy.

Knowing that the low support samples are more noisy might lead to the conclusion that this subset should be removed. However, the noisy data still contains useful information. This is because models that were trained on all support data (all on all/high/low) generally had the highest performance. In Figure 3.4, this can be seen by comparing the performances in any dataset of the same colour.

For example, in the F_B dataset, the model trained on all support samples and tested on all support samples (all on all), has higher performance than the model trained on high support samples and tested on all support samples (high on all). For this reason, the low support samples should be retained and not removed.

In conclusion, our results indicate that high support samples are more reliable than low support samples, since for all datasets the performance on this specific subset is higher than on the samples containing both low and high support samples. However, contrary to what our initial tests seemed to suggest, it is not necessary to remove the low support samples from the datasets.

3.5 Improving performance with multi-task learning

In total, nine individual models were created – one for each dataset. Each model required a parameter optimisation step and a training process. Depending on the size of the training set, this was both computationally intensive and time consuming.

As an alternative strategy, fcANNs were trained on multiple training sets at once allowing the samples of multiple testing sets to be classified in parallel. This is called multi-task learning and has the advantage that multiple tasks (i.e. datasets) are learnt by one model.

3.5.1 Creating multi-label datasets

To create a single model for application across distinct datasets, we first need to determine how the datasets can be combined. One obvious similarity is that the datasets that were derived from the same initial promoter sequence share the same nucleotide sequence surrounding the randomised spacer (Section 5.2).

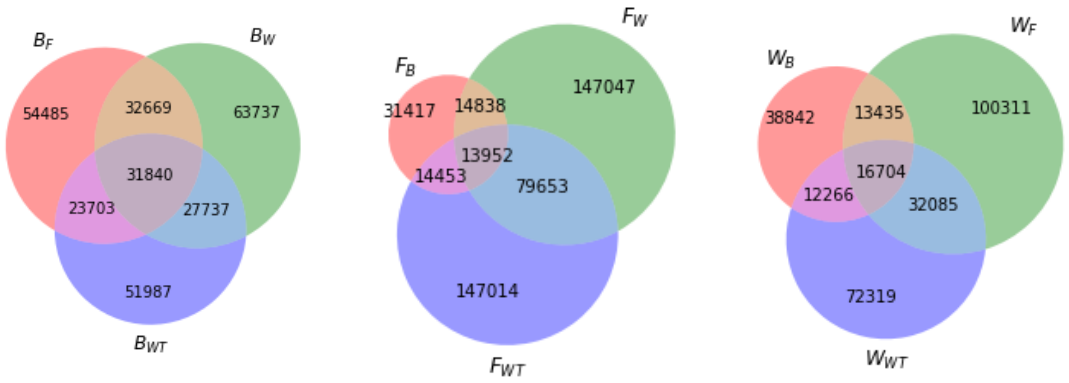

Figure 3.5: Venn diagrams showing the intersections of identical sequences between the individual datasets. Each subset of the Venn diagrams is annotated with the number of samples it contains.

Thus, three new datasets could be created from the individual nine datasets:

- B containing B_F, B_W and B_{WT}
- F containing F_B, F_W and F_{WT}
- W containing W_B, W_F and W_{WT}

In the new combined datasets, a promoter sequence can occur more than once across the individual datasets. The intersections of the individual datasets within the new datasets show that the individual datasets share a significant amount of sequences (Figure 3.5). As a result, samples from the new combined datasets can have one, two or three labels corresponding to the three original datasets. These labels are semantically different because they pertain to a specific sigma factor. For example, a negative sample in the B_F dataset is not the same as a negative sample in the B_W dataset. The former represents no interaction with σ^{70} nor σ^F, while the latter represents no interaction with σ^{70} nor σ^W. For that reason, the B, F and W datasets are multi-label datasets.

3.5.2 Training multi-task ANNs

To train a multi-task ANN on a multi-label dataset, three adjustments had to be made. First, the number of output nodes was increased. For a binary prediction problem, a standard fcANN most often contains two output nodes – one node that outputs the probability that the sample is a positive and one that outputs the probability that the sample is a negative. However, for a multi-task fcANN, the probabilities of the output nodes correspond only to the positive samples of the different tasks[†]. Therefore, the number of output nodes has to correspond to the number of individual datasets (three) within the multi-label datasets.

The second adjustment was to the activation function in the output layer. Instead of a softmax activation function, a sigmoid function was used. The softmax function calculates a probability for each output node so that the sum of the probabilities is one. Since a sample in a multi-label dataset can have multiple labels, the softmax function cannot be used anymore. The sigmoid

[†]The probability that a sample is a negative sample is then one minus the output probability.

function squishes the activation values into a range between 0 and 1 independently of the other labels.

Finally, the loss function was also altered. For a multi-task fcANN, gradient descent has to minimise the loss for each of the tasks instead of for just one. To accomplish this, the sum of the losses of the individual datasets was minimised. Furthermore, the loss also has to be able to disregard outputs and labels in the loss calculation that correspond with a non-existing label. This is because not all samples are shared between the three individual datasets of the multi-label datasets. Regardless of the number of labels that a sample actually has, a multi-task fcANN will always output a probability for each label. However, if a label is missing, it is not possible to calculate the loss for that label. After all, trying to determine how far off the output probability for a label is compared to a missing value does not make any sense. Therefore, the predicted probability was disregarded in the calculation of the total loss for samples with missing labels. This was accomplished specifically by applying a Boolean mask to the output and labels before the loss was calculated. The boolean mask is a function that removes probabilities and labels that do not meet a certain criterion, which was in this case: "is there a label?"

Similar to the analysis for the individual datasets, the ANN architecture that was selected to classify the multi-label datasets was obtained through parameter optimisation. This time, CNN architectures were not included though, since the analyses conducted with the individual datasets demonstrated that they did not show any significant improvement over fcANNs.

From the parameter optimisation, only fcANN_3HiddenLayer architectures (Table 3.1) were selected for all three multi-label datasets which shows that for the multi-label dataset, a more complex model is needed. The multi-task models with the parameter combination that yielded the best mean performance were then trained on the entire multi-label training sets and tested on the separate testing sets. Overall, the multi-task models showed slightly better performance on the separate testing sets than the individual models. The results for this analysis are described below, where the final model is also introduced.

3.5.3 One multi-task model for all datasets

The increased performance on the separate testing sets that was accomplished by combining datasets into multi-label datasets and training multi-task models proved that it was beneficial to have a single model carry out multiple, related tasks. With this in mind, a new and bigger multi-label dataset was created using all datasets.

A fcANN has a fixed number of input neurons, and as a consequence, the length of all input sequences has to be the same. The spacer sequence for B_F, B_W and B_{WT} were only 12 bp long, while for the other datasets the sequences were 16 bp long. Therefore, instead of using just the spacer sequence, the entire promoter sequence of length 50 bp[‡] was used.

Apart from a longer input layer, the length of the output layer was also changed to output nine probabilities instead of three – one for each individual dataset. The batch size was also increased to accommodate the larger dataset, and finally, the loss function was modified to minimise the sum of the nine individual losses.

From the results with the multi-task models, it was determined that for the classification of a multi-label dataset, a more complex model was required. Therefore, the parameter optimisa-

[‡]Promoters from the the B_F, B_W, B_{WT}, W_B, W_F and W_{WT} datasets were actually 51 bp long, but here the trailing nucleotide was removed.

Table 3.3: The performance on the separate testing sets is highest for the full multi-task model. For each dataset, the performance on a separate testing sets is given for three types of models in PR AUC and ROC AUC. The individual model is a model that was trained only on the training data of its corresponding dataset. The multi-task model is trained on the training sets of the datasets that share the same promoter sequence surrounding the randomised spacer. The full multi-task model is trained on the training sets of all datasets.

PR AUC									
	B_F	B_W	B_{WT}	F_B	F_W	F_{WT}	W_B	W_F	W_{WT}
individual models	0.46	0.39	0.47	0.32	0.20	0.18	0.04	0.05	0.07
multi-task models	0.47	0.41	0.49	0.34	0.22	0.19	0.05	0.05	0.08
full multi-task model	0.46	0.41	0.49	0.34	0.22	0.19	0.08	0.08	0.09
ROC AUC									
	B_F	B_W	B_{WT}	F_B	F_W	F_{WT}	W_B	W_F	W_{WT}
individual models	0.70	0.70	0.67	0.63	0.66	0.64	0.55	0.62	0.63
multi-task models	0.71	0.70	0.68	0.67	0.65	0.64	0.63	0.64	0.65
full multi-task model	0.71	0.71	0.68	0.67	0.65	0.65	0.63	0.65	0.66

Table 3.4: Ratio of positive samples in the separate testing sets.

	B_F	B_W	B_{WT}	F_B	F_W	F_{WT}	W_B	W_F	W_{WT}
ratio of positive samples	0.14	0.11	0.21	0.13	0.07	0.05	0.03	0.03	0.04

tion algorithm was allowed to select layers with more nodes and an architecture with four hidden layers. From this optimisation step, a model with four fully-connected hidden layers and many nodes per layer was selected.

In Table 3.3, the PR AUC and ROC AUC on the separate testing sets of every dataset is shown for all three types of models. Averaged across all datasets, the multi-task model has an increase in PR AUC of 0.008 compared to the individual models. For the full multi-task model, the increase in PR AUC is on average across all datasets 0.02 higher than the PR AUC of the individual models. Compared to the multi-task model, the full multi-task model does not have an increase in PR AUC for most testing sets. However, for the W_B, W_F and W_{WT} testing sets, there is an increase in PR AUC of 0.01 to 0.03. The same trends are seen for the ROC AUC on the separate testing sets, though less pronounced.

The ROC AUC reveals that the performance of the full multi-task model on the B_F is the highest, and the performance on the W_B testing sets is the lowest

In conclusion, considering the ease of use and the overall performance on all testing sets, we demonstrate that the full multi-task model gives the best results. In the next Section, this model is analysed in more detail.

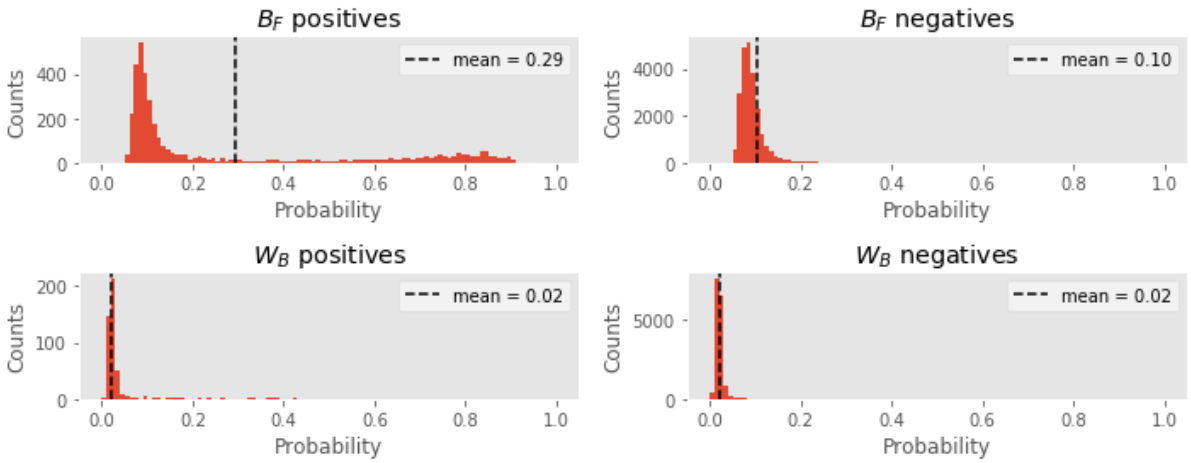

Figure 3.6: Histograms of the output probabilities generated by the full multi-task model for the B_F and W_B testing sets. Probability is the output probability and Counts the frequency of samples. The testing sets are split for the positive samples (positives) and negative samples (negatives) to highlight the difference in the distributions.

3.6 Analysing the full multi-task model

3.6.1 Visualising the output probabilities

The output probabilities of an ANN describe how confident the model is in its prediction. For the full multi-task model, the output of a specific node is the probability that a sample is a positive sample for the dataset corresponding to that node. A perfect multi-task model would thus output a probability of 1 for all positive samples, and a probability of 0 for all negative samples. Plotting the distributions of the output probabilities gives an understanding of how well the model performs that is more intuitive to understand than performance metrics such as the PR AUC or the ROC AUC.

In Figure 3.6, the distribution of probabilities generated by the full multi-task model is plotted for the B_F testing set and the W_B testing set. It can be seen that for both testing sets, the output probabilities of the positive and negative samples are low, which shows that the model classifies most samples as a negative and thus struggles to recognise positive samples. This is probably because the training sets for both models contained far fewer positive samples than negative samples. The B_F training set has 14% positive samples, and the W_B training set only 3%.

These distributions also show why the performance on the B_F testing set is so much higher than the performance on the W_B testing set. The output probabilities for the positive samples of the B_F testing set are on average 0.19 higher than the output probabilities for the negative samples, while for the W_B testing set, there is on average no difference between the output probabilities of the positive and negative samples.

3.6.2 Extracting features

To gain a deeper understanding of the trained models, it is useful to see which positions in the input promoter sequence have the most impact on classification. There are many methods to

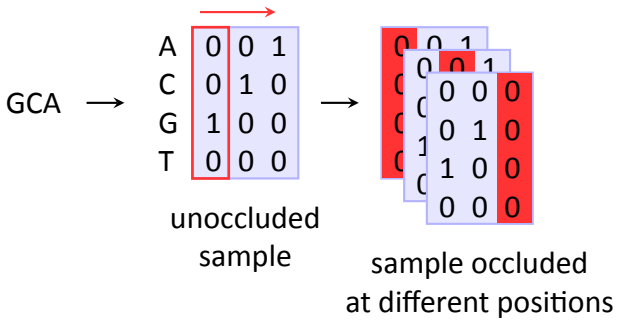

Figure 3.7: Schematic representation of the sliding window approach that was used to generate samples that are occluded at different positions. Samples for the full multi-task model are represented as 3 bp sequences for the sake of simplicity. First, the sequence of the sample is one-hot encoded. Then, a window (in red) slides over each position in the sequence matrix and sets all values of that position to zero, thus occluding that part of the sequence.

do this, but in this thesis, we used occlusion maps. An occlusion map is a representation of the relevance of each position in a sample, here, it shows specifically the performance of the full multi-task model on the separate testing sets after each position in the spacer is systematically occluded (hidden). Positions that show a drop in performance in the occlusion map compared to the performance of the unoccluded samples are likely to be important to the model for correct classification.

For DNA sequences, occluding a specific position corresponds with setting the column corresponding to that position to all zeros in the one-hot encoded sequence matrix. Using a sliding window approach, all positions in a sequence can then be iteratively occluded (Figure 3.7). In Figure 3.8, occlusion maps are shown for all testing sets using the full multi-task model.

As mentioned earlier, the input to the full multi-task model was the complete promoter sequence of 50 bp. In preliminary testing, occluding the positions surrounding the spacer showed to have no impact on the PR AUC when occluded. For that reason, these results were omitted from Figure 3.8 and only the spacer is shown.

From the scatter plots on the left, it can be seen that for all testing sets, the PR AUC decreases when positions are occluded compared to the PR AUC of the unoccluded samples (red dashed line). This was to be expected since information that can aid classification of the samples is removed when a position is occluded.

On a side note, for the W_F and W_{WT} testing sets it might seem as though the PR AUC of some positions in the scatter plot is higher than the PR AUC of the unoccluded samples. However, for these testing sets specifically, occluding the positions of the spacer does not have a big impact in general. The positions that seem to have a higher PR AUC (1, 2, 13, 14, 16 in W_{WT} and 3 in W_F) are actually less than 0.005 higher, which is not significant and probably due to rounding down of the PR AUC of the unoccluded samples.

The heatmaps on the right in Figure 3.8 are a more in-depth version of the scatter plots. They highlight which nucleotides are responsible for the PR AUC at any particular position. A single square in the heatmap represents the PR AUC of all test samples. However, the samples containing the nucleotide at the position corresponding with the square in the heatmap were occluded. Yellow squares represent relatively high PR AUC values, while purple squares represent relatively low PR AUC values.

For the B_F, B_W and B_{WT} testing sets, all positions in the spacer are relevant to the full multi-

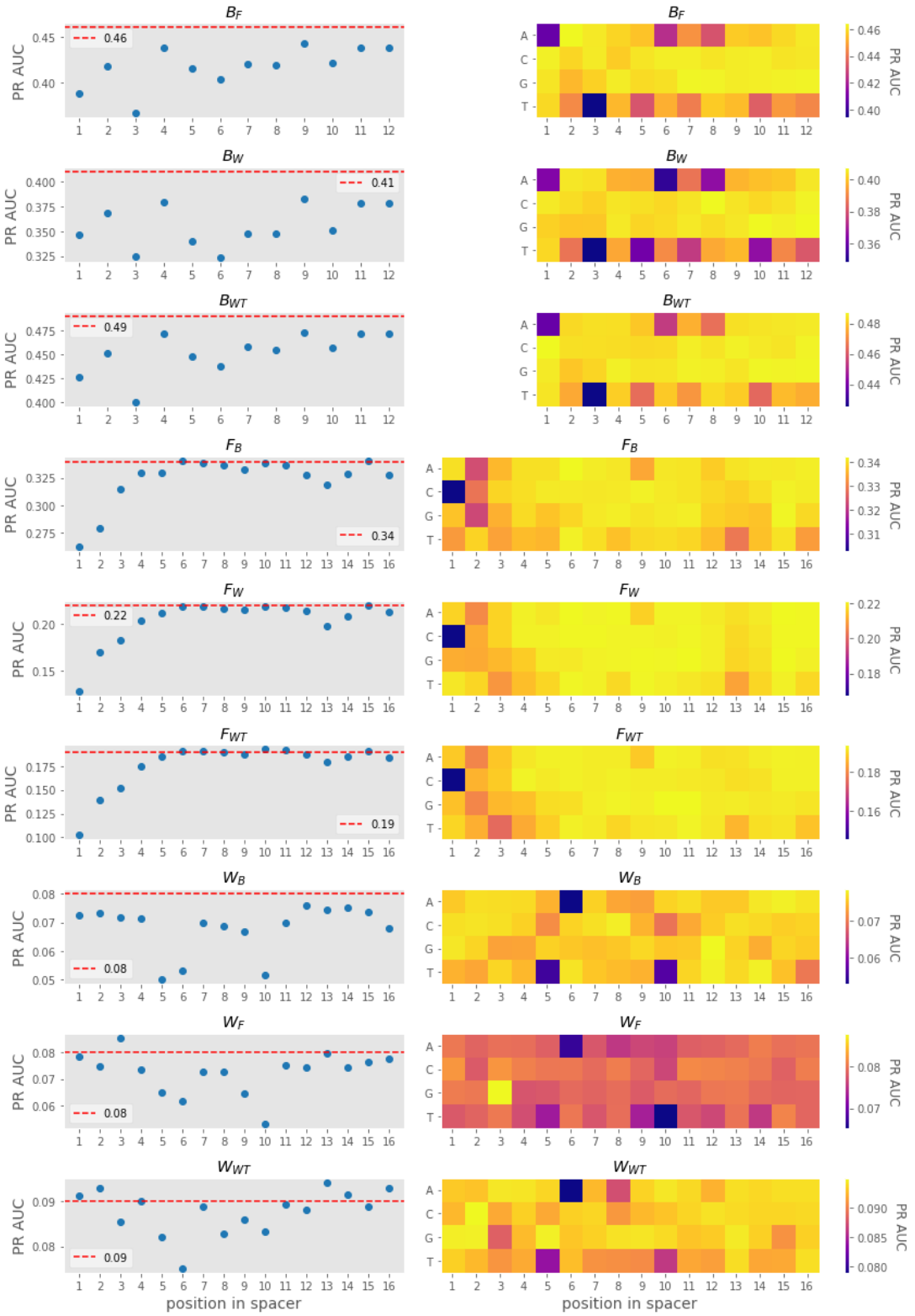

Figure 3.8: Occlusion maps of all testing sets generated using the full multi-task model. Left: Scatter plots depicting the change in PR AUC caused by occluding a position in the spacer for all samples in the testing set. The red dashed line is the PR AUC of the unoccluded test samples, and is used as a reference. **Right:** An in-depth version of the scatter plots on the left, depicting the change in PR AUC per nucleotide. The value of each square was calculated using all test samples, but the samples corresponding with the nucleotide at a certain position were occluded. Yellow squares have relatively high PR AUC and are not as important to classification, while dark purple squares have relatively low PR AUC and are important to classification.

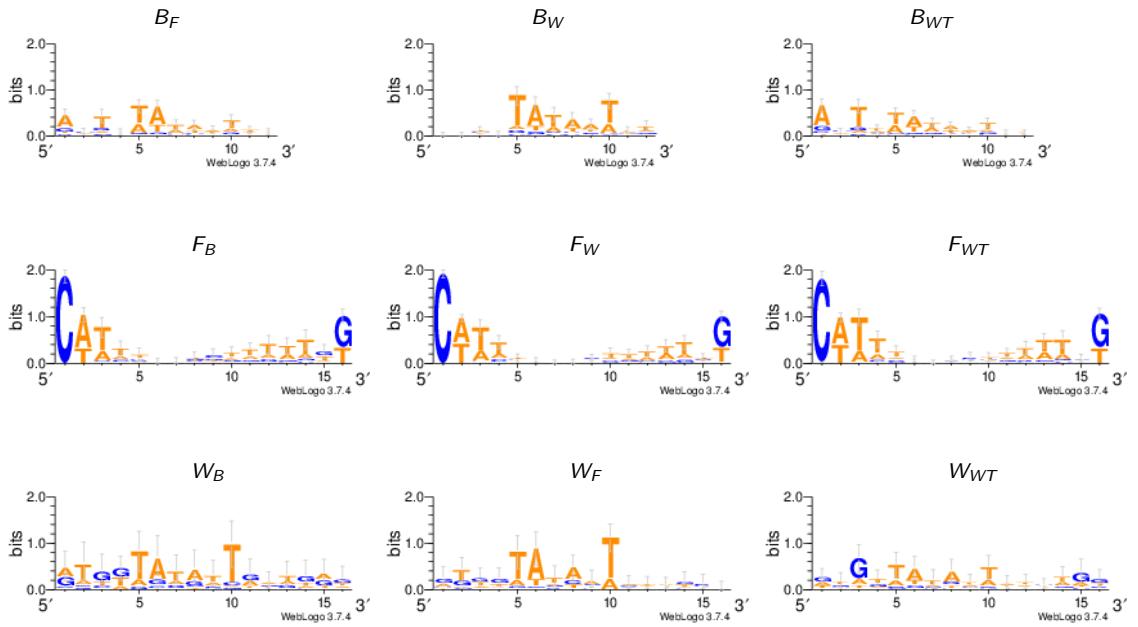

Figure 3.9: Sequence logos of the positive samples with the 5% highest output probabilities for the full multi-task model. The size of the letters represents the frequency of occurrence.

task model. Specifically, ATT at positions 1, 2 and 3 seems to be the most important, along with TATA at positions 5, 6, 7 and 8 and a T at position 10. In the entire spacer sequence, the C and G nucleotides have noticeably less importance.

For the F_B, F_W and F_{WT} testing sets, the positions at the beginning of the spacer are most relevant to the full multi-task model, especially CWWT (where W represents A or T) at positions 1 to 4. After that, a T at position 13 seems to also have a significant impact when occluded.

For the W_B, W_F and W_{WT} testing sets, TA at position 5 and 6 is the most important together with a T at position 10.

One disadvantage of this method is that the effect of the occlusion of a particular nucleotide on the PR AUC might not be visible if only few samples contain the nucleotide at that position. The interpretation of these results should thus be seen as an indication of importance and not as conclusive evidence.

From the output probabilities of the full multi-task model in Section 3.6.1, we concluded that the model classifies most samples as negatives. Therefore, a misclassified sample is likely to be a positive sample, and the features elucidated in Figure 3.8 are likely to pertain only to positive samples.

To investigate this, sequence logos were created from the positive samples in each testing set that had the 5% highest output probabilities (Figure 3.9). By using only those samples that the model classified most confidently, we ensured that the most important features were captured in the sequence logos. As can be seen, the majority of the features described in the occlusion maps were also recovered in the sequence logos.

3.6.3 Investigating the effect of Multi-task Learning

Many of the features extracted from the individual testing sets appear to be similar. In particular those features that were extracted from the datasets that were derived from the same initial promoter (i.e. datasets that were derived from the same initial promoter sequence). Even features extracted from the seemingly unrelated testing sets B_F, B_W, B_{WT}, W_B, W_F and W_{WT} share a TATANT motif at positions 5 to 10 (Figure 3.9). To quantify the similarity between the features, the performance of the different testing sets was measured using the output probability for other testing sets (Supplementary Table S1). For example, instead of using the output probability for B_F to predict B_F (as should be normally done), the output probabilities for the other testing sets were also used to predict B_F. If the performance of the full multi-task model on a particular testing set is high using the output probabilities for a different testing set, then this indicates that the model learnt similar features to classify the samples of both datasets.

It was found that the performance of the full multi-task model is identical on testing sets derived from the same initial promoter. For example, the PR AUC of the full multi-task model on the B_F testing set is 0.46. But by using the output probabilities for the B_W testing set, also a performance of 0.46 was found. This confirms that the classification of positive and negative samples is less related to the heterologous sigma factor that was expressed in the *E. coli* cells. Instead, it seems to be only related to the initial promoter sequence that the datasets were derived from. Furthermore, the performance of the full multi-task model on the B_F, B_W and B_{WT} testing sets was only slightly lower when the output probabilities for the W_B, W_F and W_{WT} testing sets were used, than if the correct output probabilities were used. All in all, these observations suggest that the full multi-task model learnt general features that are applicable to more than one individual dataset.

To confirm that multi-task learning is the reason for these shared features, sequence logos and occlusion maps were also created using the individual models (Supplementary Figures S1, S2). For the F_B, F_W and F_{WT} testing sets, the general features found in the full multi-task model are very apparent in the individual models, while for the B_F, B_W and B_{WT} testing sets, the general features are less defined, and in comparison, for the W_B, W_F and W_{WT} testing sets, the general features are not at all visible. Multi-task learning is thus somewhat responsible for these general features since they are more apparent in the full multi-task model than for the individual models. However, even in the individual models shared features are apparent, which suggests that the full multi-task model enhances these features, but they are genuinely important and not just a relic of multi-task learning.

Discussion

In this chapter, we discuss the results and end with an overall conclusion and future perspectives. But before this, a brief recap is provided of the goal of this thesis, and of the data generation procedure as this is important to interpret the results. For a more in-depth description, refer to Section 2.2 and 3.1.

4.1 Recap

In this thesis, we aimed to provide insight in the optimal design of orthogonal promoters by uncovering features in promoters that are responsible for orthogonality or non-orthogonality in *E. coli* . For this, we built and evaluated ML models that are able to distinguish non-orthogonal promoters from orthogonal promoters in *E. coli*, and we extracted learnt features.

To train such ML models, nine datasets titled B_F, B_W, B_{WT}, F_B, F_W, F_{WT}, W_B, W_F and W_{WT} were generated with each containing non-orthogonal and orthogonal promoter sequences. Each dataset was derived from one of three native *B. subtilis* promoters: P_B, P_F and P_W that are orthogonal in *E. coli* – meaning that they only interact with their respective *B. subtilis* sigma factors σ^B, σ^F or σ^W and not with σ^{70}, the primary sigma factor of *E. coli*.

For each *B. subtilis* promoter, three libraries of promoter sequences with randomised spacers were created. By randomising the spacer of the *B. subtilis* promoters, the promoters could lose their orthogonality. Each library was thus a collection of orthogonal and non-orthogonal promoters.

To test the orthogonality of the promoters, all libraries were cloned into a reporter plasmid and transformed into different *E. coli* K12 MG1655 strains. Three libraries were transformed into Wild Type (WT) *E. coli*, and six libraries were transformed into an *E. coli* strain expressing the *B. subtilis* genes encoding for σ^B, σ^F or σ^W.

The promoter sequences in the libraries were then classified as orthogonal or non-orthogonal by FACS and sequenced. Ultimately, this resulted in the nine aforementioned datasets. Each dataset is denoted by a capital letter that refers to the original *B. subtilis* promoter that the promoter library was derived from and a subscript that refers to the *E. coli* strain into which it was transformed.

For example, the B_F dataset contains the synthetic promoter sequences that were initially derived from the *B. subtilis* promoter P_B and that were transformed into *E. coli* cells expressing the genes for σ^F. A positive sample in this dataset is thus a promoter sequence that interacts with either σ^{70} and/or σ^F. A negative sample in this dataset is a promoter sequence that interacts with neither of the two sigma factors. Such a promoter sequence can be used as a separate

sigma factor-promoter pair with σ^B even in the presence of σ^F.

4.2 fcANNs and CNNs have similar performance

One important conclusion from this thesis is that despite the fact that state-of-the art classification algorithms primarily use Convolutional Neural Networks (CNN) to classify sequences (Yu et al. 2019; Oubounyt et al. 2019; Alipanahi et al. 2015; Zeng et al. 2016; Umarov et al. 2017), we demonstrate that for our datasets, both fully-connected artificial neural networks (fcANN) and CNNs have a comparable overall performance.

One possible reason that CNNs did not out-perform fcANNs is that the input DNA sequences to the CNNs were very short. In the literature, CNNs are mainly used with longer input promoter sequences. For example, Zeng et al. (2016) used a chromatin immunoprecipitation sequencing dataset with a sequence length of 101 bp, and Rizzo et al. (2015) use 16S gene fragments with a sequence length of 500 bp.

By comparison, in this thesis, very short spacer sequences with a length of only 12 to 16 bp were used. With longer input sequences, CNNs have the advantage over fcANNs, in the sense that they can extract motifs using convolutional layers and these motifs can then subsequently be used as input for fully-connected hidden layers instead of the entire input sequence. As a result, CNNs generally have fewer weights than fcANNS, which decreases the chance of overfitting and speeds up training. For short sequences, however, extracting motifs is less beneficial because the number of weights is already small.

In addition, it was demonstrated that a pooling layer following a convolutional layer with small filters is detrimental to the overall performance of a CNN. More specifically, the pooling layer is a global max-pooling layer that outputs the maximum value of the feature maps of the convolutional layer. Hereby, the global max-pooling layer further reduces the number of weights in a CNN. Since the feature maps contain information about the location of the extracted motifs, a global max-pooling layer specifically reduces the locational information about the motif in the input sequence to “present” or “not present” (Zeng et al. 2016).

The adverse effects of a pooling layer following a convolutional layer with small filters can potentially be understood as a consequence of the relative importance of locational information in a CNN with small filters, as compared to a CNN with large filters. For example, a small filter that extracts a 3 bp motif from the 12 bp long input sequence has 10 possible positions, while a 10 bp motif only has 3 possible positions. Therefore, a smaller motif without locational information is much more ambiguous than a longer motif.

In conclusion, irrespective of the algorithms and methods that have been used in the literature, it is more important to adapt the model to the characteristics of the datasets at hand. In this case, CNNs and fcANNs have similar performance because the benefits of a CNN are lost on short input sequences, and in a CNN with small filters, it is essential that the locational information is conserved because small motifs without locational information are more ambiguous.

4.3 High support samples are more reliable

Each sample in each dataset consists of a promoter sequence and an associated label, either positive or negative. The label of a sample was derived from the clustering during FACS. Here,

promoter sequences were sorted into either the positive bin or negative bin based on the fluorescence of the reporter plasmid. However, a minority of promoter sequences were present in a promoter library more than once. We suspect that this is due to a bias in the PCR step, which caused some sequences to be present more than others. These duplicate promoter sequences were sorted more than once, and thus their labels were supported by one or more observations in the negative or positive bin.

Some samples had observations in both the positive and negative bin. These samples were deemed unreliable and they were removed from the datasets during the data cleaning step, since the labels of such samples are essentially supported by conflicting observations. Conversely, samples with multiple observations in either the positive or negative bin can be seen as more reliable, since their labels are supported by more non-conflicting observations.

To test the hypothesis that samples with labels supported by multiple observations are more reliable, we defined the "support" of a sample as the total number of observations in the negative or positive bin. Samples with a support of two or more are called high support samples, while samples with a support of just one are called low support samples. We then compared the performance of models trained and tested on different subsets of data. More specifically, models were trained on low support, high support or all support samples (both low and high support) and tested on low support, high support or all support samples. From these analyses, three important conclusions were made.

Firstly, when determining which model is best, it is essential that the models are tested on the same data. From the analyses we can conclude that this is not possible when the models are tested on different testing sets, since the most important factor contributing to the differences in performance was not the data that was used to train the model, but rather the data that was used to test the model.

Secondly, high support samples are indeed more reliable. The performance on the high support samples was significantly higher than the performance on the all support samples or the low support samples. More specifically, the ROC AUC averaged across all datasets of a model that was trained on all support samples and tested on high support samples was 0.06 higher than the same model tested on all support samples, and 0.10 higher than the same model tested on low support samples. We hypothesise that this is because, overall, the datasets are quite noisy, but the high support samples are less noisy than the low support samples. This seems logical considering the fact that high support samples with conflicting evidence (i.e. observations in both the negative and positive bin) were removed during the data cleaning step. Therefore, high support samples are more reliable and less noisy because it was possible to detect and remove these noisy samples during the data cleaning step. Likewise, the low support samples are noisier because it was not possible to detect and remove the noisy samples.

Finally, removing the noisy low support samples from the dataset does not necessarily result in a better model, since the performance of the model across all datasets was highest when trained on all support samples. In other words, the noisy low support samples still contain some useful information.

It could be argued that training a model on only high support samples is a good alternative to training on all support samples when training speed is essential because the subset of samples with high support is smaller, yet such a model has a comparable performance.

In conclusion, we reiterated the importance of testing various models on the same testing set when determining which model is best, we confirmed our intuition that high support samples are more reliable, we proved that the datasets are noisy, and we provided an alternative subset

of the data that can be used for training when training speed is important.

4.4 The advantages of multi-task learning

The final model (the full multi-task model) is a multi-task model that was trained on all nine training sets . Averaged across all datasets, the PR AUC of this model was 0.02 higher than the average PR AUC of the individual models.

From a practical point of view, the full multi-task model is computationally more efficient than nine individual models since it has fewer weights in total. A multi-task model is also biased to learn features for one task (i.e. dataset) which may be of use in other tasks (Dahl et al. 2014; Angermueller et al. 2016). This is called representation bias and it makes the model less complex, which in turn prevents overfitting.

To determine if the full multi-task model learnt general features (i.e. features common to more than one dataset), we extracted and compared features of the different datasets. The use of two complementary visualisation methods (occlusion maps and sequence logos) revealed that there are many similarities between the features of the different datasets, and further that the extracted motifs are almost identical for those datasets that were derived from the same initial promoter sequence.

To confirm that representation bias is indeed the reason these general features were identified, features were also extracted from the individual models that were each trained on only one individual training set. Once more, the extracted features were found to be similar to each other, yet to a lesser extent than the features extracted from the full multi-task model. Therefore, we can conclude that general features are important for classification even for the individual models, and that the full multi-task model enhances the general features because of multi-task learning.

Overall, the full multi-task model is a better model because it is both computationally more efficient and because it is biased to learn general features.

4.5 The datasets are imbalanced and need more positive samples

The performance of the full multi-task model on certain testing sets is markedly lower than the performance on other testing sets. This is because for all testing sets, the model classified most positive samples as negative samples, which is a consequence of the imbalance of the datasets (i.e. all datasets have far more negative than positive samples). The datasets that are most imbalanced are then also those for which the full multi-task model was least able to correctly classifying the samples in the testing sets. It is known that classification algorithms used with imbalanced datasets will typically over-classify the majority group (in this case the negative samples) due to its increased prior probability (J. M. Johnson et al. 2019).

Common techniques to combat imbalanced datasets revolve around balancing the training sets. This includes the oversampling of positive samples, the undersampling of negative samples and synthetic minority oversampling (Chawla et al. 2002). However, none of these techniques contribute unique information to the model and oversampling can even lead to over-

fitting. Although not specifically reported in the results, balancing the training sets using the aforementioned techniques did not increase performance of the full multi-task model on the testing sets.

Therefore, we suggest that simply introducing more positive samples in the data generation step might prove beneficial for future studies. One way this could be done is by introducing a bias in the randomised spacers of the promoter libraries to generate more positive (i.e. non-orthogonal) samples. For instance, by deliberately avoiding spacer sequences in the data generation step that are similar to the spacer sequence of the orthogonal *B. subtilis* promoters P_B, P_F or P_W.

4.6 The extracted features might be motifs for interaction with σ^{70}

In this Section, we discuss the potential biological relevance of the features extracted from the full multi-task model. Note that the extracted features only apply to the positive samples, which means that the features are potential motifs in the promoter required for interaction with a sigma factor. This is again a relic from the imbalance of the datasets, which caused the models to only learn to distinguish positive samples.

Since the extracted features are general features that are shared between multiple datasets, we hypothesise that they are motifs that are specifically required for the interaction of promoters with σ^{70} in *E. coli*. For the B_{WT}, F_{WT} and W_{WT} datasets, this is a given considering that σ^{70} is the only sigma factor that is expressed (during normal growth) in WT *E. coli*. But for the B_F and B_W datasets, for example, we might expect to identify different features since these datasets were derived from *E. coli* promoter libraries where σ^F and σ^W were also expressed in addition to σ^{70}. Positive samples could thus be caused by the interaction of the promoter with σ^{70} or with a heterologous *B. subtilis* sigma factor. However, somewhat surprisingly, the features extracted from the B_F and B_W datasets are almost identical to those from the B_{WT} dataset. Likewise, all datasets that were derived from the same initial promoter sequence have almost identical features.

To understand whether the extracted features are motifs for interaction with σ^{70}, we can compare the extracted motifs with motifs known to be involved in interactions with σ^{70}. The TATANT motif at positions 5 to 10 in the spacer sequence was extracted for six out of nine datasets, namely B_F, B_W, B_{WT}, W_B, W_F and W_{WT}. This motif is very similar to the consensus sequence of the -10 box in *E. coli* (TATAAT), and together with a -35 box, this could explain the interaction with σ^{70}. However, the consensus sequence or any sequence close to the -35 box (TTGACA) was not extracted from the spacer nor was it detected in the promoter sequence surrounding the spacer for any of the datasets (Section 5.2). Nonetheless, it is known that *E. coli* promoters that do not contain a -35 box can still interact with σ^{70}, provided that they contain an extended -10 element (TGN) (Bervoets and Charlier 2019; Roberts et al. 2017; Browning et al. 2004). The extended -10 element is located immediately upstream of the -10 box and stabilises the transcription bubble that is formed when the RNAP holoenzyme binds the promoter to such an extent that the -35 box is not required (X. B. Johnson et al. 2006). The extended -10 element seems to be present in the W_B, W_F and W_{WT} datasets, however for the B_F, B_W and B_{WT} datasets, this promoter element was not found.

In the F_B, F_W and F_{WT} datasets, a leading CWWT motif was extracted along with a trailing K

(where W represent A or T, and K represent T or G). The trailing K is a relic of the design of the spacers for the F_B, F_W and F_{WT} datasets. This nucleotide in the spacer is conserved and essential for recognition of σ^F, however, the CWWT motif was not found to be meaningful for interaction with σ^{70}.

Overall, we can see that there are biologically meaningful explanations for some of the extracted motifs, which strengthens the idea that these features are relevant. On the one hand, the observation that the features are shared as well as the presence of a TATANT motif and an extended -10 motif for some datasets corroborate our hypothesis that the extracted features are motifs for interaction with σ^{70}. On the other hand, not all features can readily be explained and are possibly only present by chance. Therefore, it will be necessary to confirm the biological relevance of these motifs in follow-up experiments.

4.7 Overall conclusions and future perspectives

Overall, we can see that we succeeded in our goal of building and evaluating ML models that are able to distinguish non-orthogonal from orthogonal promoters in *E. coli*. From these models we were able to extract motifs from the spacers of the promoters that we suspect cause non-orthogonality due to the interaction with σ^{70}.

While the performance of the final model may not be particularly high, we have shown that this is mainly due to the inherent noisiness present in the datasets. Hereby, we reiterate the old maxim that flawed data produces flawed results. In the future, it will be necessary to determine why some samples can have conflicting observations from the wet-lab and to see if this process can be improved.

Furthermore, we highlight the importance of choosing the most suitable algorithm for an ML task and we demonstrate that multi-task learning has several advantages over normal supervised learning.

Finally, we provide a biological interpretation of the features extracted from the best model. We hypothesise that the model contains mainly learnt features that are important for interaction with σ^{70} of *E. coli*. Since many of the datasets are also associated with the expression of a specific heterologous sigma factor, the model was expected to have learnt distinct features for each dataset. Surprisingly, this was not the case.

We showed that the general features between the datasets are present in part due to the representation bias from multi-task learning, but also because they are genuinely important.

In the future, it may be easier to extract specific features for datasets that are associated with the expression of an additional heterologous sigma factor by preventing the primary sigma factor from interacting with the promoters. Perhaps this could be accomplished by short-term overexpression of Rsd, an anti-sigma factor for σ^{70} in *E. coli* that inhibits the formation of an RNAP holoenzyme containing σ^{70}. However, this may cause undesired side effects since σ^{70} is a housekeeping sigma factor that is essential for normal growth in *E. coli*.

Whether the extracted features are, from a biological perspective, genuinely the most important features present in the datasets, or only what the models “thinks” is important remains an open question, but the fact that there is some support suggests that it will be worthwhile following up on the motifs identified here.

Additionally, only those features that are important for interaction with σ^{70} were extracted be-

cause there were far more negative than positive samples. To extract features that specifically define orthogonal promoters, we suggest that more positive samples are created. This may be accomplished by biasing the random spacers in the data generation step to not include sequences that are similar to the spacer sequence of the original *B. subtilis* promoters, which are orthogonal.

In conclusion, orthogonal promoters are a widely used tool in Synthetic Biology to provide reliable gene expression. In this thesis, we have identified motifs in the spacers of synthetic promoters derived from *B. subtilis* promoters that might underly non-orthogonality in *E. coli*. Most importantly, a TATANT and TGN motif for promoters P_B and P_W, and the CWWT motif for promoter P_F. With this thesis, we hope to have to have shed some light on the design of orthogonal promoters, and to have demonstrated the usefulness of ANNs to classify DNA sequences.

Materials and Methods

5.1 Parameter optimisation and model evaluation

All the models in this thesis – the individual models, the multi-task models and the full multi-task model – were optimised using random search. Random search is a parameter optimisation strategy that selects the best combination of parameters from a predefined set by iteratively testing the performance of a model with a random combination of parameters. In Figure 5.1, this strategy is outlined for the individual models.

First, a separate testing set was created with 20% of the total data in the individual dataset. Then, a set of values was predefined for every parameter of the model and a random combination of those parameter values was selected. Following that, the performance of the combination of parameters was estimated using stratified 5-fold cross validation.

Cross validation provides a more reliable performance estimate of the model. Here, the performance estimate is the average of five different models that were each tested on different testing sets. In this way, the effect of the absence or presence of certain samples in the testing set is mitigated.

The combination of parameters that yielded the best mean PR AUC and the fewest number of weights was then used to train and validate a final model with all data (apart from the separate testing set). This model was then tested on the separate testing set to create the final performance values that are also provided in Table 3.3.

For the multi-task models and the full multi-task model, the parameter optimisation strategy was very similar. However, instead of starting with a complete multi-label dataset and then splitting it into a training set and a separate testing set, the training sets of the individual models were merged to form the multi-label training sets. Because the same testing sets were used, the performance of the multi-task and full multi-task models could then be compared with the performance of the individual models. It should be noted however that the percentage of samples in each subset as depicted in Figure 5.1 for the individual datasets was different for the multi-label datasets because the individual datasets share sequences.

Table 5.1 shows the range of possible parameter values per parameter and per model. The learning rate, the batch size and the type of Gradient Descent algorithm were not optimised using random search, since this would be too time consuming. Instead, preliminary testing showed that the ADAM Gradient Descent algorithm with a learning rate of 0.001 and a batch size of 100 were sufficient parameters for training. The batch size was only changed from 100 for the individual and multi-task models to 1000 for the full multi-task model to accommodate for the much larger dataset. For the individual models, at least 30 combinations of parameters

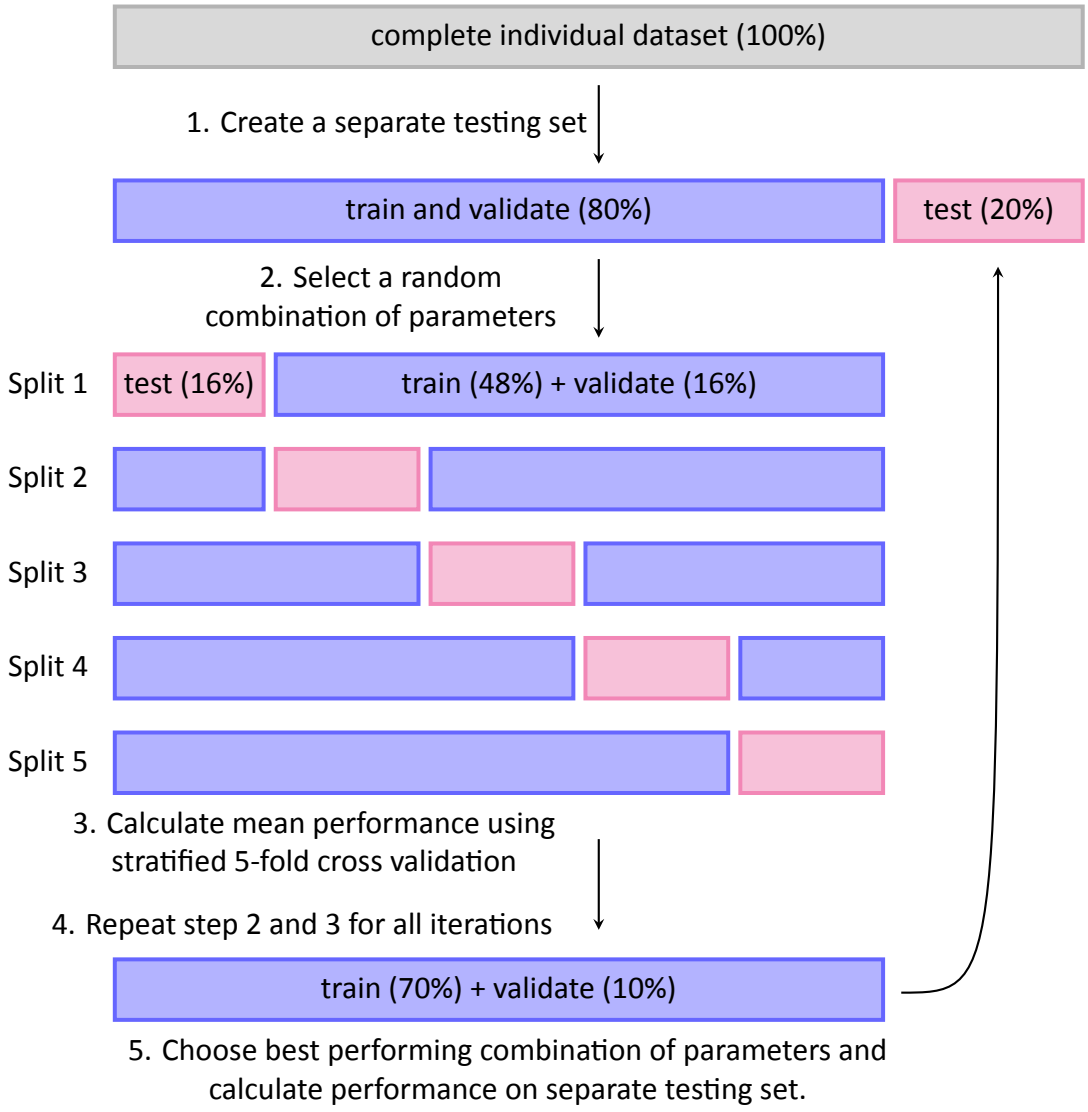

Figure 5.1: Schematic depiction of the parameter optimisation and model evaluation strategy for the individual models. All the splits were done in in a stratified manner so that each subset of the data (train, validate or test) contained a similar ratio of labels. Each subset of the data is annotated with the percentage of samples in that subset.

Table 5.1: Possible parameter values. For the individual models, if the ANN contained convolutional layers, maximum two fully-connected hidden layers were possible. Based on the dataset, the convolutional filter size was also restricted to the size of the spacer sequence (12 for B_F, B_W and B_{WT}, 16 for the other datasets).

individual models	
parameter	possible values
number of fully-connected hidden layers	$\{1,\ 2\}$ *or* $\{1,\ 2,\ 3\}$
fully-connected hidden layer size	$\{48,\ 52\ \ldots\ 384\}$
fully-connected hidden layer dropout chance	$\{0.0,\ 0.1\ \ldots\ 0.6\}$
number of convolutional filters	$\{48,\ 64\ \ldots\ 1536\}$
convolutional filter size	$\{2,\ 3\ \ldots\ 12\}$ *or* $\{2,\ 3\ \ldots\ 16\}$
pooling layer	$\{$*True*, *False*$\}$
multi-task models	
parameter	possible values
number of fully-connected hidden layers	$\{1,\ 2,\ 3\}$
fully-connected hidden layer size	$\{48,\ 52\ \ldots\ 384\}$
fully-connected hidden layer dropout chance	$\{0.0,\ 0.1\ \ldots\ 0.6\}$
full multi-task model	
parameter	possible values
number of fully-connected hidden layers	$\{3,\ 4\}$
fully-connected hidden layer size	$\{48,\ 52\ \ldots\ 768\}$
fully-connected hidden layer dropout chance	$\{0.0,\ 0.1\ \ldots\ 0.6\}$

were explored, while for the multi-task models and the full multi-task model, only 15 combinations were explored.

All models during parameter optimisation and model evaluation were trained for a maximum of 35 epochs, but with the constraint that if the loss on the validation set did not decrease for three consecutive epochs, training was stopped early. This is a regularisation method which is always used to prevent ANNs from overfitting (Section 1.4.4). After every epoch, the model was saved if the loss decreased. This enabled the model to be reverted to the state in the third to last epoch, where it had the smallest loss.

5.2 Data Cleaning

Before any of the models were constructed, the data was first cleaned. This entailed removing all erroneous sequences – which were present, in large, due to PCR errors. Table 5.2 shows the standard sequence of the promoters in each promoter library. All samples that did not match these sequences were removed.

The trailing K at position 16 in the spacer of promoter library P_F is a degenerate nucleotide (A or T) that is essential to recognition of σ^F. Although over 99% of samples in the F_B, F_W and F_{WT} individual datasets had this feature, some samples that did not have it were mistakenly not removed.

Table 5.2: Sequence of promoters in each promoter library. In bold and underlined is the -35 and -10 box of the promoters. N represent any nucleotide, and K represents T or G.

promoter library	DNA sequence (5' to 3')
P_B	TGTCGGAGAACGT**GTTTAT**NNNNNNNNNNNNN**GGGTAT**GTAACTTGTAGGGC
	-35 -10
P_F	TTCTGCGAT**GTTTA**NNNNNNNNNNNNNNNK**CTCATAAT**AGTAGAAACAGG
	-35 -10
P_W	TATAAAAAAAT**TGAAAC**NNNNNNNNNNNNNNNNN**CGTA**TACATACAGAGGGC
	-35 -10

Erroneous sequences were also a product of contamination. During the data generation, some libraries were likely contaminated with samples from other libraries, which caused sequences to be present in the wrong datasets. These sequences were added to a blacklist and removed from all datasets. Finally, as explained in Section 3.4, sequences with a support higher than one and observations in both the positive and negative bin were also removed.

In total, a significant amount of samples (16% to 41%) were removed for each dataset (Supplementary Table S2). This also slightly lowered the ratio of positive samples in the individual datasets up to a maximum decrease of 0.02.

5.3 Data Generation

The protocols for the generation of the different datasets were carried out before the start of this thesis, and thus we will only briefly summarise this.

All DNA fragments from the nine promoter libraries were amplified using PrimeSTAR HS DNA polymerase and purified using the innuPREP PCRpure Kit. The DNA fragments were cloned into reporter plasmids using electrocompetent *E. coli* Top10 cells, which were grown in Lysogeny Broth supplemented with kanamycin. The construction of the reporter plasmid (pLibrary) and its complete annotated DNA sequence is described in detail in Bervoets, Van Brempt, et al. (2018). Reporter plasmids were transformed into Wild Type *E. coli* K12 MG1655 cells or *E. coli* K12 MG1655 cells encoding the genes for the *B. subtilis* sigma factors. These cells were grown on Complex medium supplemented with kanamycin. Fluorescence Activated Cell Sorting (FACS) was used to sort cells into two separate bins: positives and negatives. Positives containing the cells with high red fluorescent protein expression (mKate2), and negatives those with low expression. Constitutive expression of green fluorescent protein (sfGFP) was taken into account for bin selection to exclude artifacts. The sorted cells were subsequently grown overnight in Complex medium with kanamycin, and plasmid DNA was isolated for all individual bins using a Qiagen Plasmid Mini Kit. The sorted plasmid DNA samples were prepared for sequencing with a workflow adapted from the "16S Metagenomic Sequencing Library Preparation" protocol (Illumina Inc.). Ultimately, all samples were sequenced and controlled for quality by the NXTGNT lab (Faculty of Pharmaceutical Sciences, Ugent) using a dual-index, single-read 50bp sequencing Illumina MiSeq system.

5.4 Implementation

All code was written in Python 3.6.5, making use of the Python libraries PyTorch 1.0.1.post2, NumPy 1.16.2, Scikit-learn 0.21.3, pandas 0.24.2 and matplotlib 3.0.3. The main analyses are uploaded as jupyter notebooks to Github at https://github.com/ldaveyl/master-thesis-lucas-davey together with text files and plots documenting the parameter optimisation, training and testing of all models that were reported. Furthermore, the data before and after cleaning is also available, and the individual models, multi-task models and full multi-task model were also saved as PyTorch state dictionaries, allowing them to be reanalysed at any time.

All results were obtained using the KERMIT research unit GPU server, consisting of one Nvidia Tesla K40c graphics card, and two Nvidia GTX 1080 graphics cards.

References

Alipanahi, Babak et al. (2015). "Predicting the sequence specificities of DNA-and RNA-binding proteins by deep learning". *Nature biotechnology* 33.8, p. 831.

Angermueller, Christof et al. (2016). "Deep learning for computational biology". *Molecular systems biology* 12.7.

Atsumi, Shota, Taizo Hanai, and James C Liao (2008). "Non-fermentative pathways for synthesis of branched-chain higher alcohols as biofuels". *nature* 451.7174, p. 86.

Bauer, Amy L et al. (2010). "Using sequence-specific chemical and structural properties of DNA to predict transcription factor binding sites". *PLoS computational biology* 6.11, e1001007.

Bervoets, Indra and Daniel Charlier (2019). "Diversity, versatility and complexity of bacterial gene regulation mechanisms: opportunities and drawbacks for applications in synthetic biology". *FEMS microbiology reviews* 43.3, pp. 304–339.

Bervoets, Indra, Maarten Van Brempt, et al. (2018). "A sigma factor toolbox for orthogonal gene expression in Escherichia coli". *Nucleic acids research* 46.4, pp. 2133–2144.

Bojarski, Mariusz et al. (2016). "End to end learning for self-driving cars". *arXiv preprint arXiv:1604.07316*.

Bottou, Léon and Olivier Bousquet (2008). "The tradeoffs of large scale learning". *Advances in neural information processing systems*, pp. 161–168.

Browning, Douglas F. and Stephen J. W. Busby (Jan. 2004). "The regulation of bacterial transcription initiation". *Nat Rev Micro* 2.1, pp. 57–65. ISSN: 17401526. URL: http://dx.doi.org/10.1038/nrmicro787.

Burgess, Richard R et al. (1969). "Factor stimulating transcription by RNA polymerase". *Nature* 221.5175, p. 43.

Cardinale, Stefano and Adam Paul Arkin (2012). "Contextualizing context for synthetic biology–identifying causes of failure of synthetic biological systems". *Biotechnology journal* 7.7, pp. 856–866.

Cases, Ildefonso and Víctor de Lorenzo (2005). "Genetically modified organisms for the environment: stories of success and failure and what we have learned from them". *International microbiology* 8.3, pp. 213–222.

Chawla, Nitesh V et al. (2002). "SMOTE: synthetic minority over-sampling technique". *Journal of artificial intelligence research* 16, pp. 321–357.

Christina, V, S Karpagavalli, and G Suganya (2010). "Email spam filtering using supervised machine learning techniques". *International Journal on Computer Science and Engineering (IJCSE) Vol* 2, pp. 3126–3129.

Cireşan, Dan Claudiu et al. (2010). "Deep, big, simple neural nets for handwritten digit recognition". *Neural computation* 22.12, pp. 3207–3220.

Dahl, George E, Navdeep Jaitly, and Ruslan Salakhutdinov (2014). "Multi-task neural networks for QSAR predictions". *arXiv preprint arXiv:1406.1231*.

De Mey, Marjan et al. (2007). "Construction and model-based analysis of a promoter library for E. coli: an indispensable tool for metabolic engineering". *BMC biotechnology* 7.1, p. 34.

Ghandi, Mahmoud et al. (2014). "Enhanced regulatory sequence prediction using gapped k-mer features". *PLoS computational biology* 10.7, e1003711.

Goeddel, David V et al. (1979). "Expression in Escherichia coli of chemically synthesized genes for human insulin". *Proceedings of the National Academy of Sciences* 76.1, pp. 106–110.

Haldenwang, William G, Naomi Lang, and Richard Losick (1981). "A sporulation-induced sigma-like regulatory protein from B. subtilis". *Cell* 23.2, pp. 615–624.

Hawley, Diane K and William R McClure (1983). "Compilation and analysis of Escherichia coli promoter DNA sequences". *Nucleic acids research* 11.8, pp. 2237–2255.

Ioffe, Sergey and Christian Szegedy (2015). "Batch normalization: Accelerating deep network training by reducing internal covariate shift". *arXiv preprint arXiv:1502.03167*.

James, Gareth et al. (2013). *An introduction to statistical learning*. Vol. 112. Springer.

Jensen, Peter Ruhdal et al. (1998). "The sequence of spacers between the consensus sequences modulates the strength of prokaryotic promoters". *Appl. Environ*, pp. 64–82.

Johnson, Justin M and Taghi M Khoshgoftaar (2019). "Survey on deep learning with class imbalance". *Journal of Big Data* 6.1, p. 27.

Johnson, Xanthia B and Deborah M Hinton (2006). "Escherichia coli RNA polymerase recognition of a σ70-dependent promoter requiring a- 35 DNA element and an extended- 10 TGn motif". *Journal of bacteriology* 188.24, pp. 8352–8359.

Keseler, Ingrid M et al. (2016). "The EcoCyc database: reflecting new knowledge about Escherichia coli K-12". *Nucleic acids research* 45.D1, pp. D543–D550.

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

Knight, Thomas (2003). "Idempotent vector design for standard assembly of biobricks".

Kohavi, Ron et al. (1995). "A study of cross-validation and bootstrap for accuracy estimation and model selection". *Ijcai*. Vol. 14. 2. Montreal, Canada, pp. 1137–1145.

Kroos, Lee et al. (1999). "Control of σ factor activity during Bacillus subtilis sporulation". *Molecular microbiology* 31.5, pp. 1285–1294.

LeCun, Yann, Yoshua Bengio, et al. (1995). "Convolutional networks for images, speech, and time series". *The handbook of brain theory and neural networks* 3361.10, p. 1995.

Li, Jingwei and Yunxin Zhang (2014). "Relationship between promoter sequence and its strength in gene expression". *The European physical journal E* 37.9, p. 86.

Liu, Jianhui and Jin Hou (2020). "Multidimensional Metabolic Engineering for Constructing Efficient Cell Factories". *Trends in Biotechnology*.

Lorenzo, Víctor de (2011). "Beware of metaphors: chasses and orthogonality in synthetic biology". *Bioengineered bugs* 2.1, pp. 3–7.

Martin, Vincent JJ et al. (2003). "Engineering a mevalonate pathway in Escherichia coli for production of terpenoids". *Nature biotechnology* 21.7, pp. 796–802.

Minsky, Marvin and Seymour A Papert (2017). *Perceptrons: An introduction to computational geometry*. MIT press.

Mutalik, Vivek K et al. (2013). "Precise and reliable gene expression via standard transcription and translation initiation elements". *Nature methods* 10.4, p. 354.

Nonaka, Gen et al. (2006). "Regulon and promoter analysis of the E. coli heat-shock factor, σ32, reveals a multifaceted cellular response to heat stress". *Genes & development* 20.13, pp. 1776–1789.

Olazaran, Mikel (1996). "A sociological study of the official history of the perceptrons controversy". *Social Studies of Science* 26.3, pp. 611–659.

Oubounyt, Mhaned et al. (2019). "DeePromoter: Robust promoter predictor using deep learning". *Frontiers in genetics* 10.

Paget, Mark (2015). "Bacterial sigma factors and anti-sigma factors: structure, function and distribution". *Biomolecules* 5.3, pp. 1245–1265.

Petänen, T et al. (2001). "Construction and use of broad host range mercury and arsenite sensor plasmids in the soil bacterium Pseudomonas fluorescens OS8". *Microbial ecology* 41.4, pp. 360–368.

Pribnow, David (1975). "Nucleotide sequence of an RNA polymerase binding site at an early T7 promoter". *Proceedings of the National Academy of Sciences* 72.3, pp. 784–788.

Rakhlin, Alexander et al. (2018). "Deep convolutional neural networks for breast cancer histology image analysis". *International Conference Image Analysis and Recognition*. Springer, pp. 737–744.

Rizzo, Riccardo et al. (2015). "A deep learning approach to dna sequence classification". *International Meeting on Computational Intelligence Methods for Bioinformatics and Biostatistics*. Springer, pp. 129–140.

Ro, Dae-Kyun et al. (2006). "Production of the antimalarial drug precursor artemisinic acid in engineered yeast". *Nature* 440.7086, p. 940.

Roberts, Mark et al. (2017). "The Role of Alternative Sigma Factors in Pathogen Virulence". *Foodborne Pathogens*. Springer, pp. 229–303.

Rong, Minqing et al. (1998). "Promoter specificity determinants of T7 RNA polymerase". *Proceedings of the National Academy of Sciences* 95.2, pp. 515–519.

Rosenblatt, Frank (1957). *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory.

Russell, Stuart J and Peter Norvig (2010). *Artificial Intelligence-A Modern Approach (3rd internat. edn.)*

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

Samuel, Arthur L. (1959). “Some studies in machine learning using the game of Checkers”. *IBM journal of research and development*, pp. 71–105.

Sarraf, Saman, Ghassem Tofighi, et al. (2016). “DeepAD: Alzheimer’ s disease classification via deep convolutional neural networks using MRI and fMRI”. *BioRxiv*, p. 070441.

Sonnenburg, Sören, Alexander Zien, and Gunnar Rätsch (2006). “ARTS: accurate recognition of transcription starts in human”. *Bioinformatics* 22.14, e472–e480.

Srivastava, Nitish et al. (2014). “Dropout: a simple way to prevent neural networks from overfitting”. *The journal of machine learning research* 15.1, pp. 1929–1958.

Stehman, Stephen V (1997). “Selecting and interpreting measures of thematic classification accuracy”. *Remote sensing of Environment* 62.1, pp. 77–89.

Stormo, Gary D et al. (1982). “Use of the ‘Perceptron’algorithm to distinguish translational initiation sites in E. coli”. *Nucleic acids research* 10.9, pp. 2997–3011.

Suwajanakorn, Supasorn, Steven M Seitz, and Ira Kemelmacher-Shlizerman (2017). “Synthesizing obama: learning lip sync from audio”. *ACM Transactions on Graphics (TOG)* 36.4, p. 95.

Tran, Truyen et al. (2014). “Risk stratification using data from electronic medical records better predicts suicide risks than clinician assessments”. *BMC psychiatry* 14.1, p. 76.

Umarov, Ramzan Kh and Victor V Solovyev (2017). “Recognition of prokaryotic and eukaryotic promoters using convolutional deep learning neural networks”. *PloS one* 12.2.

Weber, Wilfried et al. (2008). “A synthetic mammalian gene circuit reveals antituberculosis compounds”. *Proceedings of the National Academy of Sciences* 105.29, pp. 9994–9998.

Werbos, Paul (1974). “Beyond Regression:” New Tools for Prediction and Analysis in the Behavioral Sciences”. *Ph. D. dissertation, Harvard University*.

Wösten, MMSM (1998). “Eubacterial sigma-factors”. *FEMS microbiology reviews* 22.3, pp. 127–150.

Yim, Harry et al. (2011). “Metabolic engineering of Escherichia coli for direct production of 1, 4-butanediol”. *Nature chemical biology* 7.7, p. 445.

Yu, Hui et al. (2019). “Architectures and accuracy of artificial neural network for disease classification from omics data”. *BMC genomics* 20.1, p. 167.

Yura, Takashi and Kenji Nakahigashi (1999). “Regulation of the heat-shock response”. *Current opinion in microbiology* 2.2, pp. 153–158.

Zeng, Haoyang et al. (2016). “Convolutional neural network architectures for predicting DNA–protein binding”. *Bioinformatics* 32.12, pp. i121–i127.

Zhou, Jian and Olga G Troyanskaya (2015). “Predicting effects of noncoding variants with deep learning–based sequence model”. *Nature methods* 12.10, p. 931.

Zhou, Tianyin et al. (2015). “Quantitative modeling of transcription factor binding specificities using DNA shape”. *Proceedings of the National Academy of Sciences* 112.15, pp. 4654–4659.

Zu, Chao and Jiansheng Wang (2014). “Tumor-colonizing bacteria: a potential tumor targeting therapy”. *Critical reviews in microbiology* 40.3, pp. 225–235.

Supplementary Material

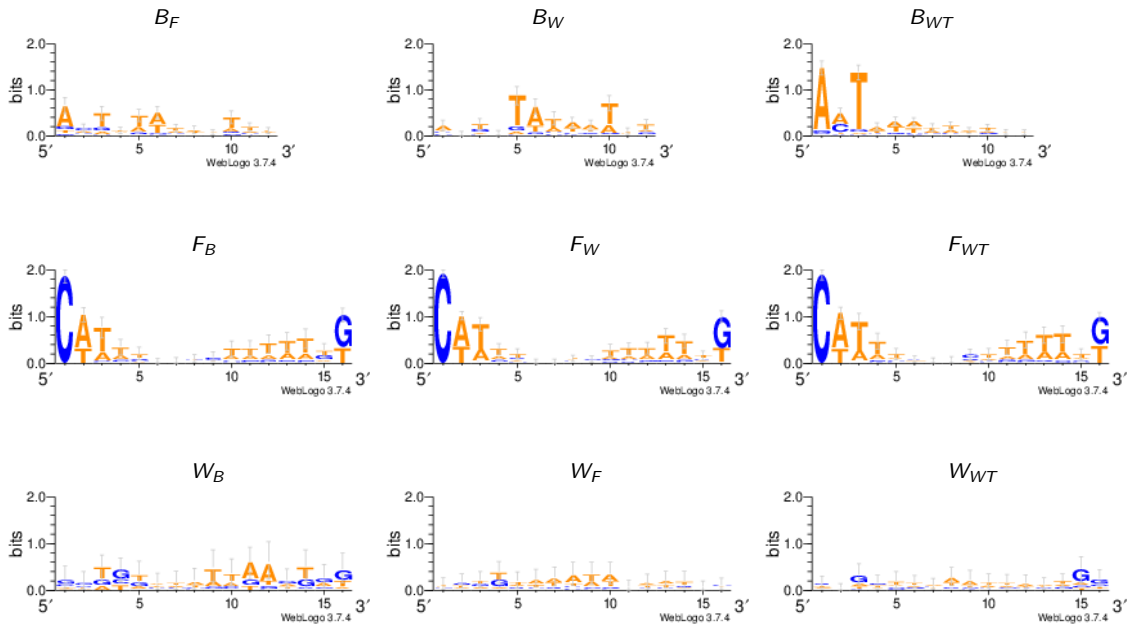

Figure S1: Sequence logos of the positive samples with the 5% highest output probabilities for the individual models. The size of the letters represents the frequency of occurrence.

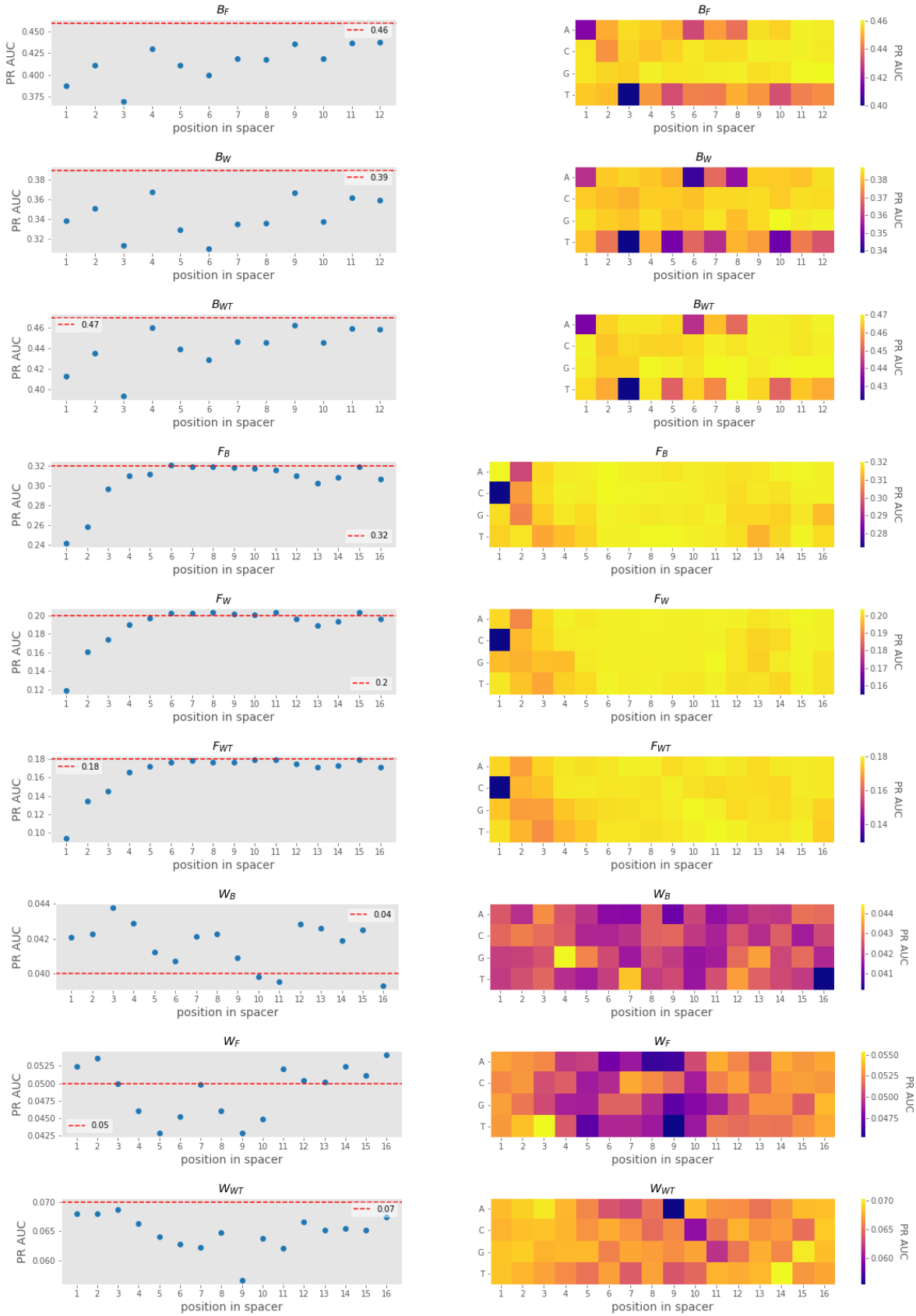

Figure S2: Occlusion maps of all testing sets using the individual models (continued on the following page). Left: Scatter plots depicting the change in PR AUC caused by occluding a position in the spacer for all samples in the testing set. The red dashed line is the PR AUC of the unoccluded test samples, and is used as a reference.

Figure S2: Right: An in-depth version of the scatter plots on the left, depicting the change in PR AUC per nucleotide. The value of every square was calculated using all test samples, but the samples corresponding with the nucleotide at a certain position were occluded. Yellow squares have relatively high PR AUC and are not as important to classification, while dark purple squares have relatively low PR AUC and are important to classification.

Table S1: Performance on every testing set with different output probabilities of the full multi-task model. The rows correspond to the testing set and the columns to which output probabilities were used. The value in row B_{WT} and column B_W represents the performance (PR AUC or ROC AUC) on the B_{WT} testing set using the output probabilities for the B_W testing set. Highlighted in yellow are the performance values that are sufficiently above the baseline: 0.5 for the ROC AUC, and the ratio of positive samples for the PR AUC.

PR AUC

	B_F	B_W	B_{WT}	F_B	F_W	F_{WT}	W_B	W_F	W_{WT}
B_F	0.46	0.46	0.47	0.10	0.11	0.11	0.40	0.40	0.41
B_W	0.40	0.41	0.40	0.08	0.08	0.09	0.34	0.34	0.35
B_{WT}	0.49	0.49	0.49	0.16	0.17	0.18	0.44	0.44	0.45
F_B	0.19	0.20	0.18	0.34	0.34	0.34	0.14	0.27	0.25
F_W	0.08	0.09	0.08	0.22	0.22	0.22	0.08	0.16	0.15
F_{WT}	0.06	0.07	0.06	0.19	0.19	0.19	0.06	0.13	0.11
W_B	0.03	0.03	0.02	0.02	0.02	0.02	0.08	0.08	0.08
W_F	0.03	0.04	0.02	0.02	0.02	0.02	0.08	0.08	0.08
W_{WT}	0.05	0.05	0.03	0.03	0.03	0.03	0.09	0.09	0.09

ROC AUC

	B_F	B_W	B_{WT}	F_B	F_W	F_{WT}	W_B	W_F	W_{WT}
B_F	0.71	0.71	0.71	0.33	0.35	0.37	0.70	0.71	0.71
B_W	0.71	0.71	0.71	0.34	0.36	0.39	0.69	0.70	0.70
B_{WT}	0.68	0.68	0.68	0.36	0.37	0.40	0.67	0.67	0.68
F_B	0.53	0.55	0.52	0.67	0.66	0.66	0.45	0.62	0.61
F_W	0.50	0.51	0.48	0.64	0.67	0.65	0.48	0.63	0.62
F_{WT}	0.49	0.51	0.47	0.64	0.64	0.65	0.46	0.63	0.62
W_B	0.50	0.48	0.52	0.37	0.39	0.39	0.63	0.62	0.62
W_F	0.50	0.49	0.52	0.37	0.39	0.39	0.65	0.65	0.65
W_{WT}	0.49	0.46	0.41	0.35	0.36	0.36	0.65	0.65	0.66

Table S2: Size of dataset and ratio of positives before and after cleaning for each dataset. Each dataset was cleaned to remove samples with erroneous sequences (PCR errors), samples that were in the wrong dataset (contamination), or samples that had labels supported by conflicting observations (observations in both the positive and negative bin). For all datasets, a significant amount of samples were removed. size: number of samples before cleaning, size cl: number of samples after cleaning, size cl %: percentage of samples after cleaning, pos ratio: ratio of positive samples, pos ratio cl: ratio of positive samples after cleaning.

	size	size cl	size cl %	pos ratio	pos ratio cl
B_F	188949	142697	76	0.16	0.14
B_W	204843	155983	76	0.12	0.11
B_{WT}	187289	135267	72	0.22	0.21
F_B	103094	74660	72	0.15	0.13
F_W	302962	255490	84	0.08	0.07
F_{WT}	309178	255072	83	0.06	0.05
W_B	138485	81247	59	0.04	0.03
W_F	257037	162535	63	0.03	0.03
W_{WT}	188078	133374	71	0.05	0.04