Techniques to Improve Deep Learning for Phenotype Prediction from Genotype Data

A Thesis Submitted to the
College of Graduate and Postdoctoral Studies
in Partial Fulfillment of the Requirements
for the degree of Master of Science
in the Department of Computer Science
University of Saskatchewan
Saskatoon

By
Logan Kopas

Permission to Use

In presenting this thesis in partial fulfilment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science
176 Thorvaldson Building
110 Science Place
University of Saskatchewan
Saskatoon, Saskatchewan S7N 5C9
Canada

Dean
College of Graduate and Postdoctoral Studies
University of Saskatchewan
116 Thorvaldson Building, 110 Science Place
Saskatoon, Saskatchewan S7N 5C9
Canada

Abstract

We show that by representing Single Nucleotide Polymorphism (SNP) data to a neural network in a way that incorporates quality scores and avoids filtering out low quality SNPs we are able to increase the effectiveness of a deep neural network for phenotype prediction from genotype in some cases. We also show that we are able to significantly increase the predictive power of a neural network by making use of transfer learning. We demonstrate these results on a Whole Genome Sequencing (WGS) *Neisseria gonorrhoeae* dataset where we predict Antimicrobial Resistance (AMR) as well as on an exome sequencing *Lens culinaris* dataset where we predict 3 growing rate phenotypes.

Acknowledgements

I would like to thank my supervisors, Dr. Anthony Kusalik and Dr. David Schneider. In addition to all of the expert advice Tony provided, he mentored me and also spent countless hours making sure my grammar was perfect. If it wasn't for Tony's acknowledgement of my work in my undergraduate thesis I may not have pursued graduate studies. Dave was also there every step of the way, ensuring that my math was sound and constantly course-correcting when I tried to lean more into the computer science aspect of this research. Dave's high performance computing expertise was also invaluable when it came to ensuring the computing power necessary for this work was available. Both of them were incredibly supportive and always willing to meet to discuss findings or new ideas.

I would also like to thank my committee members, Dr. Ian Stavness and Dr. Kirsten Bett. Both of them provided data, advice, expertise, and most importantly, were remarkable examples of academic excellence that I could only hope to emulate.

Additionally, I would like to thank my friends and family that supported me. My parents, Terry and Wes Kopas, without whom I would not likely have seen this to completion. My grandparents, Fred and Toni Ratushny, who instilled in me the value of an education early on. My friends, James Schulte and Jesse Rolheiser, who are both incredible academics in their own fields and who I hope to stay slightly more educated than.

Finally, I would like to thank P2IRC and GIFS for providing funding and resources throughout my program.

Contents

List of Tables

List of Figures

List of equations

List of Abbreviations

AMR Antimicrobial Resistance ii, 36, 39, 46, 53

BAQ per-Base Alignment Quality 16

BLUP Best Linear Unbiased Prediction 4, 14

CIR Categorical Input Representation 21, 22, 31, 35–45, 55

CNN Convolutional Neural Network 2, 5, 7, 20, 21, 31, 32, 35, 42, 43, 45, 55

DBN Deep Belief Network 5

DNP-AAP Deep Neural Pursuit network with Average Activation Potential 11

DTF Days to Flower 27, 43, 47, 52

DTM Days to Maturity 27, 43, 47, 52

DTS Days to Swollen 27, 43, 47, 50, 52

GS Genomic Selection 14

GWAS Genome Wide Association Study 14, 15

HMM Hidden Markov Model 15, 16

MIC Minimum Inhibitory Concentration 5, 24, 36–38

MNP Multiple Nucleotide Polymorphism 13

NGS Next Generation Sequencing 4, 13

PIR Probabilistic Input Representation 31, 35–45, 55

ReLU Rectified Linear Unit 6

RMSE Root Mean Squared Error 36, 37, 44, 47, 51

RNN Recurrent Neural Network 5, 20

ROC Receiver Operating Characteristics 38

RR-BLUP Ridge Regression-Best Linear Unbiased Prediction 1, 3–5, 55

SHAP SHapley Additive exPlanations 11, 44, 53

SNP Single Nucleotide Polymorphism ii, 1, 4, 5, 7, 13–22, 24, 27, 31, 32, 35, 36, 42, 45, 53, 55

WGS Whole Genome Sequencing ii, 13, 17, 24

XGBoost eXtreme Gradient Boosting 5

Chapter 1

Introduction

The goal of this thesis is to improve the current state of deep learning for phenotype prediction from genotype data.

Deep learning is a powerful tool that has made great improvements in the past decade and has led to many advancements in different artificial intelligence fields. One area that has potential for advancement by deep learning is bioinformatics. There are many areas where deep learning can be, and is being applied in the field of bioinformatics, but, specifically, deep learning using genomic information for phenotype prediction hasn't been explored to a significant degree. Accurate phenotype prediction has many useful applications. Predicting antimicrobial resistance in bacteria allows healthcare practitioners to take samples from infected individuals and prescribe the most effective antibiotics for the infection. Predicting expected yield values for agricultural crops will allow crop breeders to grow and breed the most promising lines, allowing for effective breeding programs.

Currently, deep learning methods are not able to predict phenotypes from genotype data as well as Ridge Regression-Best Linear Unbiased Prediction (RR-BLUP) or other state-of-the-art methods. This thesis looks at a few ways of improving the use of deep learning techniques for phenotype prediction from genomic data to narrow the gap between deep learning and the current state-of-the-art. While deep learning is not currently as good as RR-BLUP, deep learning has shown massive potential in other areas, and we believe that with enough work deep learning will become the state-of-the-art for phenotype prediction.

In this document we present two techniques that can be applied to deep learning to improve the performance of phenotype prediction deep neural networks. The first technique explored was the incorporation of quality score data into the input of the neural network. Our hypothesis was that having a way to incorporate quality scores and reduce the amount of information discarded by Single Nucleotide Polymorphism (SNP) filtering can increase the power of a neural network in some cases. The second technique explored was transfer learning, which is explained in Section 5.2. Our hypothesis was that leveraging transfer learning can allow us to increase the amount of information available to a neural network and increased its predictive power.

This document consists of multiple experiments on multiple datasets. In order to isolate the different experiments and prevent confusion between experiments, each experiment is presented as a whole in a single section. This means that each experiment is fully described with methodology, results, and discussion in a single place, freeing the reader from flipping between sections and possibly confusing the results of one

experiment with the methodology of another.

The purpose of this chapter is to introduce the scope and layout of this document. Next, in Chapter 2, we go through the background necessary to understand this document. Any terms not introduced in the background section can be found in the glossary, Appendix A. The background is divided into 3 sections: computer science, biology, and bioinformatics, which is the intersection of the previous two areas. Then, we discuss the motivation for this thesis in Chapter 3. Next, in Chapter 4, we explain the datasets used in the experiments in this thesis. Chapter 4 also includes the methodology used to prepare the datasets for use in the experimental section. Chapter 5 describes, in full, two separate experiments performed on both datasets. For each experiment we start with a hypothesis, the methodology is explained, the results are presented and discussed, then future work is presented. The first experiment explores the incorporation of quality score information into the input of a Convolutional Neural Network (CNN). The second experiment investigates transfer learning. Finally, we conclude the thesis in Chapter 6. The appendices include a glossary of technical terminology and extended explanations of the mathematics used in this document. Supplementary materials are provided that provide raw results and extra files that are too large to contain within this document.

Chapter 2

Background

Bioinformatics is a multidisciplinary field. As such, there are multiple background topics that must be introduced before one can understand the following document. Bioinformatics exists at the intersection of computer science and biology; both of these will be discussed first. Next, the bioinformatic topics discussed in this document will be introduced: genome-sequencing and variant-calling.

2.1 Computer Science

Computer science is a very popular and important field in the digital era. Massive amounts of data are present in all aspects of the world and being able to work with and compute on data is a prerequisite for most cutting edge research. The field of biology has not escaped this; bioinformatics and computational biology are the results of marrying computer science to biology in order to work with the ever-increasing amount of biological data available to researchers. This document looks in particular at artificial intelligence, and how it can be used to model genotype-phenotype relationships in order to predict phenotypes from genetic variation.

2.1.1 Artificial Intelligence

Within computer science there is a field called artificial intelligence. Inside of artificial intelligence the subfield we will focus on is machine learning. In it's simplest form, artificial intelligence is just a programmed set of rules that are executed in different cases in order to perform a task that, on the surface, appears to require intelligence, by some definition of the word intelligence. Machine learning is a subset of artificial intelligence. Machine learning is named as such because machine learning systems learn and improve over time. During training a machine learning system creates an internal mathematical model, which maps inputs to some output. The machine learning system then iteratively updates its internal model based on feedback from the training data. Over many training iterations, this internal model will, if successful, become good enough at mapping inputs to outputs that it can be considered useful. Additionally, this model will, hopefully, find connections between parts of the input that are important for determining the output, and will be able to use these connections to generalize the mapping from input to output so that it will be successful for inputs that were not contained in the training data. A key machine learning algorithm we will look at is RR-BLUP.

Further within the subfield of machine learning there are artificial neural networks. Deep neural networks,

and the field of deep learning that refers to them, are a specialized form of artificial neural network. Artificial neural networks encompass neural networks with any number of hidden layers; a network with more than one hidden layer is considered a deep neural network. Artificial neural networks focus on iteratively approximating complicated nonlinear functions [1]. Artificial neural networks, and by extension deep neural networks, have the ability to approximate any continuous function given enough representative capacity and are often only limited by the quality and size of the training dataset. Most of this document will focus on deep learning.

2.1.2 Machine Learning

The first machine learning algorithm that we will explore is RR-BLUP. This algorithm was created specifically for phenotype prediction and breeding prediction for genomic selection studies, which will be discussed in section 2.3.

The original algorithm, Best Linear Unbiased Prediction (BLUP), was developed by Henderson in 1984 [2]. BLUP is used to estimate random effects in linear mixed models. Henderson makes a distinction between prediction of of a random variable and the estimation of the realized value of a random variable, and in his application of BLUP to genetics he models breeding potential as the random variable to be predicted. In his example, if an animal is already born then the evaluation of its breeding value is an estimation problem, but if the goal is to evaluate the potential breeding value of two potential parents then that is a prediction problem. BLUP is used for the latter. BLUP would be considered a kernel method, which is a class of machine learning algorithm. A kernel method is a machine learning algorithm that uses what is called the "kernel trick" in order to operate in high-dimensional feature space without actually computing the coordinates of the data in that feature space. This is accomplished by using a user-defined kernel function to compute the similarity of all data-pairs in the feature space, which is less computationally expensive than explicitly converting each data point into high-dimensional space. The kernel method must then find a hyperplane that separates the data using a method like convex optimization or gradient descent.

Endelman pointed out that BLUP with mixed models is equivalent to ridge regression. He created RR-BLUP and published the R package implementation, `rrBLUP` [3]. Computing the high-dimensional feature space necessary to linearly separate genotypes with many markers would take a huge amount of computational resources. In order to avoid this computation, RR-BLUP uses a relationship model as a kernel method to find the similarities between each genotype pair. There are multiple kernel functions provided in the R package, and RR-BLUP uses either maximum likelihood or restricted maximum likelihood to optimize prediction. BLUP was developed before Next Generation Sequencing (NGS) sequencing techniques; that is, it was developed to be used for small genetic marker arrays. The combinatorial complexity that occurs from NGS sequencing with tens of thousands of SNPs makes searching genotype space in order to use BLUP impossible. Therefore, the kernel method of RR-BLUP is more appropriate for use with current sequencing technologies.

Ma *et al.* [4] compared RR-BLUP to a deep learning package they developed. They pointed out the limitations of RR-BLUP: in addition to predicting phenotypes based on linear functions of genetic markers,

RR-BLUP assumes that genetic marker effects are normally distributed with small but non-zero variance [5].

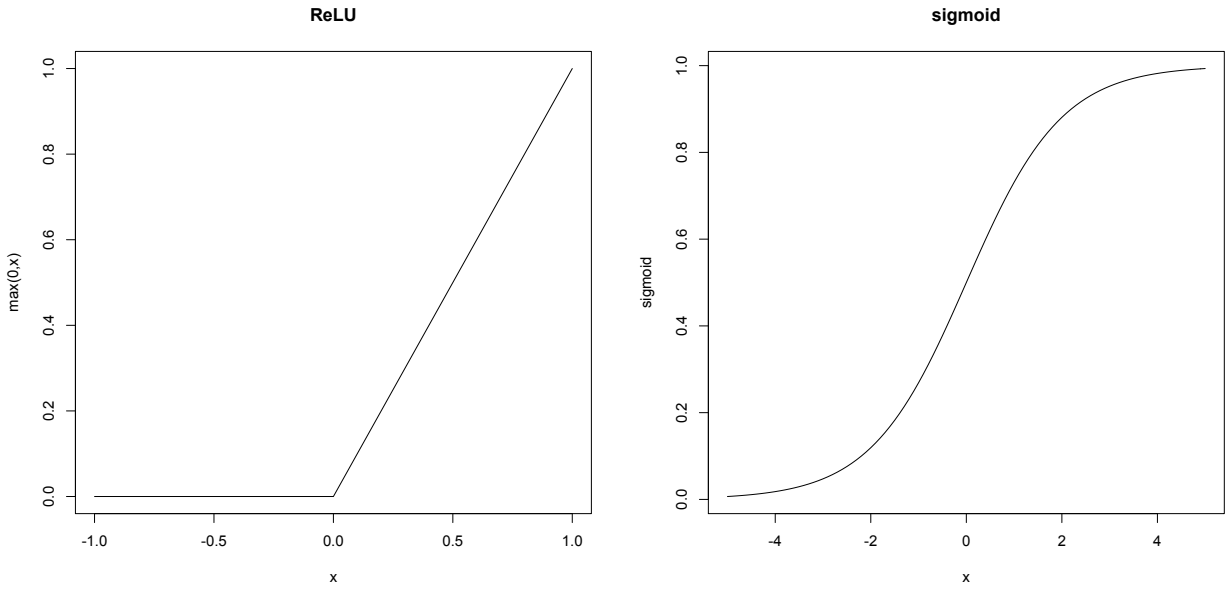
Nguyen *et al.* also found success using machine learning to predict antimicrobial resistance in *Salmonella* [6]. They used an eXtreme Gradient Boosting regressor (XGBoost), which is a decision-tree based machine learning algorithm, to predict the Minimum Inhibitory Concentration (MIC) of different strains of *Salmonella*. Antimicrobial resistance was measured for 15 different antibiotics on 5,278 different *Salmonella* strains. In their paper the accuracy was measured as the ability to predict MIC within +/- one two-fold dilution step of the laboratory-derived MIC. The two-fold dilution method is explained in Section 2.2

2.1.3 Deep Learning

Deep learning is quickly becoming the tool-of-choice for many artificial intelligence applications. Deep learning is a class of machine learning algorithms that includes Recurrent Neural Networks (RNN), Deep Belief Networks (DBN), and Convolutional Neural Networks (CNN). These networks learn simple connections at low layers, and at higher layers learn connections using combinations of the lower layers. The layers in a deep neural network are made up of neurons in a way similar to a brain. Lee [7] showed that sparse deep belief networks learn shapes and contours in a similar fashion to neurons in the visual area V2 of the human brain. A hierarchical learning algorithm such as a deep neural network is especially appealing for genomics work as complex traits are expected to result from combinations of genes as well as regulatory elements in the genome. Hierarchical learning allows lower layers to learn simple traits such as single SNPs or methylation sites, and higher layers can learn gene interactions, chromosome folding patterns, or other complex biological traits that involve many interacting pieces. In this thesis we will focus on CNNs. CNNs have been successfully applied to image classification problems, where models are trained on datasets like ImageNet [8].

A deep neural network is made up of layers. At every layer a neural network is dealing with matrices. A matrix shape is denoted by this notation: (x, y, z) where x, y, and z denote the size of each dimension. Each layer sees different types of operations performed on an $x \times y \times z$ matrix, such as convolution, dropout, or max pooling. Each of these layers is combined to transform the input into the neural network into the output, where the output can be a classification or a quantitative prediction. When a neural network is being trained, the inputs to the neural network are transformed into the outputs, then the output is evaluated against the expected output (from some ground truth), then the error is used to update the trainable layers in a step called backpropagation. The neural network is constantly evaluated against validation data, which is data that is never used for training, but is instead set aside to ensure the neural network generalizes to unseen samples. These concepts will all be discussed in this chapter.

Each trainable layer in a neural network is made up of neurons, weights, and biases. The type of layer determines the configuration and connections between neurons, as well as the function used to transform the input values, but regardless of the configuration each neuron simply stores a single value. Each trainable layer contains one or more weights and biases, which are used to transform the inputs from the connected neurons in the previous layer, and are updated as the network learns. For example, in a convolutional layer,



(a) ReLU Function **(b)** Sigmoid Function

Figure 2.1: Common activation functions are the ReLU function and the Sigmoid function.

during inference each neuron computes the dot product of the weights and the inputs, adds a bias term, and feeds the result through a non-linear activation function before the output is passed to the next layer. Non-linear activation functions are used to introduce non-linearity and allow the neural network to approximate non-linear functions. Examples of activation functions are Rectified Linear Unit (ReLU) and sigmoid, which are shown in Figure 2.1.

In order for a neural network to be effective it must first be trained. Training requires ground truth data; the neural network will attempt to predict the correct output from the input, then the error or deviation from the ground truth is used to updated the trainable layers in order to perform better on the next iteration. There are multiple hyperparameters that influence how training behaves. Batch size, number of epochs, learning rate, and early stopping functions are all examples of hyperparameters that influence training. If hyperparameters are set incorrectly this can lead to slow training or overfitting. Overfitting occurs when a neural network does not learn a generalizable mapping from significant input features to outputs, and instead just memorizes which inputs lead to which outputs. Overfitting can occur when there is not enough training data and the symptom is that the neural network performs well on the training data but not on the validation data.

When the network is being trained, the weights and biases are updated during the backpropagation step using gradient descent to reduce error as measured by the loss function. Common loss functions are root-mean-squared error (Equation 2.1) for regression tasks and binary cross-entropy for binary classification tasks (Equation 2.2).

$$\sqrt{\frac{\sum_{t=1}^{T}(\hat{y}_t - y_t)^2}{T}} \tag{2.1}$$

Root Mean Squared Error

$$-\frac{1}{T}\sum_{t=1}^{T}[y_t \log \hat{y}_t + (1 - y_t)\log(1 - \hat{y}_t)] \tag{2.2}$$

Binary Cross-Entropy

In both of these formulas, T is the total number of samples, y_t is the labelled phenotype value for sample t, and $\hat{y}_t$ is the predicted phenotype value for sample t.

A deep neural network is made up of many layers, including multiple hidden layers, stacked on top of each other, which is where the term "deep" learning comes from. There are many different types of layers that can be used in neural networks. Each layer, as well as the entire network, has multiple hyperparameters which influence how well the network is trained and how well it will perform. The first layer is just the input. In our case the input layer will be a matrix representation of SNPs. The input to a CNN is often a 3-dimensional matrix. For colour image inputs this matrix would have the shape $width \times height \times 3$, where the depth is 3 because a colour image is represent by red, green, and blue intensity values. In this document we use SNP inputs, so the input shape is $p \times 1 \times m$ where p is the number of SNPs and m is the number of possible values a SNP can have, which varies between datasets. For simplification we illustrate a $p \times 1 \times m$ matrix as the equivalent $p \times m$ matrix in this section. After the input layer there are many hidden layers that perform transformations before the output layer at the end of the neural network. There are many different types of layers that can be used for deep learning, but the 5 explored here proved effective for a CNN that fits our use-case. The different types of neural network layers can be subdivided into 2 classes: trainable and non-trainable layers.

The first class of layers are the trainable layers. Each trainable layer other than the input looks at features from the previous layer and learns the weights or connections between the current layer and the previous layer. At every learning step the weights are updated to improve the prediction accuracy by minimizing the loss function, which is why these layers are classified as trainable.

The first type of trainable layer is the convolutional layer. Convolutional layers utilize a sliding window approach using multiple filters. There are multiple parameters for a convolutional layer: convolution size, step size, depth, and activation function are just a few. The number of filters can be adjusted using the depth parameter; this parameter, along with convolution and step size, affect the number of neurons in the layer. Each filter is a different convolution with different weights that, through backpropagation, learns to detect a certain pattern or set of features. These filters are sliding windows that convolve across the input matrix and compute the dot-product of their inputs and their weights. Then, the bias term is added resulting in a single activation value for each location each filter convolves over. Then the activation value is fed through

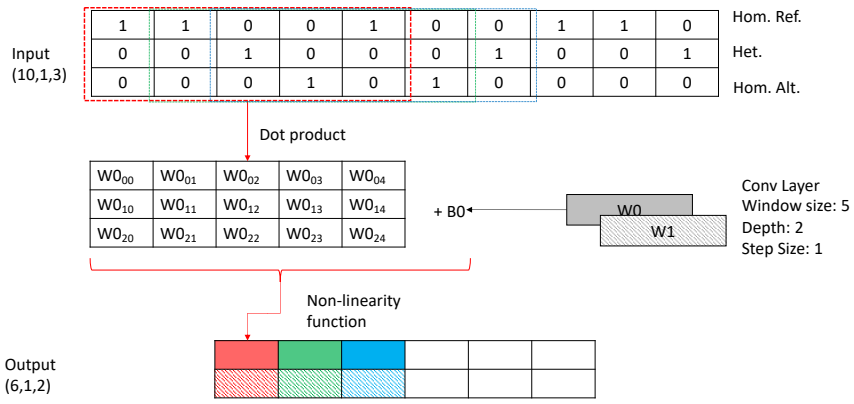


Figure 2.2: A single convolutional layer with depth 2, size 5, and stride 1. A depth of 2 means that there are 2 filters. The input shape is (10,1,3), which means the matrix is of size $10 \times 1 \times 3$; the output matrix is shape (6,1,2). Each filter looks at each position in the input and performs the dot product of the layer weights and the input, resulting in a single value for each filter that is fed through a non-linearity function before being passed to the next layer.

a non-linearity function such as the sigmoid function or, more commonly, the ReLU function, both of which are shown in Figure 2.1. This process can be seen in Figure 2.2.

In image processing applications, different filters would learn to find different shapes, line orientations, and curves. When applied to SNP data, different filters would learn to find different different granularities of SNP combinations. For example, the lower level filters would learn to recognize small numbers of SNPs that are located near one another, and every layer of filters above would build on those beneath it, building up complex networks of SNPs that span across the entire genome.

There is also a step-size parameter that determines the number of matrix positions the convolution advances at each step. A convolution with a step-size of 1 would convolve over every position in the previous layer, while a convolution with a step-size of 2 would skip every other position. Each convolutional layer may reduce the size of the input based on the step size and sliding window but may add more depth. The depth in a neural network is the size of the 3rd dimension. For RGB images the input depth is 3: one for each colour channel. For our examples the input depth is also 3 (for the diploid dataset), because when the genotype is one-hot-encoded it results in a vector of size 3 indicating whether a SNP corresponds to the homozygous reference, heterozygous, or homozygous alternate genotype.

The second type of trainable layer is the fully connected layer. The fully connected layer is like a convolutional layer except that the entire input is considered at once. In this layer each neuron is connected to every neuron in the previous layer. This results in many trainable parameters as the weight for each connection is a single trainable parameter. The features this layer detects are not limited spatially, so these

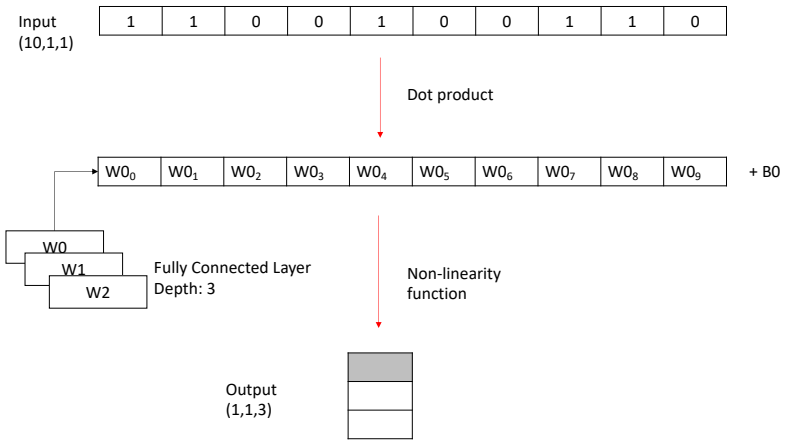


Figure 2.3: A fully connected layer is connected to every neuron in the previous layer, resulting in a large number of trainable weights. This fully connected layer has a depth of 3, meaning there are 3 filters which results in an output shape of $(1, 1, 3)$. In this case each filter shown consists of 10 weights and a bias. The dot product of the weights and the input is taken, a bias term is added, and the result is passed to the grey output neuron.

layers are often required for high level reasoning. This type of layer is illustrated in Figure 2.3. Fully connected layers are often used as the final layer in a CNN in order to bring everything together and compute the final output.

The next class of layers are the non-trainable layers. One may notice that the inputs to the convolutional layers are pictured in two dimensions, while the fully connected layer accepts a one-dimensional input. This is a slight simplification: convolutional layers can accept inputs in multiple dimensions (such as a colour image, which has 3 dimensions), but for the sake of simplification only two-dimensional inputs are shown. However, fully connected layers can only accept one-dimensional inputs; therefore, we must introduce the flatten layer. A flatten layer doesn't perform any numerical transformation, it simply reshapes the input matrix into a one-dimensional matrix. For example, if the input to a flatten layer was an $m \times n \times p$ matrix, the flatten layer would output an $mnp \times 1 \times 1 = mnp$ matrix. This reshaping is necessary for preparing inputs for fully connected layers. A flatten layer can be seen in Figure 2.4. Since we intend to use a flatten layer to prepare inputs for a fully connected layer it does not matter how the matrix is reshaped. In a fully connected layer weights are learned for every possible pairing between input and output neurons, removing all locality, so the method for flattening the input matrix does not matter.

The next two types of non-trainable layers we will look at are the max pooling layers and the dropout layers. Both of these types of layers only exist to make training a network more efficient and to prevent overfitting. The max pooling layer performs a non-linear down-sampling by taking only the maximum activation of a small area. For example, a 2×1 max pooling layer with a stride of 2 discards 50% of the

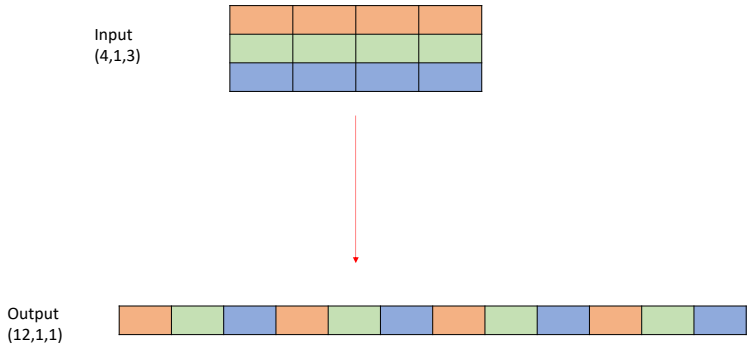


Figure 2.4: A flatten layer reshapes the inputs into a one-dimensional matrix. This is necessary to prepare inputs for use in fully connected layers.

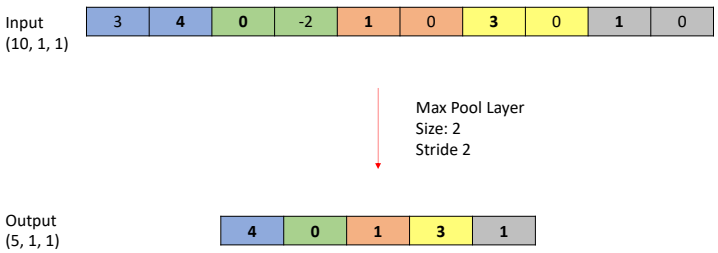


Figure 2.5: A max pooling layer with size $(2, 1)$ and stride 2. This layer passes only the largest activation values for each window to the output. Each window is shown in a different colour. The window size is 2, so the first window includes the first 2 input neurons. The next window advances 2 neurons because the stride length is 2. This means that the 2nd position of the convolution looks at the 3rd and 4th input neurons. In this case there is no overlap between windows, but if the window size was larger or the step size was smaller there would be overlap. The result is that the max of each consecutive pair of neurons is passed to the next layer, resulting in an output layer that is 50% smaller than the input layer.

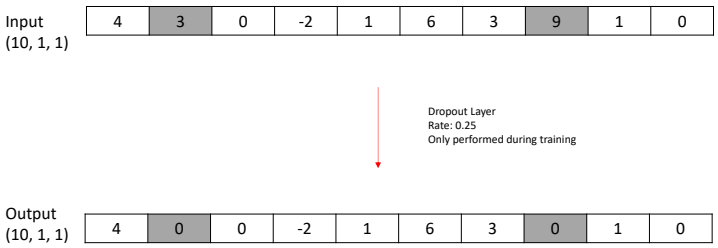


Figure 2.6: Dropout layer with 25% dropout rate. The neurons shown in grey have been selected at random to be dropped, and are replaced with a 0 in the output. This layer is included in a neural network in order to prevent overfitting and is not executed during inference.

activations by downsampling by a factor of 2 along the second dimension, which causes the next layer to be 50% smaller. This is demonstrated in Figure 2.5 where only the maximum activation value is passed on to the next layer. The depth dimension remains unchanged.

The final type of non-trainable layer is the dropout layer, which can be seen in Figure 2.6. This layer does not affect the size of the next layer, but is instead used to prevent overfitting. The dropout layer is only executed during training; during inference this layer performs the identity operation. In the dropout layer neurons are randomly "dropped out", or set to 0. The probability of keeping or dropping a neuron is indicated by the dropout rate parameter of the layer. This layer reduces overfitting by preventing the network from learning to recognize noise or irrelevant features. Additive noise occurs according to some random distribution, and if there aren't enough input samples a neural network can learn to associate noise with a certain label if they are observed together enough. By randomly removing a percentage of neuron connections during training it reduces the likelihood of the neural network observing the same noise pattern and label together and learning to associate the two. On the other hand, significant features that are always observed in conjunction with a certain label are less likely to be dropped by the dropout layer, so a neural network is able to train properly.

One issue with deep learning is that deep neural networks are mostly black boxes. Once a network is trained it is difficult to look into the network and analyze the layers to get an understanding of why the model predicts what it does, or what input elements are significant to the output. There are ways to gather insights; however, sensitivity analysis, deconvolution, and Deep Neural Pursuit network with Average Activation Potential (DNP-AAP) [9] are all ways to get a small understanding into the inner workings of a neural network [9]. SHapley Additive exPlanations (SHAP) [10] is another method of explaining a neural

network or any machine learning model. It unifies multiple different model explainers and could be leveraged to gain insight into the inner workings of a neural network.

Another deep learning technique that will be explored is transfer learning. Transfer learning is a technique that leverages existing features that have been learned by a neural network for one task and applies them to another task. A common example of this is image classification. To illustrate, we start with a neural network that was trained for a generic classification task like the ImageNet [8] contest. Neural networks trained for the ImageNet contest are readily available online. In the case of the ImageNet contest, the neural network is a very large network with many trainable layers that was trained on a very large dataset and took a tremendous amount of time and computing power to train. Training a network like this from scratch is beyond the scope of the average user, so instead they are able to download this network, fix the bottom layers (prevent them from updating during backpropagation), and retrain the top layer. The bottom layers have already learned to recognize simple features like straight lines, angles, and corners, which are usable in many image recognition tasks. The higher layers have learned connections of features that can represent more complicated objects; for example, tires, ears, and even faces. Retraining the top layers on a specialized dataset allows the network to learn to use the existing features for more specific tasks such as classifying images of dogs into the correct breed, or recognizing species of trees. This allows users to leverage deep learning for specialized tasks which may have small, niche datasets that may not be large enough for use with deep learning on their own.

Transfer learning has found many uses in image classification tasks. State-of-the-art models are available that have been trained on ImageNet, and these models can be downloaded and used as a starting point for other image classification tasks. Niazi *et al.* [11] used transfer learning for tumour identification. The model they began with was an ImageNet trained model which they then retrained to classify images of pancreatic tumours. In another example, Kim *et al.* [12] used transfer learning as well. They began with another model trained on the ImageNet dataset and retrained the model on lens-free digital in-line holography images in order to detect cells labelled with microbeads.

2.2 Biology

The focus of this thesis is on predicting phenotype information from genotype data. In an organism, gene expression is the process by which genetic information is used to create a functional product in a cell. These products can be proteins or functional RNA products. Genetic information also encodes for different regulatory elements that determine if genes are expressed. Whether a genetic variant is located in a gene or a non-coding DNA region, the genetic information is part of a pathway that may lead to a particular phenotype being expressed in the organism. Genetic variation in genes may result in different proteins being created, which leads to a different phenotype being observed. Genetic variation in regulatory locations may result in a particular protein being expressed or inhibited, which can also lead to a different phenotype being

observed. Each phenotype or trait is the result of any number of genetic variants interacting in complex ways and, given enough analytical power, is predictable to some degree from an organism's genetic information. However, a trait won't always be predictable from genotype information alone. There are many other factors that influence how a phenotype is expressed, the simplest example of which is environmental factors such as sunlight or nutrition availability. Additionally, there are other biological factors that influence gene expression but are not visible via genome sequencing such as DNA methylation or chromosomal condensation.

The genotype data this thesis will focus on is acquired by DNA sequencing. In particular, this project will be looking at SNP data. A SNP is a single nucleotide (or nucleotide pair in heterozygous organisms) that differs from the reference genome to which the sampled genome is compared. This is different from indels, which are where nucleotides are inserted into, or deleted from the reference. It is also different from Multiple Nucleotide Polymorphisms (MNP), where multiple nucleotides are changed from the reference. MNPs can be decomposed into multiple SNPs, and this is done for our *Neisseria gonorrhoeae* dataset.

2.2.1 Genome Sequencing

One of the SNP datasets used was obtained by Whole Genome Sequencing (WGS) via NGS technologies. NGS is a class of sequencing technology where high throughput and scalability allows the entire genome to be sequenced at once. NGS often uses a high sequencing depth, which means that many biological replicates of the DNA are sequenced. The sequencing depth used is the main factor in determining the accuracy of the sequenced data; a higher depth leads to higher accuracy. There are many different NGS technologies, but all of these technologies yield "short" read sequences that must be further analyzed. Prior to NGS, Sanger sequencing [13] was used which resulted in longer reads but at higher costs. Nanopore sequencing [14] is a new technology that promises very long read lengths, but few datasets are available that have been sequenced by this technology at the time of writing.

Sequencing errors come from errors in the sequencing process. These errors are dependent on the DNA sequence being read, and different sequencing platforms [15] yield different error models. Much work has been done to come up with error models for different sequencing platforms but these are all approximations. These sequencing errors need to be handled in the bioinformatics pipelines that make use of this sequence data and are often incorporated into quality estimates after analysis is completed.

The other dataset used in this document was sequenced using exome sequencing technology. Exome sequencing uses similar read lengths to WGS; however, it differs from WGS because only the exome is sequenced rather than the whole genome. The exome is the part of the genome that is made up of exons. An exon is the remaining part of mRNA after transcription and the removal of introns. The exome only consists of genetic material that is part of a gene. Since exome sequencing only looks at coding genes it may be preferable for some genetic studies; however, it will not include any genetic information that is only present in noncoding regions.

2.3 Bioinformatics

2.3.1 Genomic Selection

Genomic Selection (GS) [16] is a class of computational methods concerned with predicting the phenotype of a sample when given the genotype. GS is focused towards predictive breeding, where the genetic information of a breeding line is examined and the phenotype of that line is predicted, which allows breeders to make decisions about which breeding lines to propagate for a growing season. GS techniques are necessary for predicting polygenic traits. For simple traits simple techniques such as Punnet squares can be used, but for traits that involve the interactions of many different genes computational methods like GS are required. Punnet squares are a very simple method for predicting the phenotypes of cross breeding experiments where all possible combinations of genes are enumerated with pen and paper.

Meuwissen *et al.* [17] first coined the term genomic selection when looking at estimating the effects of genetic markers sequenced with the then-current state-of-the-art dense marker maps. The benefits of dense marker maps meant some genetic markers were close to, and possibly in linkage disequilibrium with, the quantitative trait loci. They compared multiple techniques for analyzing the quantity of data produced from dense marker maps. They found that least-squares methods did not work because of the curse of dimensionality. They also found that Bayesian methods are able to outperform BLUP techniques because of their ability to account for haplotypes.

Zenger *et al.* [18] looked at the potential commercial benefit of GS for aquaculture. They found that, while there were many obstacles to designing breeding programs based on GS, the declining cost of genomic sequencing allowed GS to become a viable option for increasing breeding program efficiency. Some of the obstacles included GS for polygenic traits, issues predicting disease resistance because of the difference between data collection environments and commercial farms, and the difficulty and importance of incorporating familial relationships into the GS analysis. Simply predicting phenotypes is not always sufficient; it is often important to know what SNPs cause a particular phenotype, which brings us to Genome Wide Association Study (GWAS).

Neural networks trained for phenotype prediction can also be analyzed to extract significant SNPs as well. Shi *et al.* [9] showed that a technique called DNP-AAP could be used to find a ranked list of the most significant SNPs involved in antimicrobial resistance for *N. gonorrhoeae*. In the aforementioned paper, DNP-AAP is discussed in the context of GWAS, but the technique crosses the boundary between GWAS and GS. The difference between GS and GWAS studies is that while GS studies focus on predicting phenotypes from genetic information of some sort, GWAS studies focus on finding the genomic feature(s) associated with a phenotype, such as the SNPs or alleles that are associated with a trait. In the Shi *et al.* paper [9], a neural network was trained to predict a phenotype and found the most significant SNPs for predicting that phenotype. Because of the predictive neural network, it can be argued that the paper has roots in GS as well

as GWAS.

2.3.2 Sequence Alignment

If the read fragments from NGS technologies are to be used for variant-calling (which will be discussed in subsection 2.3.3), they must be aligned to a reference genome. Some deviation from the reference is expected in the reads, either from genetic variation or from sequencing errors, so the alignment algorithm must account for this variation. Because of this variation, the genome built from the aligned reads is subject to error, and the alignment program must give some indication of the mapping quality. Further complicating this, genetic duplication and similar genetic regions in the reference genome means that reads may align to multiple places on the reference genome, and therefore more estimations and uncertainties are introduced into the alignment. Heterozygosity also makes alignment difficult as the reads from each chromosome pair (for diploid organisms) will differ and therefore be harder to align.

Haplotype estimation, or phasing, can also be done on the raw reads. This is done by using statistical methods to estimate which alleles are inherited together from the parents. Haplotype estimation is important because it allows researchers to leverage linkage disequilibrium to improve the statistical power of variant-calling software.

All of these factors are considered and reported as quality scores. Quality scores can be presented at many stages in the analysis pipeline; for example, raw reads have an associated quality score, alignments have a quality score, and SNPs are also reported with quality scores.

2.3.3 SNP Callers

The following 2 subsections will briefly introduce the 2 SNP calling software packages used and how quality scores are calculated for each package. FreeBayes excels by incorporating the entire population dynamics to look at the *a priori* allele frequencies and to estimate haplotypes which allows for more statistical power. Samtools excels by using a Hidden Markov Model in order to improve accuracy when it comes to indels. A more comprehensive look at the mathematics involved is presented in Appendix B.

FreeBayes

The *N. gonorrhoeae* dataset, which is described in Section 4.2, was SNP-called using FreeBayes [19]. FreeBayes provides users with quality scores for each SNP that is derived using the posterior probability of the genotype, which will be explained in this subsection. FreeBayes also provides a genotype likelihood score for the non-called genotypes at each loci, which is necessary for our data representation for our models, as explained in Subsection 5.1.2.

FreeBayes uses Bayesian statistics to compute the posterior probability of each genotype given the observed reads. This posterior probability is related to both the *a priori* expectation of the allele distribution

within a population as well as the observed sequence quality. G_i is the true genotype of sample i at a given locus, R_i is the set of observed reads for sample i at the same locus, and the posterior probability is

$$P(G_1, ..., G_n|R_1, ..., R_n) = \frac{P(G_1, ..., G_n)P(R_1, ..., R_n|G_1, ..., G_n)}{P(R_1, ..., R_n)}. \tag{2.3}$$

$P(R_1, ..., R_n|G_1, ..., G_n)$ represents the likelihood that our observations match a specific combination of genotypes and $P(G_1, ..., G_n)$ is the prior likelihood of observing a specific genotype combination. If we assume that sequencing reads are independent, then $P(R_1, ..., R_n|G_1, ..., G_n) = \prod_{i=1}^{n} P(R_i|G_i)$. This assumption is not always true (depending on the sequencing technology used), but this term is ultimately estimated in FreeBayes as calculating it directly is intractable. In order to find the likelihood of the reads, $P(R_1, ..., R_n)$, we can take the likelihood of the reads given a genotype, $\prod_{i=1}^{n} P(R_i|G_i)$, and sum that across all possible genotypes. This allows us to decompose Equation 2.3 into

$$P(G_1, ..., G_n|R_1, ..., R_n) = \frac{P(G_1, ..., G_n)\prod_{i=1}^{n} P(R_i|G_i)}{\sum_{\forall G_1,...,G_n}(P(G_1, ..., G_n)\prod_{i=1}^{n} P(R_i|G_i))}. \tag{2.4}$$

In order to estimate $P(R_i|G_i)$, mapping quality and sequencing quality are considered. In other words, the probability of obtaining a set of reads given an underlying genotype is proportional to the probability of sampling a set of observations from the underlying genotype, scaled by the probability that the reads are correct.

Further explanations of the calculations used by FreeBayes can be found in Section B.1 as well as the original FreeBayes paper by Garrison and Marth [19].

Samtools

The Samtools `mpileup` function is another SNP calling package used in this thesis. While FreeBayes is largely considered superior to `mpileup`, FreeBayes is considerably slower than `mpileup`. This tool is used to compute the SNPs and their quality scores for the *Lens culinaris* dataset. The `mpileup` function makes use of the work of Li *et al.* [20]; a Hidden Markov Model (HMM) is used to predict whether a genetic variant was the result of a SNP or an insertion or deletion. This reduces false SNP calls because of misalignments around insertions and deletions.

The `mpileup` program computes a set of BAQs, or per-Base Alignment Qualities, for each called SNP. BAQ is defined to be $-10\log_{10}[1 - Pr(x_i|A)]$ where x_i is the probability that the i-th read base is correctly aligned to position i within the entire alignment, A. This takes into consideration the mapping quality of read i, which is provided by the mapping software, and also takes into account the HMM model of the entire alignment.

The BAQ is provided not just for the called genotype, but also for the alternative genotypes. More information on how these values are calculated can be found in Section B.2, and in the paper by Li that introduces BAQ [20].

Chapter 3

Motivation

Deep learning is a tool that shows tremendous potential for genomic studies, but there are no accepted best-practices for deep learning-based genomic studies. For example, the best way to represent genomic data to a deep learning system is unknown. Deep learning for genomics is also difficult because the datasets that are available are not suited for deep learning. SNP data is error-prone, full of noise, difficult to work with, and expensive to obtain for large-scale studies with many samples. Because of this, most WGS datasets suffer from the curse of dimensionality; that is, if n is the number of individuals in a dataset, and p is the number of attributes considered for an individual (the number of genetic features sequenced), then $p >> n$. This makes it extremely difficult to train machine learning algorithms.

Deep learning has been successfully applied to many areas of bioinformatics. There are many examples where CNNs have been applied to image data, such as Chilamkurthy's work extracting features from head CT scans [22] and Dubost's work with brain MRI images [23]. DeepChrome is a CNN framework that classifies gene expression from histone modification data [24]. RNNs are often used when working with sequential data, such as when looking at DNA and RNA binding sites, as shown by Jolma [25] and by Alipanahi [26]. Hua used a combination of a CNN and a DBN to diagnose lung cancer in images [27]. These are just some recent examples of deep learning applied to bioinformatics problems.

Deep learning models, specifically CNNs, are often used with images as inputs. Representing an image to a neural network is straightforward: the image is represented as a matrix of pixel values and the input layer for the network feeds the input matrix into the next layer of the network. The input matrix differs slightly depending on whether the image is greyscale, black and white, or colour, but in any case the matrix is a direct representation of the pixels in the image; that is, each pixel's intensity value is spatially located in the input matrix next to adjacent pixels' values, and a pixel's intensity value can be directly represented in the input matrix. For black and white images, the input matrix is a matrix with the same dimensions as the image and each position in the matrix is a 0 or a 1, indicating that the corresponding pixel is either black or white, respectively. Greyscale images are similar, except the values at each position are an indication of the intensity value of the pixel. Most commonly this is a decimal number inclusively between 0 and 1, or an integer in the range of 0 to 255, where 255 is a white pixel and 0 is a black pixel. These values are often normalized to a decimal scale from 0 to 1 when used in machine learning. Colour images are similar, but each pixel is represented as a vector of size 3: one value for each colour channel (RGB); this can be seen in

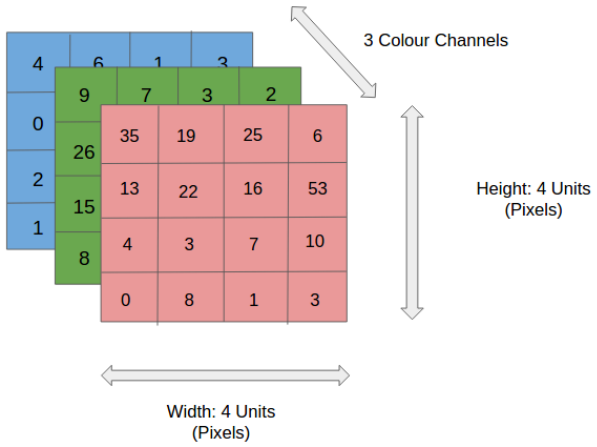


Figure 3.1: A typical input matrix for a colour image for use in a neural network. There is one layer for each colour channel: red, green, and blue. In this case the image is a $4 \times 4 \times 3$ pixel image. Source: XRDS [21].

Figure 3.1. Note that in this figure, the input appears to be a single plane for each colour channel instead of a vector for each pixel. It is equivalent to say the entire input is a 3-dimensional matrix; each pixel is a $1 \times 1 \times 3$ vector and there are $width \times height$ of these vectors, resulting in a $width \times height \times 3$ matrix.

Genomic data is not as straightforward as images; there is no such intuitive way to represent a set of genomic markers as a standard matrix. Neural networks must have a consistent input matrix size across all samples, so a standard representation must be used that is flexible enough to handle the variation that often occurs in genomic datasets. Not all samples always have the same SNP set, so the chosen data representation strategy must also be able to handle inconsistent SNP sets.

Biological data is messy, and genomic data is no exception: when it comes to SNP data each called SNP has an associated quality score and there is a lot of potential variation in quality across a dataset. Oftentimes low quality SNPs are filtered out of a dataset; some SNPs may be removed for all samples, and some may only be filtered out for a fraction of samples. Different experiments can also yield different numbers of SNPs depending on the sequencing platform and experimental methods used. This causes inconsistent inputs because each sample might have a different set of SNPs that pass quality filtering, so data representation strategies must be able to handle this. Additionally, information is thrown out that could be potentially useful when filtering out SNPs. This is very similar to the common statistical problem of missing or censored data. On top of that, quality scores are disregarded once the filtering is complete. Having a way to incorporate quality scores and reduce the amount of information discarded would potentially increase the power of a neural network.

When we use deep learning to predict a phenotype from genotype data, rare SNPs might actually be associated with the phenotype that we are trying to predict. These rare SNPs can be uncommon minor alleles with a low minor allele frequency, or they can even be alternate alleles that are not considered the minor allele. If a SNP s has a low prevalence compared to other SNPs, it will inherently have a lower quality

score because it has a smaller allele frequency, which decreases $P(G_s)$, and from equation 2.4 the posterior probability of the rare genotype is decreased. If this SNP is filtered out then the deep learning model will have troubles linking phenotypes to that genetic marker if it is a causal SNP.

In addition to having a standardized way of representing SNP data to a deep learning model, being able to deal with the curse of dimensionality would tremendously improve the performance of deep learning for phenotype prediction. We present transfer learning as a way to deal with this problem. One use case that can be imagined is to have a centralized repository of trained deep learning models that can be used as a starting point for future genomic studies. A researcher hoping to use a specialized dataset only consisting of a small number of individuals could download a model that was already trained for use on the same or a closely related species and could then retrain his/her model on the specialized dataset. This would allow for faster training times and more effective models than the researcher could have created just by using the specialized dataset on its own, as long as the phenotypes are related enough to make use of transfer learning.

Successfully applying deep learning to phenotype prediction will benefit plant breeding programs by allowing for more effective phenotype predictions from genomic data. This means that breeders can better predict which lines from their breeding stock have the best phenotype potential for the trait they are interested in, and can grow and breed a more targeted selection of lines to release or to breed again next season. Similarly, analyzing predictive neural networks may result in the discovery of novel SNPs that are significant for the prediction of phenotypes. These novel SNPs can then be investigated biologically to determine the role they play in the development or regulation of the trait in question.

Chapter 4

Data

The biggest hurdle that must be overcome in order to use deep learning techniques for phenotype prediction is finding a dataset with enough samples and corresponding phenotype labels in order to train a neural network to be effective and generalizable. There are many datasets available with large numbers of sequenced individuals, but most of these datasets do not have consistent phenotype information for the corresponding individuals. For this work it was important that the genomic data be in the form of SNPs with quality scores and genotype likelihood information, or that the raw reads were accessible so the data could be variant-called with FreeBayes. Datasets with around 1000 different samples and corresponding phenotypes were sought, but datasets that large were not found at the time of writing. As a result of using datasets with fewer than 1000 different samples, there were problems with overfitting and small validation sets, making it difficult to create generalizable prediction models.

The number of SNPs required depends on the size of the genome and the distribution of the called SNPs. If the SNPs are uniformly distributed across the genome, a small number of SNPs from various parts of the genome can capture more genetic variation than many SNPs all located in a small portion of the genome. An organism with a smaller genome also requires fewer SNPs to create a comprehensive sample of the genetic variation within the organism than an organism with a larger genome. Alternatively, a dataset with fewer SNPs takes less memory and less time for neural network training, so datasets with 10s of millions of SNPs are not desirable because of the computing resources they require.

Most importantly, phenotype information must be available and associated with each sample. A machine learning model, such as a neural network, learns to predict labels based off inputs. For each individual in a dataset, the phenotype is the label and the genotype is the input. Labelled samples are used for training and validation, so that once a neural network is trained it can predict the phenotype information for unlabelled samples. Without labels each genotype cannot be used for either training or validation, and therefore is useless unless labels can be generated some other way. Phenotypes can be numerical, such as height, or categorical, such as eye colour. Phenotypes can also include a time element, so long as there is a time that can be sampled consistently (e.g. height at 10 weeks old). The neural networks in this thesis do not handle phenotypes with time series data, such as a consistent sampling of height over the lifetime of an individual. CNNs do not handle time series data well; RNNs [28] are often used for working with time series data, which goes beyond the scope of this document.

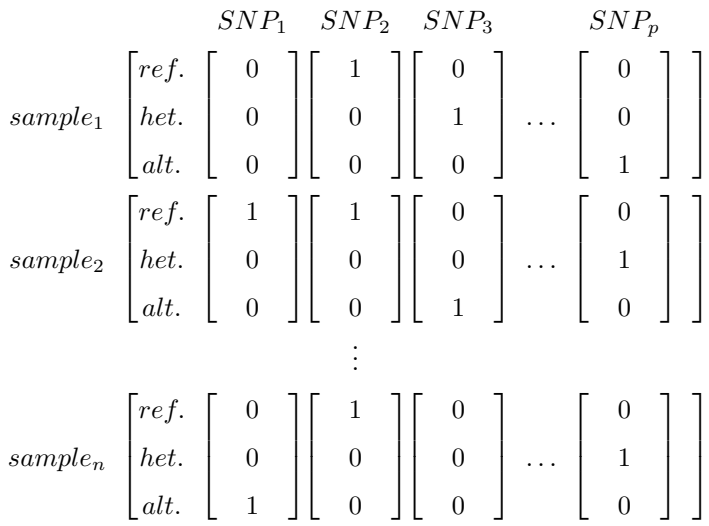


Figure 4.1: Input matrix for CIR. Each SNP is represented by the one-hot encoding of its genotype classification. The abbreviations $ref.$, $het.$, and $alt.$ correspond to the homozygous reference, heterozygous, and homozygous alternate respectively. Notice that SNP_1 for $sample_1$ and SNP_3 for $sample_n$ did not meet the quality threshold and were filtered out.

4.1 SNP Representation

Before continuing, one-hot encoding must be explained. One-hot encoding is when a categorical variable is encoded as a binary vector. Genotypes can be considered categorical data with 4 classes, [homozygous reference, heterozygous, homozygous alternate, unknown], but instead of representing homozygous reference as 0, heterozygous as 1, homozygous alternate as 2, and unknown as 3 we encode them as a vector (homozygous reference = [1, 0, 0], heterozygous = [0, 1, 0], homozygous alternate = [0, 0, 1], and unknown = [0, 0, 0]). Each vector can have at most a single 1, with the remaining elements being 0. This prevents inadvertently introducing magnitude into the input representation. Since in CNNs many operations are multiplicative it is misleading to encode homozygous alternate as 2, because that implies that homozygous alternate is twice the value of heterozygous, which we represent as 1. One-hot encoding avoids this. This input representation can be seen in Figure 4.1, and is similar to the way images are represented to a neural network. Where an image is represented as three 2-dimensional matrices stacked on top of each other (one for each of red, green, and blue values), this SNP input representation can be thought of as three 1-dimensional matrices stacked on top of each other (one for each genotype category). This is the basic way of encoding SNPs and will be referred to as Categorical Input Representation (CIR) from this point on. An alternative SNP input representation is looked at in Section 5.1.2.

We are interested in determining how the quality of a dataset affects a CNN's ability to predict phenotypes. Other factors that will affect a CNN's performance include: phenotype measurement quality, the genes

associated with the phenotype, distance between measured SNPs and causal genes, the neural network architecture, and the neural network hyperparameters. All of these factors must be kept as constant as possible when analyzing each experimental variable's effectiveness so they do not confound the results.

4.2 *Neisseria gonorrhoeae*

The first dataset explored is the combination of multiple *Neisseria gonorrhoeae* datasets. Each sample in this dataset consists of a pair of both genome and phenotype data. The phenotype data is the antimicrobial resistance levels for multiple antibiotics. Demczuk *et al.* published a dataset of *N. gonorrhoeae* strains sampled in Canada from 1997 to 2014 [29]. Grad *et al.* published a similar dataset sampled from the USA [30]. De Silva *et al.* collected samples in the UK from 2011 to 2015 [31]. These three datasets were combined to form a dataset containing 41,502 SNPs across 676 different *N. gonorrhoeae* strains. *N. gonorrhoeae* is a good starting dataset because *N. gonorrhoeae* has a small genome with only a single chromosome. The small size allows for short neural network training times, meaning that development can happen rapidly and iteratively, therefore allowing preliminary results to be obtained quickly. *Gonorrhoaea* is a haploid, meaning complexities such as heterozygosity didn't need to be considered at this step.

The dataset was first prepared by downloading the raw reads from the NCBI short read archive (SRA). A list of accessions numbers can be found in Table C.1. Next, the reads were aligned to reference genome NCCP11945 [32] using the Burrows-Wheeler Aligner [33]. The resulting `sam` files were sorted and SNPs were called with FreeBayes. The script `preprocess.sh` in Listing E contains the parameters each script was called with. The drawback of using FreeBayes is that it takes a long time to run on a large dataset. It took 2 months for a machine with 64 CPU cores and 1.5TB of memory to produce SNPs for the *N. gonorrhoeae* dataset using FreeBayes; unfortunately, CPU time was not calculated during execution. Once the SNPs were called, indels were removed. Next, the resulting variants were filtered for sequencing depth greater than 4 and quality greater than 20 using the `vcffilter` tool (`https://github.com/vcflib/vcflib#vcffilter`).

The CIR presented in Section 4.1 considered only diploid datasets. *N. gonorrhoeae* is different because it is haploid. In order to encode SNPs the same way, 0 is assigned to the reference genotype, 1 is assigned to the first (most common) alternative genotype, 2 is assigned to the second most frequent alternative genotype, and so forth. This is decided and provided by the SNP-calling software and allows us to use a similar representation scheme. The maximum number of alternate genotypes in this dataset is 6, so our vector after one-hot encoding must be of length 7 (6 alternates plus the reference genotype). The reason there are more than 3 possible alternates is because in some cases neighbouring SNPs are aligned to the same reference. This leads to the possibility of many different alternates, but in this dataset the most we see is 6 alternates aligned to a single reference location. For example, the reference sequence `CGC` may have both `CTC` and `CGT` as alternates. These are still both SNPs because only one nucleotide is different from the reference. If 2 nucleotides were changed this would be decomposed into separate SNPs.

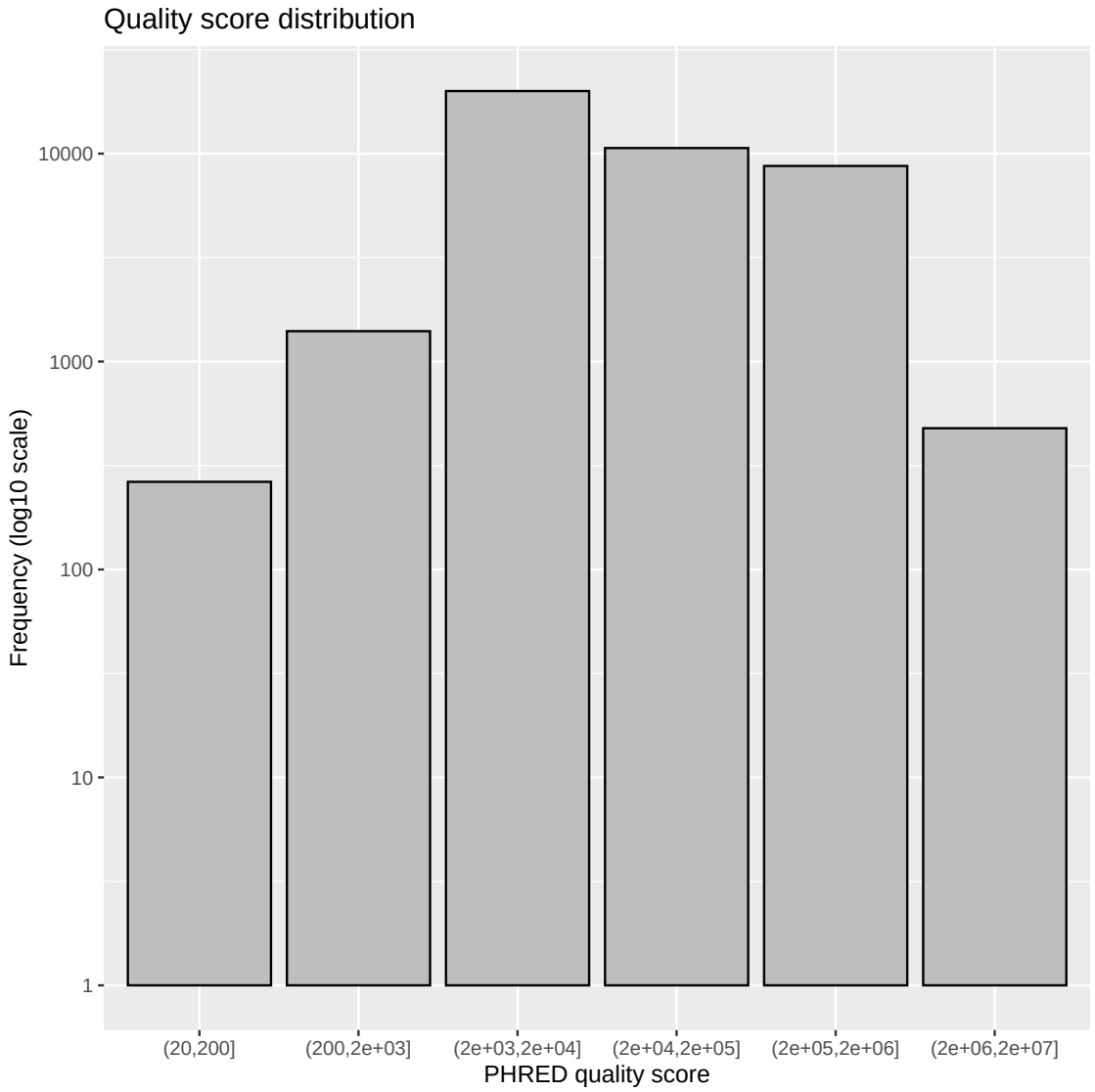
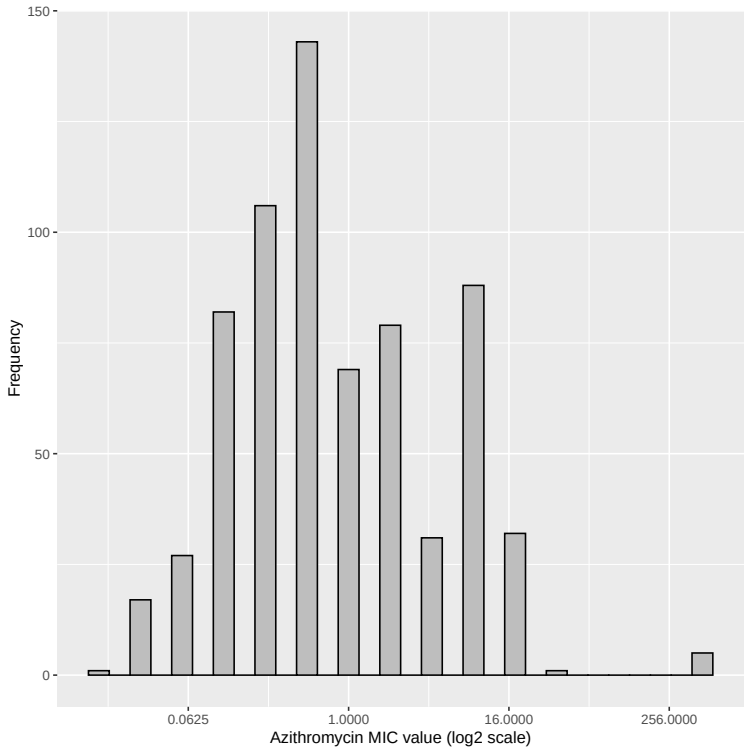
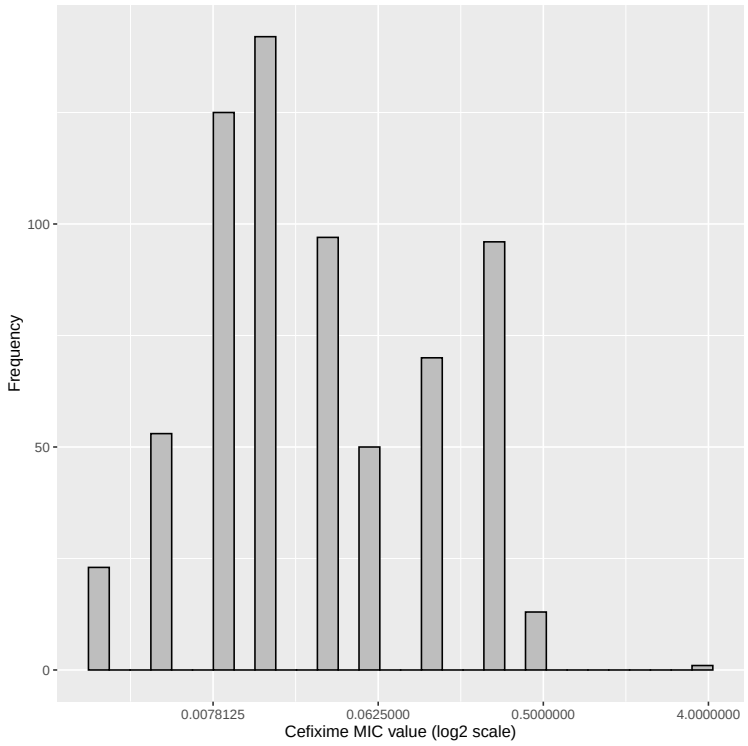


Figure 4.2: Histogram of distribution of quality scores for the *N. gonorrhoeae* dataset. Note that both axes use a logarithmic scale.

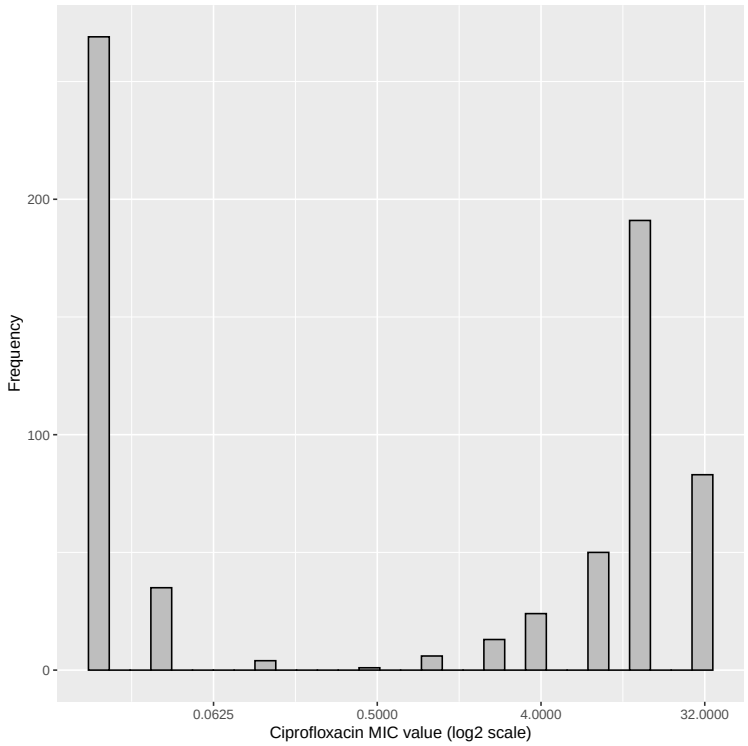


(a) Azithromycin MICs

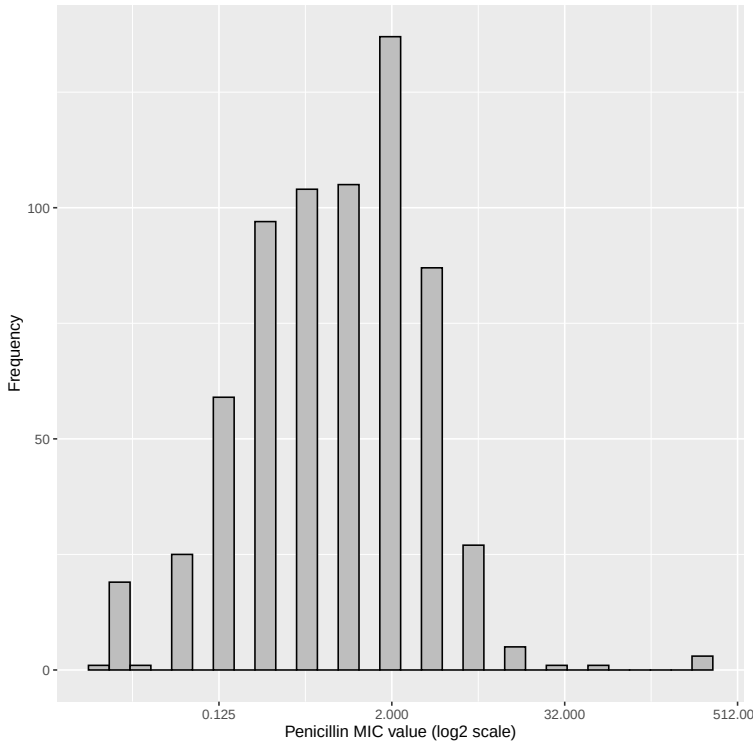


(b) Cefixime MICs

Figure 4.3: Histogram of distribution of MIC values for the *N. gonorrhoeae* dataset. The MIC for each antibiotic were measured with the two-fold dilution method, resulting in values that fall on the $\log_2$ scale.

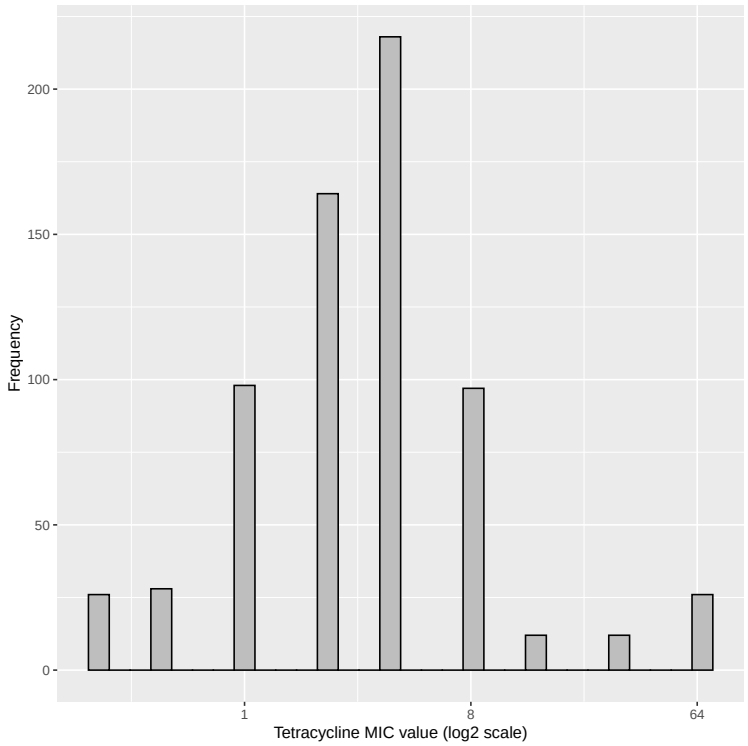


(c) Ciprofloxacin MICs



(d) Penicillin MICs

Figure 4.3: Histogram of distribution of MIC values for the *N. gonorrhoeae* dataset (cont.).



(e) Tetracycline MICs

Figure 4.3: Histogram of distribution of MIC values for the *N. gonorrhoeae* dataset (cont.).

There is a large range of quality scores of SNPs in this dataset, with many SNPs having very high scores. Figure 4.2 shows the quality score distribution of the SNPs after filtering.

The phenotypes for the *N. gonorrhoeae* dataset are the minimum inhibitory concentrations (MICs) of each of 5 antibiotics: azithromycin, cefixime, ciprofloxacin, penicillin, and tetracycline. This is a quantitative measurement of antimicrobial resistance of each *N. gonorrhoeae* strain and is measured using the two-fold dilution method. The two-fold dilution method [34] is a method of measuring antimicrobial susceptibility that involves doubling the concentration of antibiotic solution in each test until the lowest concentration that inhibits growth is found. This will produce MIC values that follow a log_2 scale, *i.e.* $\{\ldots, 0.25, 0.5, 1, 2, 4, 8, \ldots\}$. The distribution of MIC values for each phenotype can be seen in Figure 4.3. For some phenotypes there were missing MIC values: there were 6 missing MIC values for cefixime and 4 missing MIC values for penicillin.

4.3 *Lens culinaris*

While *N. gonorrhoeae* is a good organism to use as a starting point for phenotype prediction studies, organisms used in agricultural crops are more complex and will lead to more practical applications. After studying *N. gonorrhoeae*, a *Lens culinaris* dataset was used. *L. culinaris* is considered to show that any findings are

not only useful for simple bacteria, but can also be extended to diploid organisms and may be useful in agricultural applications. Phenotype prediction with agricultural organisms is useful for both crop breeders and producers, allowing them both to make the most of the growing season by planting genetic lines that have the most potential for whatever trait the breeder or producer desires.

The AGILE lentil dataset made available by Kirstin Bett [35] contained 324 samples with 31,883,135 SNPs sequenced via exome sequencing (This particular dataset was called with reference genome version 1.2 [36]). Exome sequencing is different from WGS because it is designed to only look at DNA that will be translated into mRNA. WGS, by virtue of its name, looks at genetic variants across the whole genome. Exome sequencing disregards genetic variants in noncoding regions and in introns and only looks at variants in coding regions. Exome reads were aligned to a reference genome using the Samtools `mpileup` [37] function. The `mpileup` function produces quality scores with genotype likelihoods for each possible genotype as described in Subsection 2.3.3. FreeBayes was not used for this task because of the massive amount of time it would have taken to process the SNPs for such a large dataset. While the `mpileup` quality score calculation differs from the one used in `freebayes`, it was still useful for our purposes as we required genotype likelihoods for each possible genotype, and both `freebayes` and `mpileup` quality scores can be converted to genotype likelihoods using Equation 5.1.

Working with such a large dataset takes considerable computing resources, specifically program memory, and requires some special consideration when building neural networks. For example, the network we used had 1,946,345,473 trainable parameters, taking approximately 7.8 GB of memory to simply represent in memory. Much more memory than this was required to work with and train the network. These neural networks must be run on the CPU rather than a GPU as transferring data to the GPU incurs too much overhead. After filtering out SNPs with quality scores less than 30, 23,402,924 SNPs remained. In order to cut the dataset size down even further, any SNPs that were missing from any of the 324 samples were filtered out completely, leaving 950,298 SNPs. The quality score distribution of the remaining SNPs can be seen in Figure 4.4.

L. culinaris is a diploid organism so SNPs are represented as a vector of length 3 as described in Chapter 4.1.

There are 3 phenotypes associated with each *L. culinaris* line of interest to us. The first phenotype is the number of Days to Flower (DTF). This is defined as the number of days until 10% of plants have at least one open flower. The next phenotype is the number of Days to Swollen (DTS). This is defined as the number of days until 10% of plants have fully swollen pods. The final phenotype is the number of Days to Maturity (DTM), which is defined as the number of days until 10% of plants have half of their pods mature. Each *L. culinaris* line has been planted in multiple locations across multiple years, so each replicate has different environmental conditions resulting in different phenotype values depending on the year and location of the crop. For simplicity we focus on only the phenotypes from the Sutherland 2016 crops. The distribution of these phenotypes can be seen in Figure 4.5

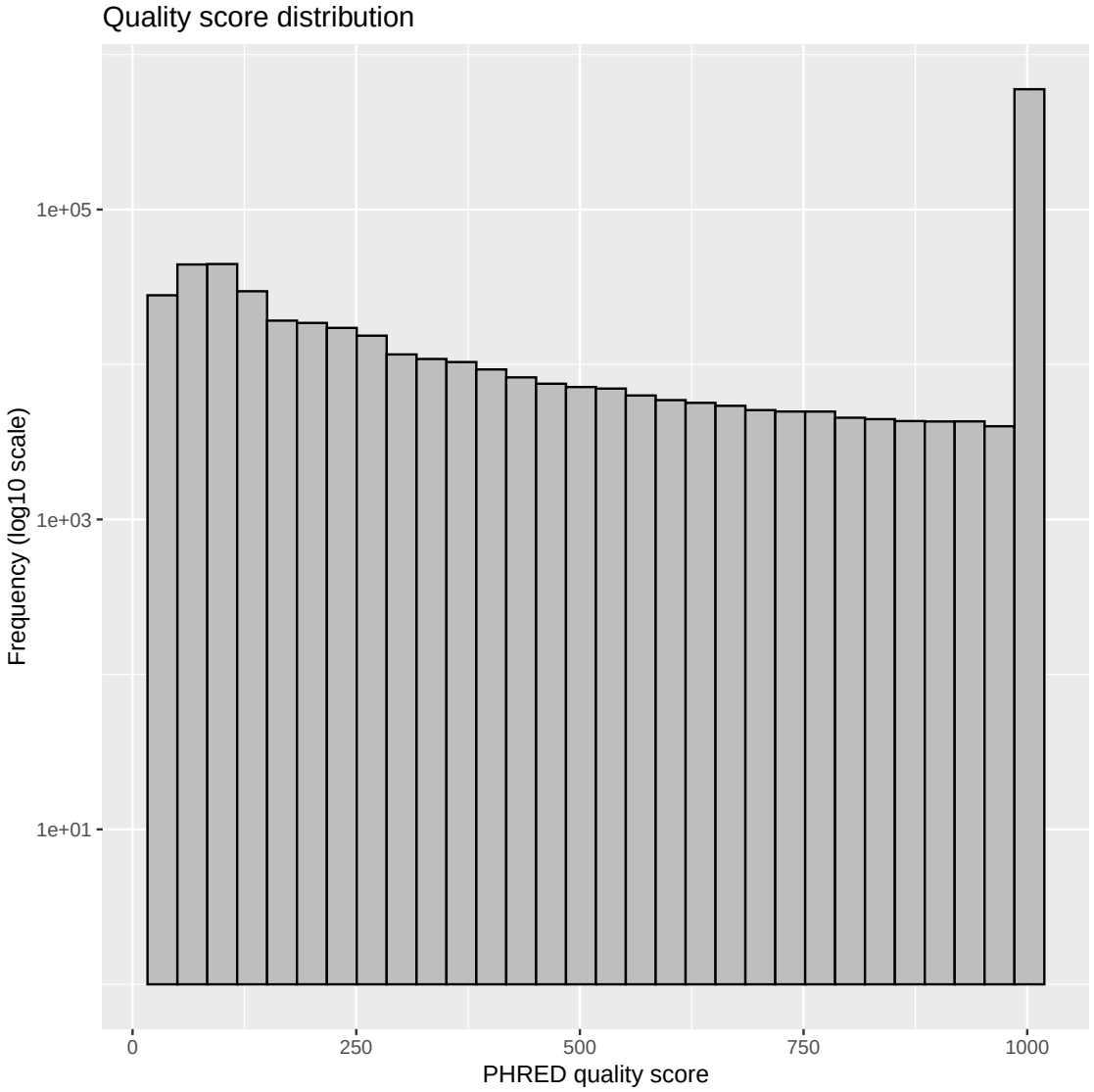
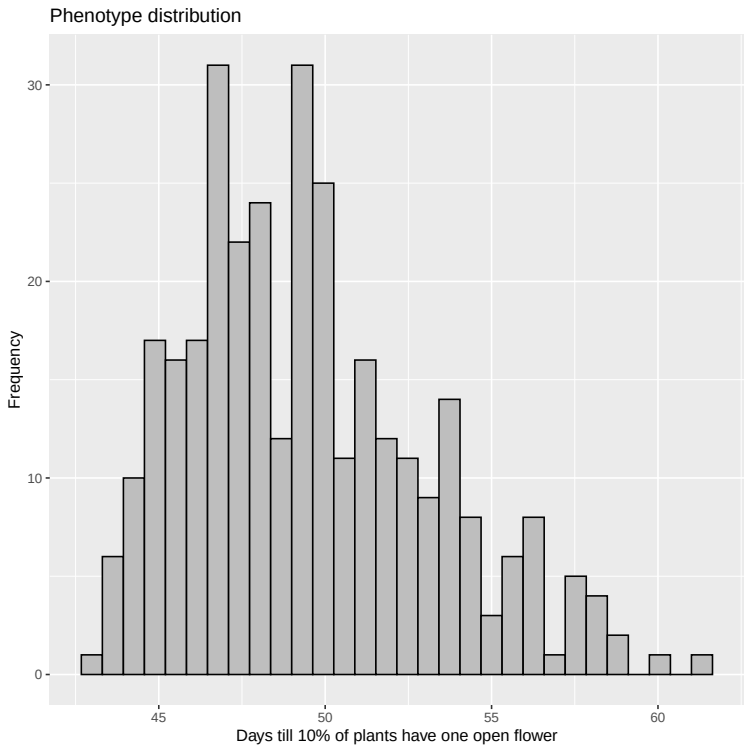
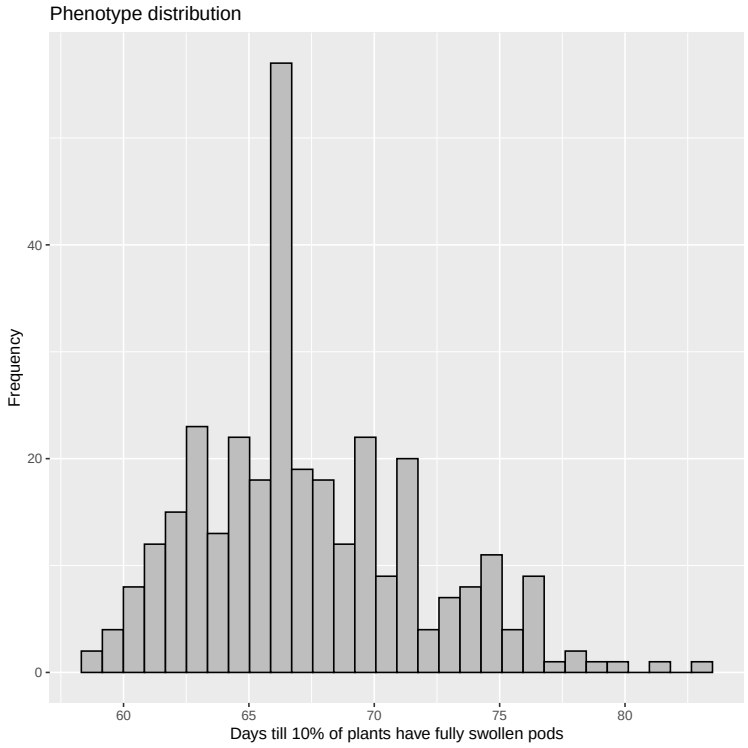


Figure 4.4: Histogram of distribution of quality scores for the *L. culinaris* dataset. Note that the highest reported quality score in this dataset is 999; all higher values have been capped at this value.

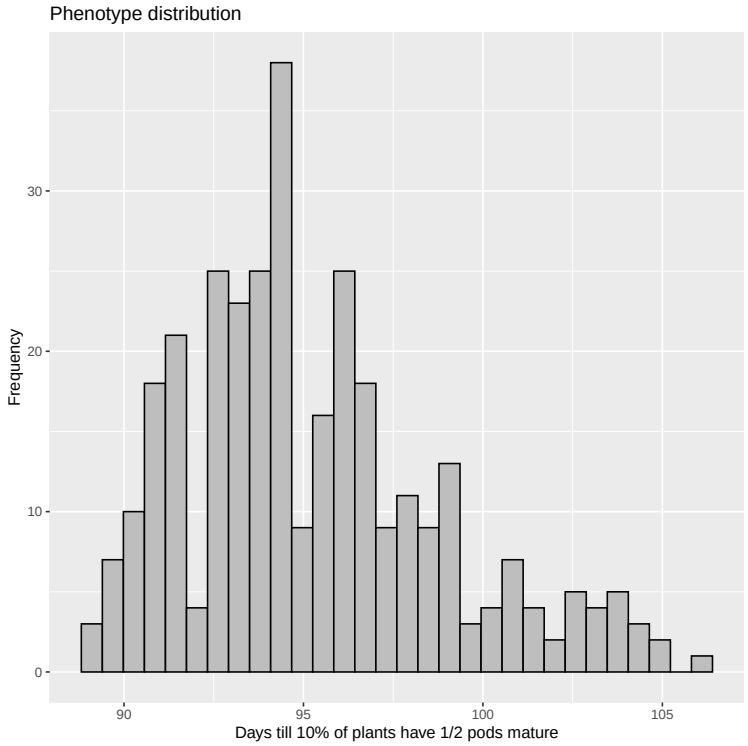


(a) Days to Flower



(b) Days to Swollen

Figure 4.5: Histogram of distribution phenotypes for the *L. culinaris* dataset.



(c) Days to Maturity

Figure 4.5: Histogram of distribution phenotypes for the *L. culinaris* dataset (cont.).

Chapter 5

Experiments

Multiple experiments were performed, each with the common theme of improving the ability to use deep learning for phenotype prediction. When taken together, the results form a toolbox of techniques that can be used to improve the use of available data for deep learning studies. All neural networks in these experiments were implemented using the `Keras` software package [38].

5.1 Genotype Representation for Input to Neural Network

The first experiment explored the best way to represent SNPs as input to a convolutional neural network. Determining an effective way to represent SNPs for use in deep learning studies provides a solid foundation from which future deep learning studies can build. Our hypothesis is that having a way to incorporate quality scores and reduce the amount of information discarded by Single Nucleotide Polymorphism (SNP) filtering can increase the power of a neural network in some cases. We look at two different input representations, Categorical Input Representation (CIR) and Probabilistic Input Representation (PIR), and compare the results of CNNs using each method on two different datasets, the *N. gonorrhoeae* dataset and the *L. culinaris* dataset.

Two posters were created exhibiting preliminary results for this investigation. A poster presented at Research Fest 2018 [39] showed the comparison of 2 neural networks on their ability to predict antimicrobial resistance of different *N. gonorrhoeae* strains. A follow-up poster was presented at the 2018 P2IRC symposium [40] that expanded upon these results and also incorporated *L. culinaris* data. Both of these posters, `kopas-poster-1.pdf` and `kopas-poster-2.pdf` can be found in Supplementary Materials.

5.1.1 Architecture

The CNN architecture was chosen based on experience in order to balance representative capacity with memory usage. Larger networks (networks with many layers and many convolutional filters) have more representative capacity, but require a lot of memory and time to train. Large networks can also lead to overfitting; if a network has too much representative capacity it will learn all the noise present in the dataset and memorize the inputs rather than learning useful connections that will generalize to other samples not seen in the training data. Small networks train faster and take less memory, but may not have the representative

capacity to learn the connections necessary to predict phenotypes.

The network architecture used was selected to minimize the risk of overfitting while still having enough representative capacity to learn to predict phenotypes. The more parameters a neural network has, the more closely it is able to estimate a non-linear function from the provided training data. This can lead to overfitting if the network learns to map entire inputs onto the provided outputs. When this happens, the network memorizes the input but does not know how to interpret samples that were not present in the training data. Alternatively, if the network does not have the capacity to completely memorize an input, it is then forced to determine which features are significant and can learn to disregard the features that may be noise. This allows the network to generalize to other samples because it is able to parse out the same significant features and perform a reasonable prediction despite not having been trained on the sample in question. The architecture selected was chosen to be large enough to be able to predict phenotypes for both datasets, but not so large as to be at risk of overfitting.

Early investigations showed that if the number of convolutional layers increased while the number of fully connected layers remained small, then network performance increased while the number of trainable parameters was reduced. It appeared that adding more convolutional layers increased a CNN's effectiveness more than adding fully connected layers.

All experiments used similar network architectures, but layer shapes had to be tailored to fit the different sized inputs of the 2 datasets (the *N. gonorrhoeae* and *L. culinaris* datasets have different number of SNPs). The number of neurons in the input layer of a CNN is always equal to the number of SNPs in an input sample. Convolutional layers following the input layer then reduce the number of neurons at each subsequent layer. While the architecture remains the same for each experiment (the choice and arrangement of layers does not change), the size of those layers must fit the input size.

The network architecture started with the input layer; next, there was a number of convolutional layers with varying numbers of filters. Every second convolutional layer was followed by a max-pooling layer and a dropout layer. The max pooling layer was added to reduce the size of the input to the next layer. All max pooling layers in this document had a size of 2×1, and therefore reduced the size of the input into the next layer by half. In order to reduce overfitting, dropout layers were added with a dropout rate of 25%. This architecture can be seen in Figures 5.1 and 5.2.

After the convolutional layers a flatten layer was added to prepare the data for the fully connected layers. Next, as stated, the data was passed to a number of fully connected layers. Fully connected layers do not retain spatial information; they connect each node in the input to each node in the output. This means that they had a very large number of trainable parameters. Each fully connected layer was followed by a dropout layer with a 50% dropout rate to reduce overfitting further.

Finally, data was fed to an output layer, which is just a fully connected layer with 1 output node. Each layer that required an activation function used the `ReLU` activation function [41] except the final output layer, which used a linear activation function for the regression tasks and a sigmoid activation function for the

classification task.

The complete architecture can be seen in Figure 5.1 for the *N. gonorrhoeae* dataset and in Figure 5.2 for the *L. culinaris* dataset. The only difference in the 2 architectures were the shapes of the layers; the types of layers and ordering of layers were the same in each network.

5.1.2 Methods

In order to assess how quality scores affect phenotype prediction we trained multiple CNNs. In the first type of experiment, quality scores were only used to filter out low-quality SNPs during data preprocessing. More details about the preprocessing of each dataset can be found in Subsections 4.2 and 4.3. The dataset was first filtered by a quality score threshold, then the remaining SNPs were categorized as homozygous alternate, heterozygous, or homozygous reference for the *L. culinaris* dataset and as reference, alternate 1, alternate 2, etc. for the *N. gonorrhoeae* dataset. Next, this categorization was one-hot encoded. The resulting vector was the input vector for each SNP. Any SNPs that were removed in the filtering step were represented by a vector of all zeros. This input representation will be called Categorical Input Representation (CIR) throughout this document and is described in Section 4.1 and Figure 4.1 in more detail. Next, a CNN was trained and evaluated using the phenotype values for each sample as the label the CNN learns to predict.

In the second type of experiment, SNP quality scores were provided to the model as additional information. In this experiment, SNPs were represented as a vector indicating the posterior probability of each possible genotype. For both datasets this posterior probability vector was a function of the SNP quality scores provided by the SNP calling software. For the *N. gonorrhoeae* dataset this quality score is defined in Subsection 2.3.3. FreeBayes reported quality scores as a PHRED quality score, and they were then converted into posterior probabilities. To obtain a posterior probability from a PHRED quality score, we had to solve Equation B.6 for P, which resulted in the following formula:

$$P = 10^{\frac{-Q}{10}}. \tag{5.1}$$

This allowed us to create a vector of posterior probabilities for each possible genotype. For the *L. culinaris* dataset this quality score was calculated by the Samtools `mpileup` program and was provided directly by the software, as described in Subsection 2.3.3. This input representation can be seen in Figure 5.3, and will be referred to as Probabilistic Input Representation (PIR).

Ten percent of each dataset was withheld for a validation set, then each type of neural network was trained for 200 epochs with an early stopping function. The early stopping function is set to stop training when the validation loss improves by less than 1 for 5 epochs. Both datasets contained continuous phenotype data, so the models were regression models rather than classification models. However, literature exists for thresholds to classify Antimicrobial Resistance (AMR) for *N. gonorrhoeae* as either susceptible or resistant [42], so a separate classification task was also performed for the *N. gonorrhoeae* dataset. For each experiment 3 trials were performed with randomized training/validation splits and the average across those trials was reported.

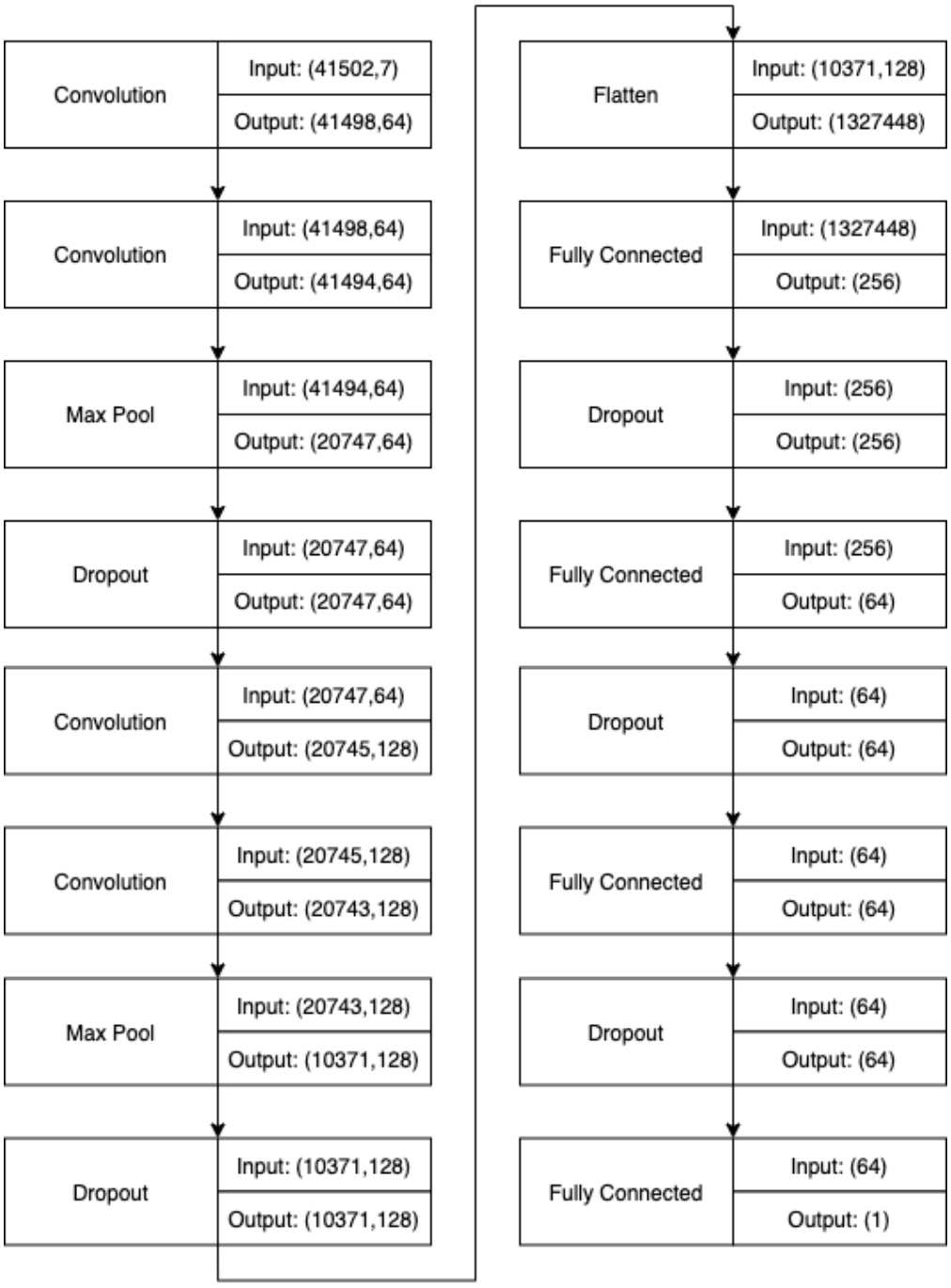


Figure 5.1: Neural network used for experiments using the *N. gonorrhoeae* dataset. The input and output dimensions for each layer are shown. The dimensions are provided in the same format as described in Chapter 2.1.3. The flatten layer accepts input in two dimensions and reshapes the layer into one dimension, as explained in Subsection 2.1.3.

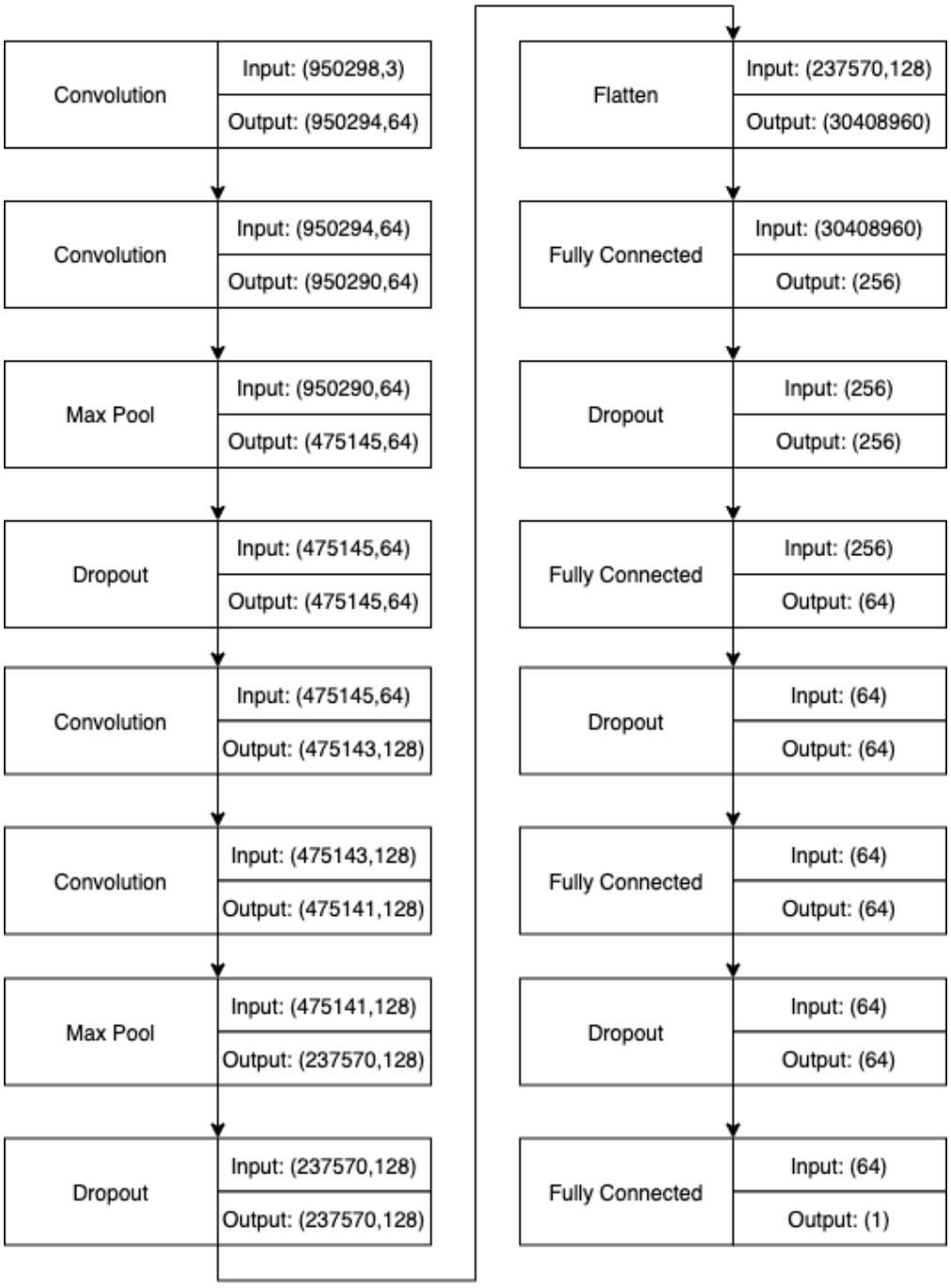


Figure 5.2: Neural network used for experiments using the *L. culinaris* dataset. The input and output dimensions for each layer are shown. The dimensions are provided in the same format as described in Chapter 2.1.3. The flatten layer accepts input in three dimensions and reshapes the layer into one dimension, as explained in Subsection 2.1.3.

$$
\begin{array}{c}
 & SNP_1 & SNP_2 & SNP_3 & & SNP_p \\
 & \begin{bmatrix} ref. \\ het. \\ alt. \end{bmatrix} & \begin{bmatrix} ref. \\ het. \\ alt. \end{bmatrix} & \begin{bmatrix} ref. \\ het. \\ alt. \end{bmatrix} & & \begin{bmatrix} ref. \\ het. \\ alt. \end{bmatrix} \\
sample_1 & \begin{bmatrix} 0.6 \\ 0.3 \\ 0.1 \end{bmatrix} & \begin{bmatrix} 0.92 \\ 0.06 \\ 0.02 \end{bmatrix} & \begin{bmatrix} 0.05 \\ 0.87 \\ 0.08 \end{bmatrix} & \dots & \begin{bmatrix} 0.01 \\ 0.01 \\ 0.98 \end{bmatrix} \\
sample_2 & \begin{bmatrix} 0.99 \\ 0.01 \\ 0 \end{bmatrix} & \begin{bmatrix} 0.91 \\ 0.05 \\ 0.04 \end{bmatrix} & \begin{bmatrix} 0.01 \\ 0.1 \\ 0.89 \end{bmatrix} & \dots & \begin{bmatrix} 0.12 \\ 0.79 \\ 0.09 \end{bmatrix} \\
 & & & \vdots & & \\
sample_n & \begin{bmatrix} 0.02 \\ 0.05 \\ 0.93 \end{bmatrix} & \begin{bmatrix} 0.98 \\ 0.02 \\ 0 \end{bmatrix} & \begin{bmatrix} 0.18 \\ 0.6 \\ 0.22 \end{bmatrix} & \dots & \begin{bmatrix} 0.04 \\ 0.9 \\ 0.06 \end{bmatrix}
\end{array}
$$

Figure 5.3: Input matrix for probablistic representation. Each SNP is represented by a vector of the posterior probabilities of each possible genotype. The abbreviations $ref.$, $het.$, and $alt.$ correspond to the homozygous reference, heterozygous, and homozygous alternate genotypes correspondingly. Notice that no SNPs have been filtered out.

Many machine learning experiments split datasets into 3 parts: the training, testing, and validation sets. This is done because hyperparameters are often tuned to perform well on the test set, so a validation set is needed to ensure that the model works on a 3rd, unseen dataset as well. The datasets available to us have very few samples, and removing samples from the training data for a 3rd set would make training the network much more difficult. The choice to only use a training and validation set is justified for two reasons. First, the hyperparameters for each experiment are not tuned to a particular validation set in these experiments. The network architecture and hyperparameters were set after some initial exploration and were unchanged for each experiment and each trial. The only hyperparameter that changed from experiment to experiment was the early stopping function, and this was only used for the *N. gonorrhoeae* experiments. Secondly, repeated trials were performed with random dataset splits. This attempts to mitigate any bias towards a particular validation set, as the validation set changed between different trials.

The loss function that we used to measure performance of the regression models was the Root Mean Squared Error (RMSE) of the predicted values. The RMSE is a measure of ability of a model to predict continuous values close to the true phenotype value; however, RMSE is relative to the magnitude of the phenotype values, and as such RMSE values cannot be compared for the prediction of different phenotypes. As a baseline, results from the current state-of-the-art, RR-BLUP, are provided in Appendix D.1. RMSE was compared for both networks predicting each phenotype, the one using CIR and the other using PIR. An Anderson-Darling test and a Kolmogorov-Smirnov test were additionally performed for the *N. gonorrhoeae* dataset to determine if the distribution of predicted MICs matched the distribution of the true MICs. The Anderson-Darling k-sample test was used, which tests the null hypothesis that k-samples are drawn from the

same distribution without having to specify the characteristics of that population. The null hypothesis in this case was that the predicted MIC values came from the same distribution as the true MIC values. The Kolmogorov-Smirnov 2-sample test has the same null hypothesis as the Anderson-Darling test used: the null hypothesis is that 2 independent samples were drawn from the same continuous distribution. Rejecting the null hypothesis in either case indicates that the predicted values do not share the same distribution function as the true values the neural network was trained on, and is evidence that the network is not providing accurate predictions. Accepting the null hypothesis does not mean that the network is providing accurate predictions, but it is evidence that the network is predicting reasonable values for both the training and validation datasets.

For the classification task the accuracy of the model was evaluated. Each model attempted to classify each *N. gonorrhoeae* strain as either susceptible or resistant for a single antibiotic. The accuracy of each model was defined as the total number of correct classifications divided by the total number of classifications performed. This was done for 5 different phenotypes for the *N. gonorrhoeae* dataset.

5.1.3 Results and Discussion

The results and discussion are presented together for each dataset. The results of each model for the *N. gonorrhoeae* are presented first, starting with the regression task and then for the classification task. After, the results are presented for the *L. culinaris* dataset. Finally, future work is discussed at the end of this section.

Neisseria gonorrhoeae

The *N. gonorrhoeae* data for this study is described in Section 4.2. In these datasets there were a total of 676 different strains and 41,502 SNPs. Each strain had 5 associated phenotypes with the exception of a few cases where data were missing. The phenotypes were the MICs for each of the 5 drugs: cefixime, azythromycin, ciprofloxacin, penicillin, and tetracycline. *N. gonorrhoeae* NGS data was used as a pilot study because of its small size and simplicity. *N. gonorrhoeae* is a haploid organism, which means heterozygosity does not need to be accounted for. The phenotypes, MIC values, were a good starting point because they could be used for both regression and classification studies. The quantitative MIC values were used to create a regression model where the model attempted to predict the actual MIC value, and the categorical resistance values were used to create a classification model where the model attempted to predict whether each strain was resistant or susceptible to the antibiotic.

The results for the regression task can be seen in Table 5.1, and in Figure 5.4. These results show that the PIR model outperformed the CIR model for prediction of cefixime, ciprofloxacin, and tetracycline MIC prediction. For penicillin MIC prediction the CIR model performed best and for azithromycin the PIR model performed better for the training set but much worse on the validation set. For azithromycin this is likely due to overfitting; there was a very large variance between the different trials.

Table 5.2 shows the average number of epochs required for training each neural network for the regression task. Number of epochs is reported because it is a hardware-independent method of measuring how long training takes. In all cases except the tetracycline prediction model, the PIR model took less time to train. For azithromycin prediction the PIR model took significantly less time to train than the CIR model: 45 epochs for the former and 147 epochs for the latter. This likely means that the PIR model was not fully trained, which may explain why the validation loss was much worse for the PIR model than for the CIR model for azithromycin MIC prediction. Tuning the early stopping function may prevent training from ending before the network is fully trained; however, with small datasets it is difficult to achieve smooth training and any early stopping function is likely to get stuck in a local minima.

A Kolmogorov-Smirnov test and an Anderson-Darling test were performed as additional analyses, which can be seen in Tables 5.3 and 5.4. These were performed to test whether the predicted MIC values were from the same distribution as the true MIC values. Under these tests the null hypothesis was that the predicted values and the true values came from the same distribution. In this case, accepting the null hypothesis would be evidence that the predictions were accurate. In all cases except one the null hypothesis can be rejected at a $p = 0.01$ significance level, meaning that the distribution of predicted phenotypes does not match the distribution of true phenotypes in most cases. The only case where the null hypothesis could not be rejected was the case of tetracycline MIC prediction using the CIR model. This was evidence that the CIR model performed the best at its prediction task in this particular case. This also meant that in most cases the predicted phenotypes did not follow a distribution similar to that of the true phenotype distribution. This was not surprising due to the small size of the dataset used. Being able to train on a larger dataset would likely yield a better distribution of predicted phenotypes.

Phenotype	CIR Training Loss	CIR Validation Loss	PIR Training Loss	PIR Validation Loss
Azithromycin	1918.56	204.90	1777.99	2085.84
Cefixime	3.79	3.77	3.77	3.66
Ciprofloxacin	143.87	164.86	93.05	92.76
Penicillin	306.93	219.10	132.47	154.96
Tetracycline	160.35	408.71	371.65	542.50

Table 5.1: Results for the CIR and the PIR models on the prediction of MIC values for each of the 5 AMR phenotypes. This task was a regression task and the reported values are the RMSE of the predicted MIC values averaged across all trials. Smaller RMSE indicates better performance.

Phenotype	Number of Epochs for CIR Model	Number of Epochs for PIR Model
Azithromycin	147	45
Cefixime	73	70
Ciprofloxacin	113	95
Penicillin	123	98
Tetracycline	74	81

Table 5.2: Average number of epochs required for training to plateau for the regression task. Training continued until loss value improvement plateaued or until 200 epochs occured.

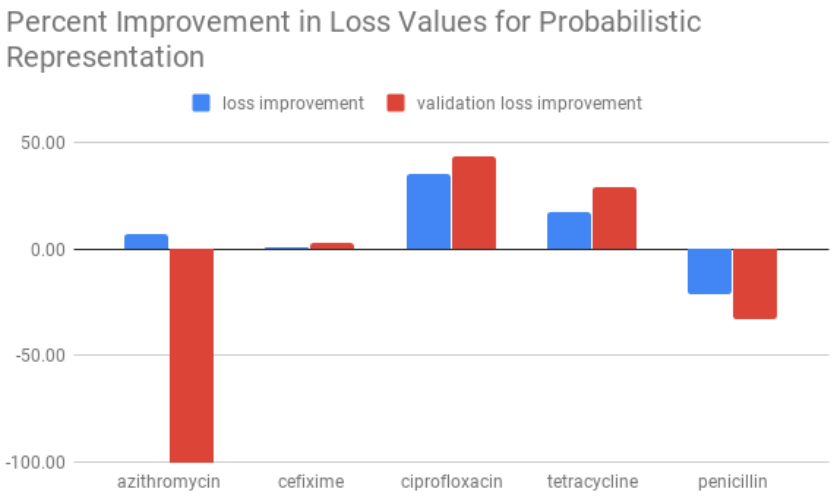


Figure 5.4: Improvement of RMSE for MIC prediction for *N. gonorrhoeae*. Values are the difference in RMSE between the CIR model and the PIR model as a percentage of the loss of the PIR model. The blue bars are the RMSE differences observed on the training data and the red bars are the RMSE differences observed on the test data. Positive values mean that the PIR model outperformed the CIR model. Note that azithromycin loss was over 900% larger for the PIR model and was truncated for this figure.

Phenotype	KS Test Statistic for CIR Model	P-value for KS for CIR Model	KS Test Statistic for PIR Model	P-value for KS for PIR Model
Azithromycin	0.47	2.79E-07	0.32	1.15E-03
Cefixime	0.76	4.19E-18	0.70	1.93E-15
Ciprofloxacin	0.50	3.65E-08	0.50	3.65E-08
Penicillin	0.52	9.43E-09	0.78	8.35E-19
Tetracycline	0.43	4.69E-06	0.25	0.023

Table 5.3: KS Test statistic for both PIR model and CIR model. The only case where we can say with significance that the predicted MIC values were sampled from the same distribution as the measured MIC values was when predicting Tetracycline resistance with the PIR model.

Phenotype	AD Test Statistic for CIR Model	P-value for AD for CIR Model	AD Test Statistic for PIR Model	P-value for AD for PIR Model
Azithromycin	9.51	1.72E-04	5.14	3.21E-03
Cefixime	34.1	8.62E-04	25.5	1.47E-05
Ciprofloxacin	18.3	8.91E-06	15.1	1.68E-05
Penicillin	15.2	1.62E-05	36.8	6.33E-03
Tetracycline	7.90	4.55E-04	3.52	0.012

Table 5.4: AD Test statistic for both PIR model and CIR model. The only case where we can say with significance that the predicted MIC values were sampled from the same distribution as the measured MIC values was when predicting Tetracycline resistance with the PIR.

Phenotype	Number of Epochs for CIR Model	Number of Epochs for PIR Model
Azithromycin	45	44
Cefixime	45	44
Ciprofloxacin	44	45
Penicillin	44	44
Tetracycline	45	44

Table 5.6: Average number of epochs required for training to plateau for the classification task. Training continued until loss value improvement plateaued or until 200 epochs occured.

For the classification task the percent improvement in accuracy of the PIR model is shown in Figure 5.5. The average results across each trial are shown in Table 5.5, and the training time results are shown in

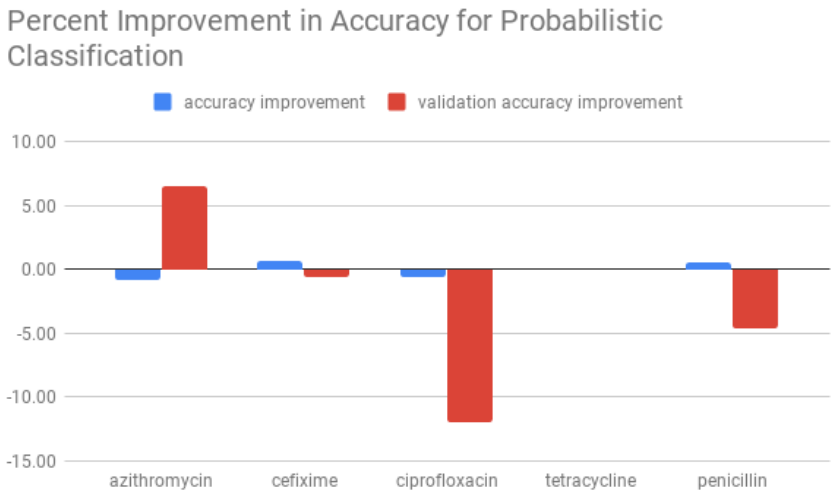
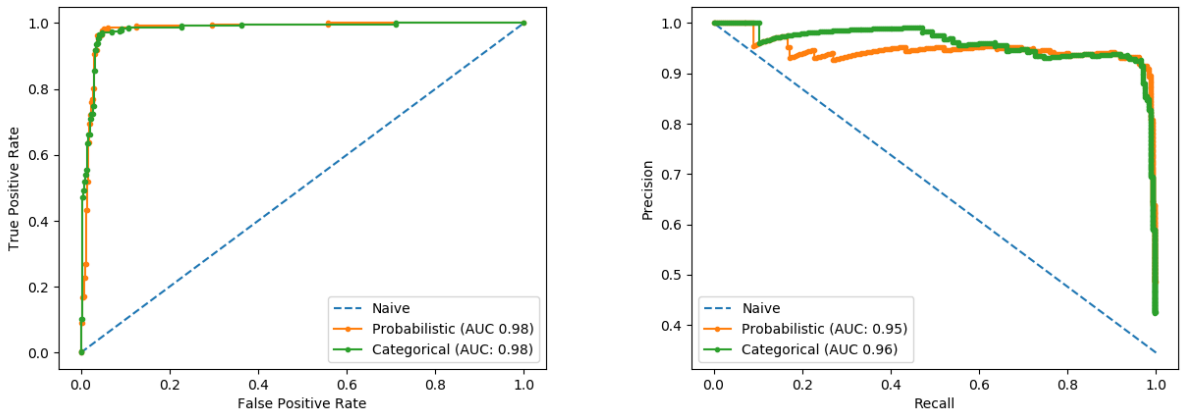


Figure 5.5: Improvement of accuracy for AMR classification for *N. gonorrhoeae*. Values are the difference in accuracy between the CIR model and the PIR model as a percentage of the accuracy of the PIR model. The blue bars are the RMSE differences observed on the training data and the red bars are the RMSE differences observed on the test data. Positive values mean that the PIR model outperformed the CIR model.

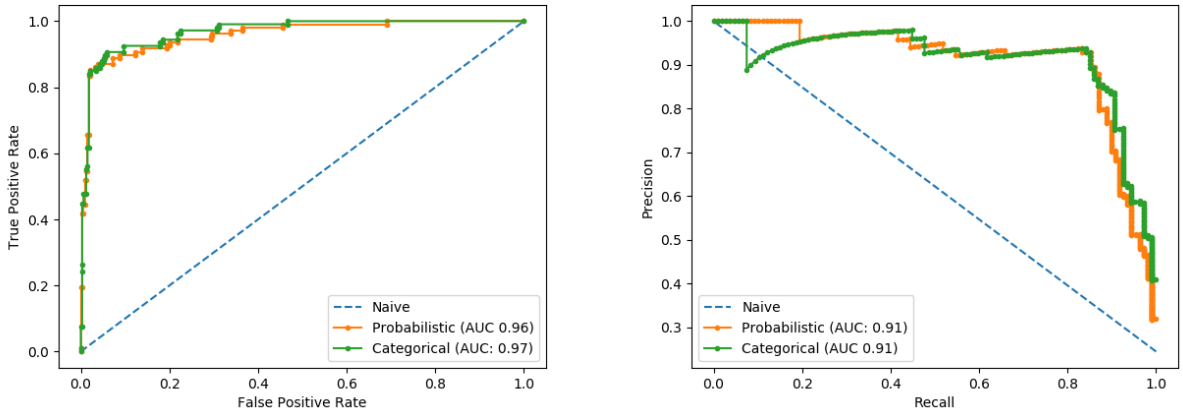
Table 5.6. Accuracy on the validation set is a better measure of how a classification performs, so analysis will focus on the validation accuracy values. The results show that for tetracycline the PIR model and the CIR model both performed equally. Azithromycin is the only case where the PIR model performed better on the validation set. Note that this is opposite to the results observed for the regression task. For ciprofloxacin and penicillin the CIR model performed better on the validation set. Figures 5.6 to 5.10 show the Receiver Operating Characteristics (ROC) and precision-recall curves for the classification task.

Overall, it appears that the CIR model performs better than the PIR model for the classification task. For the regression task it doesn't appear that one model outperformed the other; each representation performed better for the MIC prediction of two antibiotics, and for cefixime prediction the performance difference was negligible. One possible explanation why the PIR model performed better on the regression task than the classification task is because the PIR model retains more information than the CIR model. This extra information could have been useful for regression tasks, where the magnitude of the prediction is important, whereas for a classification task this extra information may be extraneous and causes training to be more difficult. Another possible explanation is that low quality SNPs are actually misleading the PIR model, and there's not enough data for the CNN to learn to discount low quality SNPS. If this is the case, the CIR model would not be affected in the same way as the low quality SNPs are removed. The PIR model retains extra information because of the way it handles quality scores. In the CIR model not only are SNPs discarded if their quality score does not meed a certain threshold, but the quality score information is also discarded. By encoding the quality score information alongside SNPs via the posterior probability vector, the PIR model contains much more information that a CNN can use to improve its internal model. However, the PIR model



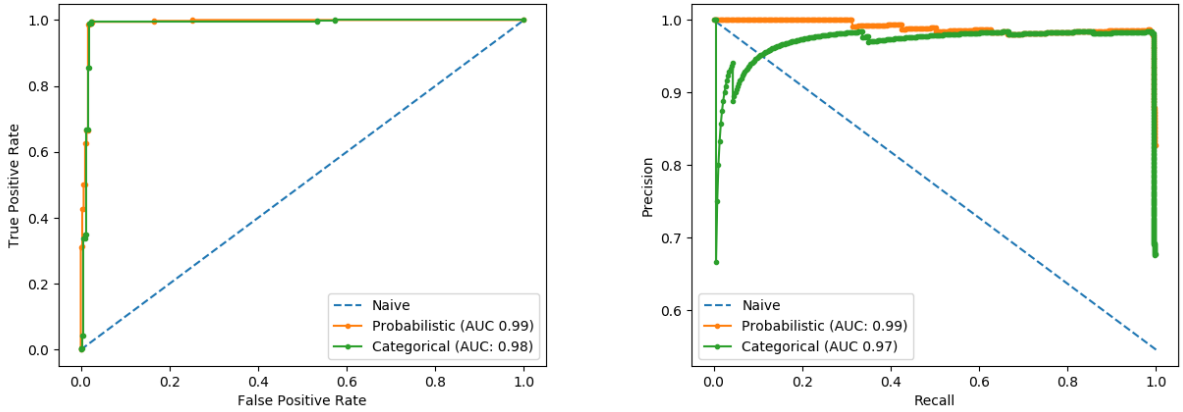
(a) ROC Curve **(b)** Precision-Recall Curve

Figure 5.6: ROC and precision-recall curves for azithromycin classification for both the PIR and CIR models and a naive model for comparison. The naive model represents a model that cannot discriminate between susceptible and resistant and guesses either classification with a 50% chance. The precision recall curve shows a larger area under the curve (AUC) for the CIR model, indicating a better performing model.



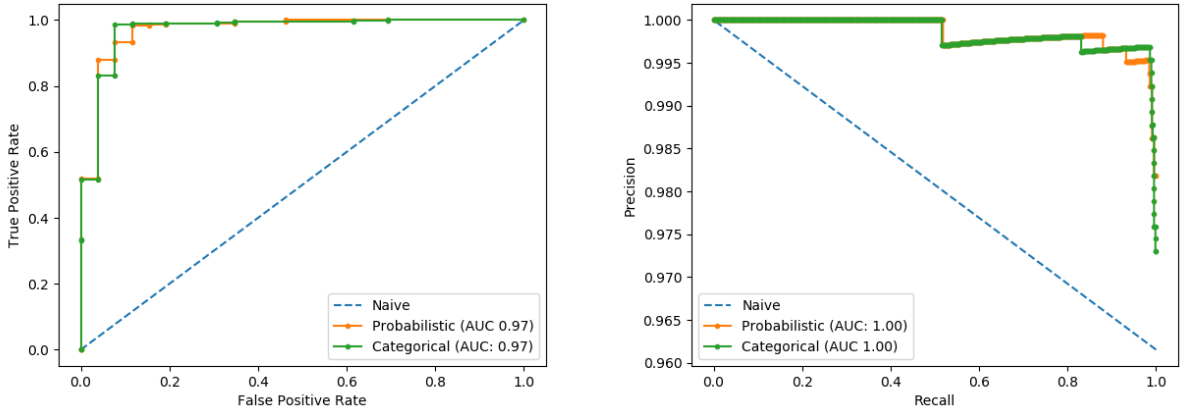
(a) ROC Curve **(b)** Precision-Recall Curve

Figure 5.7: ROC and precision-recall curves for cefixime classification for both the PIR and CIR models and a naive model for comparison. The naive model represents a model that cannot discriminate between susceptible and resistant and guesses either classification with a 50% chance. The ROC curve shows that the CIR model slightly outperforms the PIR model. This can be be seen by a slightly larger area under the curve (AUC). However, the precision recall curve shows that the CIR model underperforms when the recall is low, indicating the model may perform poorly in some circumstances.



(a) ROC Curve **(b)** Precision-Recall Curve

Figure 5.8: ROC and precision-recall curves for ciprofloxacin classification for both the PIR and CIR models and a naive model for comparison. The naive model represents a model that cannot discriminate between susceptible and resistant and guesses either classification with a 50% chance. The PIR model has a higher area under the curve (AUC) for both the ROC curve and the precision-recall curve. Additionally, the precision-recall curve for the CIR model shows that it performs worse than the naive model in some circumstances.



(a) ROC Curve **(b)** Precision-Recall Curve

Figure 5.9: ROC and precision-recall curves for tetracycline classification for both the PIR and CIR models and a naive model for comparison. The naive model represents a model that cannot discriminate between susceptible and resistant and guesses either classification with a 50% chance. Both models have nearly identical curves.

Phenotype	CIR Training Accuracy	CIR Validation Accuracy	PIR Training Accuracy	PIR Validation Accuracy
Azithromycin	0.80	0.82	0.79	0.87
Cefixime	0.84	0.86	0.84	0.85
Ciprofloxacin	0.94	0.95	0.93	0.83
Penicillin	0.92	0.95	0.92	0.90
Tetracycline	0.96	0.96	0.96	0.96

Table 5.5: Results for the CIR and the PIR models on the prediction of AMR for each of the 5 AMR phenotypes. This task was a classification task and the reported values are the accuracy of the model's prediction as either resistant or susceptible averaged across all trials. Smaller RMSE indicates better performance.

only performed better in some cases; therefore, we cannot say that encoding quality information is beneficial to all CNNs. Training CNNs on the datasets available with so few samples is unstable and results can vary across different trials. Performing these experiments with more replicates and a larger datasets would yield more conclusive results.

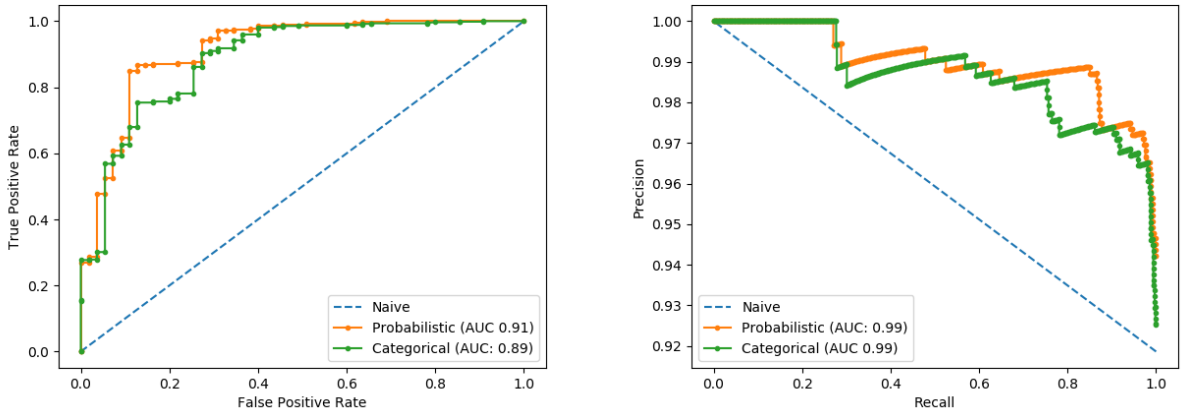
In conclusion, both input representations performed similarly. Each representation model performed better in some cases and worse in other cases. For classification tasks, the CIR model appears to be superior in most cases.

Lens culinaris

Phenotype	CIR Training Accuracy	CIR Validation Accuracy	PIR Training Accuracy	PIR Validation Accuracy
DTF	386.30	670.26	472.16	1153.86
DTS	539.03	1334.38	516.67	475.03
DTM	876.34	1451.58	925.96	1423.19

Table 5.7: Results for the CIR and the PIR models on the prediction of Phenotypes for *L. culinaris* The reported values are the RMSE of the predicted phenotype values after training for 200 epochs and averaged across 3 trials. Smaller RMSE indicates better performance.

L. culinaris data was also used to train and evaluate a CNN in its ability to predict phenotype values from genotype data. For this dataset only a regression task was performed. Any attempt to turn quantitative phenotype data such as DTF or DTS into a categorical variable such as ["late", "regular", "early"] would be subjective; therefore, a classification task was not performed. The dataset only contained 324 genomes, so it was difficult to train the models. Early stopping was disabled and each model was trained for 200 epochs. During training the loss values were not a smooth, monotonically decreasing function, and if an



(a) ROC Curve **(b)** Precision-Recall Curve

Figure 5.10: ROC and precision-recall curves for penicillin classification for both the PIR and CIR models and a naive model for comparison. The naive model represents a model that cannot discriminate between susceptible and resistant and guesses either classification with a 50% chance. The PIR model has a higher area under the ROC curve, indicating better performance.

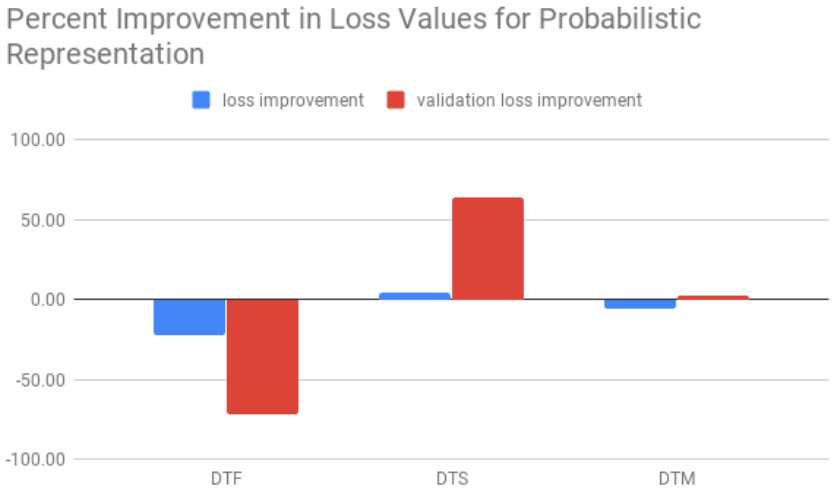


Figure 5.11: Improvement of RMSE for phenotype prediction for *L. culinaris*. Values are the difference in RMSE between the CIR model and the PIR model as a percentage of the loss of the PIR model. The blue bars are the RMSE differences observed on the training data and the red bars are the RMSE differences observed on the test data. Positive values mean that the PIR model outperformed the CIR model.

early stopping function was used it would have prematurely stopped training, resulting in a partially trained model. Results can be seen in Table 5.7.

Figure 5.11 shows that there was not much difference between the different input representation models. In the case of DTF the CIR model performed significantly better. In the case of DTS the PIR model performed significantly better. Finally, in the case of DTM both models performed equally well. This is similar to the results seen with the *N. gonorrhoeae* dataset, as both input representations performed better in some cases, and no definitive conclusion could be drawn.

5.1.4 Future Work

There are multiple different avenues that can be explored to continue this work. Firstly, the neural network hyperparameters and architecture could be further tuned; this may result in more accurate phenotype prediction for one or both input representation models. Secondly, performing these experiments on larger datasets with more samples would remove the instability in the results and allow one to draw stronger conclusions about which input representation model is superior. A dataset with ten thousand samples would allow us to train more effective models and would possibly allow us to draw statistically significant conclusions. Oftentimes, gathering more samples leads to more environmental variability, so a third avenue of exploration would be to determine if incorporating environmental data into the input of a machine learning model could mitigate this problem. Fourthly, it would be interesting to see if the PIR would yield more dramatic improvements if it was applied to a dataset with lower quality scores. Finally, putting both models through SHAP [10] or another model explainer would allow us to dig deeper into each network and investigate which SNPs each model found significant. If the PIR model found significance in SNPs that were filtered out from the CIR model then this would give some insight into why the former model performs better in some cases, and could possibly identify causative SNPs that may warrant biological investigation.

5.2 Transfer Learning

A consistent problem that makes it difficult to train neural networks is that the datasets available do not have enough samples to train an effective predictive model. In order for deep learning to be successful, there must be a way to make do with the datasets that available. One technique that can artificially increase the size of a dataset is transfer learning. Transfer learning allows one to make use of every sample in a dataset multiple times, by training a prediction model to predict one phenotype, and then retraining the model to predict another phenotype. In this section we present an experiment that shows that transfer learning can be used to increase the effectiveness of CNNs for phenotype prediction.

The mechanism that causes transfer learning to be successful is the transfer of knowledge (or learned connections within a neural network) from one task to another. In this case the tasks are the prediction of different phenotypes. The features and connections learned when predicting the first phenotype are retained

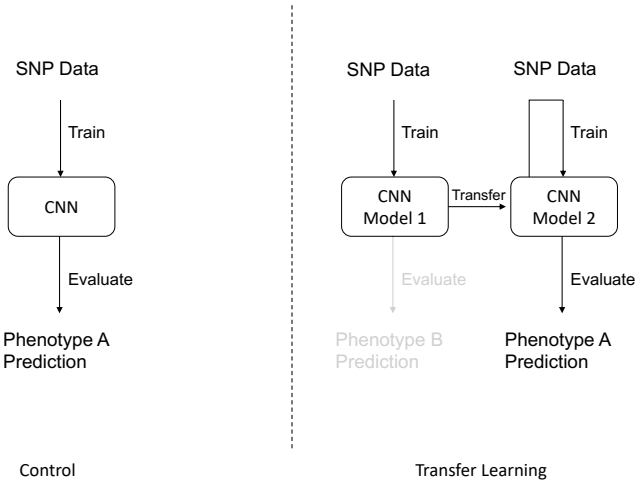


Figure 5.12: This figure shows the generalized methodology for transfer learning. For the control (left), a CNN is trained to predict phenotype A. For the transfer learning case (right), model 1 is trained to predict phenotype B. Next, the learned weights from model 1 are transferred to model 2, which is re-trained to predict Phenotype A. The transfer and control models are then compared on their ability to predict phenotype A.

and leveraged for prediction of the second phenotype. Leveraging features and connections across phenotypes is useful if the phenotypes are biologically related or involve similar genetic mechanisms. For example, if the mechanisms underlying the phenotypes both use a similar protein then all phenotype prediction models that predict phenotypes that make use of that protein can reuse the part of the neural network that has learned to detect that protein. Another example would be a convolutional layer that learns to detect a gene variant and upstream promoter region. If a neural network, specifically a CNN, is able to learn to detect these gene-promoter region configurations, then transferring these learned configurations to a second neural network may allow the second neural network to make use of these configurations. The second network may not even use the same gene and promoter region, but can leverage the learned network connections to find other gene and promoter regions that have a similar chromosomal layout to the first.

5.2.1 Architecture

The architecture of the neural networks used for both *N. gonorrhoeae* and for *L. culinaris* are the same as in the previous experiment, for both the classification and regression tasks. This explanation can be found in Subsection 5.1.1.

5.2.2 Methods

This experiment was performed using both *N. gonorrhoeae* and *L. culinaris* datasets. The methods for each are slightly different; however, Figure 5.12 depicts a general version of the methods.

Neisseria gonorrhoeae

Transfer learning for *N. gonorrhoeae* assumes that different antimicrobial resistance mechanisms share some common genetic mechanisms. If that assumption is correct, then neural networks can exploit these common mechanisms via transfer learning.

The dataset used, which is described in Section 4.2, contains AMR information for 5 different antibiotics: azithromycin, cefixime, ciprofloxacin, penicillin, and tetracycline. For each of the antibiotics listed above a neural network, called model 1, is trained to predict the MIC values of that antibiotic. Model 1 is trained until the loss value plateaus (this is determined by the early stopping function) or 200 epochs have ocurred, whichever comes first. The weights from model 1 are then used to initiate the training of a second neural network, called the model 2, for AMR prediction of a different antibiotic. Model 2 is created with the neural network's initial layer weights set equal to the final network layer weights of model 1. Next, the model 2 network is configured so that the convolutional layers are never updated during the backpropagation step. This means that only the fully connected layers are updated during backpropagation, forcing model 2 to make use of the features that were learned in the convolutional layers from model 1. Model 2 is trained until it plateaus (via early stopping) and evaluated on the validation dataset. All pairwise combinations of antibiotics was explored resulting in 20 different control-transfer pairs.

Lens culinaris

The phenotypes available for *L. culinaris* are all related in the fact that they are all indications of the plant's growth rate in different stages of development. We assume that these phenotypes also share some common genetic mechanism, and since all measurements are from the same location and growing season we can rule out environmental conditions as a factor.

The transfer learning study for *L. culinaris* is very similar to the transfer learning study for *N. gonorrhoeae*. We start with model 1, which is trained until plateau to predict one of the phenotypes: DTF, DTS, DTM. Each of these phenotypes is described in Section 4.3. That model is then used to initialize a predictive model (model 2) for either of the other two remaining phenotypes. Every permutation was explored resulting in 6 different control-transfer pairs.

5.2.3 Results and Discussion

N. gonorrhoeae

The results for transfer learning on the *N. gonorrhoeae* dataset are provided in Table 5.8. In all cases except for the case of azithromycin MIC prediction, phenotype prediction results on the validation set were improved when transfer learning was utilized over the control, as shown in Figure 5.13. Figure 5.15 show the loss values on the validation data for each model pair. For the penicillin MIC prediction task, the loss on the validation set improved significantly; however, the model 1 network showed signs of overfitting, which is indicated by

	Azithromycin		Cefixime		Ciprofloxacin		Penicillin		Tetracycline	
	Train. Loss	Val. Loss	Train. Loss	Val. Loss	Train. Loss	Val. Loss	Train. Loss	Val. Loss	Train. Loss	Val. Loss
Azithromycin	1918.6	204.9	1.7	1.6	98.2	118.7	275.8	60.6	131.4	177.8
Cefixime	631.7	1011.4	3.8	3.8	77.0	63.2	338.3	14.4	168.1	220.2
Ciprofloxacin	543.3	1065.6	3.4	3.3	143.9	164.9	268.9	36.7	69.6	115.4
Penicillin	1618.3	3278.0	17.8	17.8	138.3	128.4	160.3	219.1	180.1	256.2
Tetracycline	996.4	1370.5	7.1	7.1	98.6	103.1	142.3	71.9	306.9	408.7

Table 5.8: Transfer learning results on the *N. gonorrhoeae* data. The dark cells are the control results. Reported values are the root mean squared error (RMSE) for the Loss and Val (validation) Loss subcolumns. Each column shows the results for prediction of the resistance of a single antibiotic. Each row shows the results when the prediction model for the prediction of AMR of the corresponding antibiotic is used as the initial network weights (model 1).

Epochs required for training	DTF	DTS	DTM
DTF	147	44	43
DTS	195	73	41
DTM	61	56	113

Table 5.9: Time required to train CNNs for prediction tasks on the *L. culinaris* data. The dark cells are the control results. Reported values are the number of epochs required for training to completion. Each column shows the number of training epochs for prediction of the resistance of a single antibiotic. Each row shows the number of training epochs when the prediction model for the prediction of AMR of the corresponding antibiotic is used as the initial network weights (model 1).

high loss values on the validation set. In most cases training time was also reduced, which can be seen in Table 5.9, and the overall average training time was reduced compared to the control, as shown in Figure 5.14. These results indicate that, at the very least, there are some common genetic features for antibiotic resistance that networks can use for each prediction task.

It is important to emphasize that the evaluation of this experiment compares the results of the transfer learning models to the control models. This relative comparison is fragile as either the control, the transfer model, or both may be prone to overfitting, incomplete training, or other issues. When looking at these results it is important to look at both the training and validation results for both the transfer models and the control models as a whole. Any cases where the training and validation results are not close to each other is a red flag, and these cases may not be indicative of the actual results. In order to mitigate the effects of this, repeated trials should be performed; however, the computing resources required for this grows rapidly.

Some pairs of prediction models performed well when transfer learning was used between the two, for

example cefixime and azithromycin appeared to be complementary to each other. When cefixime is used as model 1 for azithromycin prediction and when azithromycin is used as model 1 for cefixime prediction, the results were better than when other models were used for model 1. This is strong evidence that cefixime resistance and azithromycin resistance share a similar biological mechanism. A possible candidate for this biological mechanism is shared below. Ciprofloxacin and penicillin both showed the best improvement when cefixime was used for model 1 as well. Penicillin showed slight improvements when tetracycline was used as model 1, but tetracycline performed worse when penicillin was used as model 1.

Antimicrobial	Drug Class
Azithromycin	Macrolide
Cefixime	Cephalosporin
Ciprofloxacin	Fluoroquinolones
Penicillin	Penicillin
Tetracycline	Tetracycline

Table 5.10: Information was taken from MedlinePlus [43].

All of these antibiotics are different classes of drugs, as shown in Table 5.10. Despite being different classes of drugs, some of the antibiotic resistance mechanisms may share genetic features with other antibiotic resistance mechanisms. For example, transfer learning between azithromycin and cefixime showed significant improvement. A possible explanation for this is the mtrCDE multidrug efflux pump. The mtrCDE multidrug efflux pump is known to confer azithromycin resistance in *N. gonorrhoeae*, and the mtrR gene regulates the expression of mtrCDE [44]. Shi *et al.* [9] found a significant association between mutations in both the mtrR gene and mtrR promoter regions in both cefixime and azithromycin resistance. While this is not conclusive evidence, it offers an explanation as to the relationship between phenotypes that transfer learning is able to exploit.

An alternative explanation for the success of transfer learning is that the effective dataset size is increased when using transfer learning. These datasets have few samples, which makes training difficult and overfitting easy, and transfer learning allows us to use each sample twice for training. This is one possible contributor to the success of transfer learning as a tool for phenotype prediction; however, the *L. culinaris* results in Section 5.2.3 provides evidence that this is not the only mechanism at work.

Finally, training times improve when using transfer learning. The number of epochs is reported as a measure of training time as this is a hardware independent measurement. A network that takes 20 epochs to train will take 20 epochs to train no matter how many CPU cores or how much memory the machine it is trained on has. However, machines with more RAM and CPU cores will take less time per epoch than a less powerful machine would. The number of epochs is reduced when training a neural network with transfer learning due to the early stopping mechanism. This is likely because when training the control, the layers are initialized with a random function, and when transfer learning is used the neural network layers start off with

already trained values. If the layers are initialized with meaningful values, part of the training work is already done for the neural network. The fact that neural network training takes less time with transfer learning is not entirely practical, since model 1 needs to first be trained in order to even begin with transfer learning, leading to a longer total time for using transfer learning. If a model 1 candidate is already available, however, the initial training time does not need to be considered, and in this use case transfer learning requires less time than training an uninitialized model.

L. culinaris

	DTF		DTS		DTM	
	Train. Loss	Val. Loss	Train. Loss	Val. Loss	Train. Loss	Val. Loss
DTF	386.3	670.3	359.9	469.6	663.1	1343.7
DTS	221.2	947.2	539.0	1334.4	688.1	3504.1
DTM	216.0	249.1	333.2	336.8	876.3	1451.6

Table 5.11: Transfer learning results on the *L. culinaris* data. The dark cells are the control results. Reported values are the root mean squared error (RMSE) for the Loss and Val (validation) Loss subcolumns. Each column shows the results for the prediction of a single phenotype: days to flower (DTF), days to swollen (DTS), and days to maturity (DTM). Each row shows the results when the prediction model for the prediction of the corresponding phenotype is used as the initial network weights (model 1).

Transfer learning was very successful for the *L. culinaris* data. In the best case we were able to reduce the loss on the validation set by 74.8%, as shown in Figure 5.16. Results can be seen in Table 5.11. Figure 5.17 show the loss values on the validation data for each model pair.

The *L. culinaris* dataset only contains 324 samples, which is a very small dataset on which to perform deep learning. As such, it is difficult to prevent overfitting, which is often indicated by much higher loss values on the validation set than the training set. As a result we see evidence of overfitting on the control, where most validation loss values are much higher than the training loss values. When transfer learning is used, however, predictive models show fewer signs of overfitting. One interesting case is the case of predicting DTS. In this case the control appears to overfit the training set, with a validation loss almost 3 times as high as the training loss. With transfer learning, however, the validation loss is almost as low as the training loss. This is strong evidence to suggest that transfer learning can cause small datasets to be useful for deep learning and somewhat mitigate the curse of dimensionality and reduce overfitting.

As mentioned in the previous experiment, it is important to emphasize that the evaluation of this experiment compares the results of the transfer learning models to the control models. Relative comparisons can be fragile as either the control, the transfer model, or both may be prone to overfitting, incomplete training, or other issues. These results must be considered holistically, and the possibility of inaccurate baselines must

be factored in when drawing conclusions.

Transfer learning was most effective when a relatively more complex phenotype, DTM, is used as model 1 and model 2 is used to predict a related but more simple phenotype: DTF or DTS. All three phenotypes are different chronological steps in the organism's life cycle, with the earliest milestone being DTF and the latest being DTM. Simple reasoning would argue that DTF would also be the most simple phenotype, because DTS would be affected in large part by the DTF phenotype, along with additional genetic factors that determine the number of days from flowering to swollen. Similarly, DTM would be affected by the genetic factors determining DTF and DTS, along with additional genetic factors that determine the number of days from swollen to maturity. If this is true then it makes sense why DTM would be the best candidate for model 1 to be used for transfer learning, as a DTM model would learn to recognize the most genetic factors that could then be used to predict other phenotypes. More evidence for this argument comes when we look at transfer learning for predicting the DTM phenotype. DTM prediction achieves the least gain from transfer learning. This could be because models that predict DTF or DTS have not learned to find the genetic features necessary for prediction of DTM. In the use case where a user downloads a model to use as model 1, a model that predicts a complex phenotype will detect more genetic features and be more useful than a model that only predicts a Mendelian trait and therefore the model only learns to detect one genetic feature.

The study using *N. gonorrhoeae* data showed that transfer learning works when datasets have a relatively large number of samples, but in this study a small number of samples were present when training both models 1 and 2. This means that smaller datasets can be useful, and allows researchers to make use of datasets that involve less genetic sequencing and more phenotype measurement if that is all that is available. However, datasets with more samples will always perform better than datasets using transfer learning with fewer samples (assuming the quality of the dataset is equal). Suppose that Y is the minimum acceptable number of samples to adequately train a phenotype prediction model and $2X = Y$. Let us suppose that we conduct two experiments with characteristics as follows. Experiment A uses a dataset with Y samples and only the phenotype labels for one phenotype, and experiment B uses a dataset with X samples and phenotype labels for two phenotypes. In experiment A we cannot use transfer learning, but in experiment B we can use transfer learning to give us $2X = Y$ training cases. Experiment A will still perform better than experiment B because experiment B is trained on less genetic variation than experiment A is trained on, as well as less phenotype-specific information. Experiment B, however, will perform much better having made use of transfer learning than it would have without transfer learning, in which case it could have only made use of X training samples.

A poster was presented with preliminary results for this experiment at ISMB2019 [45] as well as at the 2019 P2IRC symposium [46]. A flash talk was also presented to go along with the poster at the 2019 P2IRC symposium. This poster is hosted online at F1000research [47] and can also be found in the file `kopas-poster-3.pdf` in Supplementary Materials.

5.2.4 Future Work

There are multiple different experiments and analyses that could extend from this experiment as future work. Firstly, additional work could be done to further tune the hyperparameters and network architecture. This may yield better predictive power for the control and transfer learning models. Additionally, the number of layers that are frozen when transferring from model 1 to model 2 could be explored in more depth. In the same vein, transfer learning across three or more models could be explored. This would require different numbers of layers being frozen at each transfer step, but could potentially allow for the combination of many different genetic features into a model that generalizes for many different phenotype prediction tasks.

One additional analysis that could be performed would be to run models 1 and 2 through SHAP [10] or another model explainer or interpreter. This would yield insight into which SNPs each model found significant. SNPs or combinations thereof that are found to be significant in model 2 may be candidates for further biological exploration as these SNPs may be involved in pathways that are shared between the control and transfer phenotypes. For example, if a SNP was significant for both azithromycin and cefixime AMR classification then it may indicate that there is a shared genetic feature between the resistance mechanisms for both antibiotics.

Another follow up experiment is to see if transfer learning can be applied across related species. This could only be done in very closely related species, or by only using a subset of the SNPs, because only homologous SNPs would benefit from transfer learning. However, if transfer learning could be used across species in this way it would lead to two benefits. First, it would allow for any deep learning model repository that is set up to be more useful as a deep learning model created for one species could be used for transfer learning studies for related organisms. Second, it would yield interesting biological insights into evolutionary genetic pathways and indicate similar genetic mechanisms in related but different species.

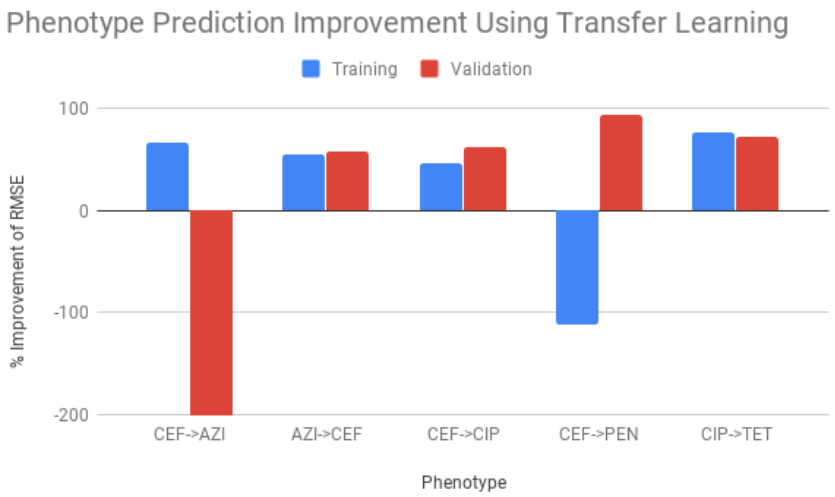


Figure 5.13: Improvement achieved for phenotype prediction via transfer learning. The values reported are the percentage the RMSE improved when using transfer learning compared to the RMSE of the control for *N. gonorrhoeae*. The blue bars are the RMSE differences observed on the training data and the red bars are the RMSE differences observed on the test data. Only the transfer cases where model 2 had the lowest validation loss are shown. For the results of all cases refer to Figure 5.15. Note that for the cefixime case the validation loss improvement was approximately -400%. This is likely due to overfitting in model 2. The abbreviations are as follows: AZI (azithromycin), CEF (cefixime), CIP (ciprofloxacin), PEN (penicillin), and TET (tetracycline).

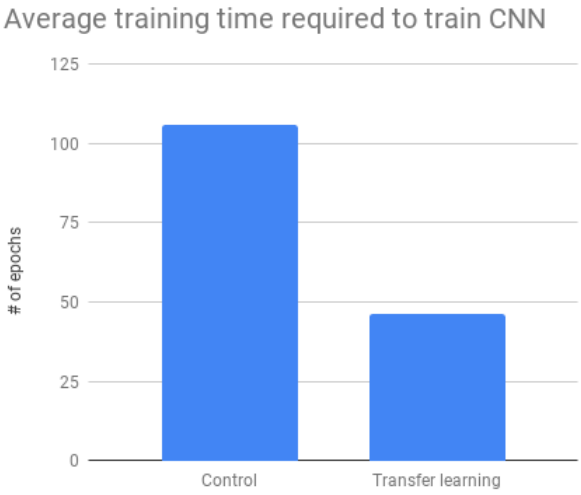
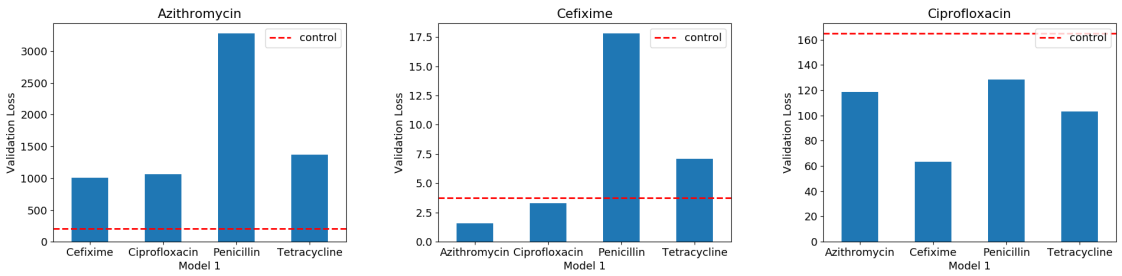


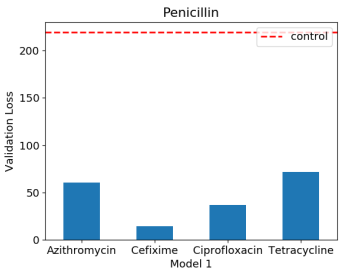
Figure 5.14: Average training time required for phenotype prediction with and without transfer learning. Less than half the number of training epochs were required for a phenotype prediction model to plateau when transfer learning was utilized compared to the control.



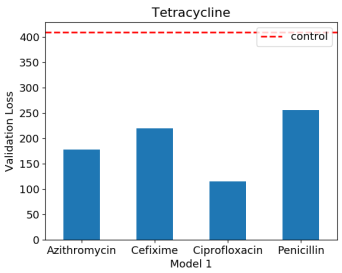
(a) Azithromycin

(b) Cefixime

(c) Ciprofloxacin



(d) Penicillin



(e) Tetracycline

Figure 5.15: Validation loss values when transfer learning was applied for phenotype prediction of each antimicrobial. Each panel is for the prediction of a single phenotype (model 2). Along the x-axis is the particular model used for model 1, and the red line is the validation loss of the control model. Any bar lower than the red line achieved better results than the control model. Only the case of azithromycin prediction did not yield any models that outperformed the control.

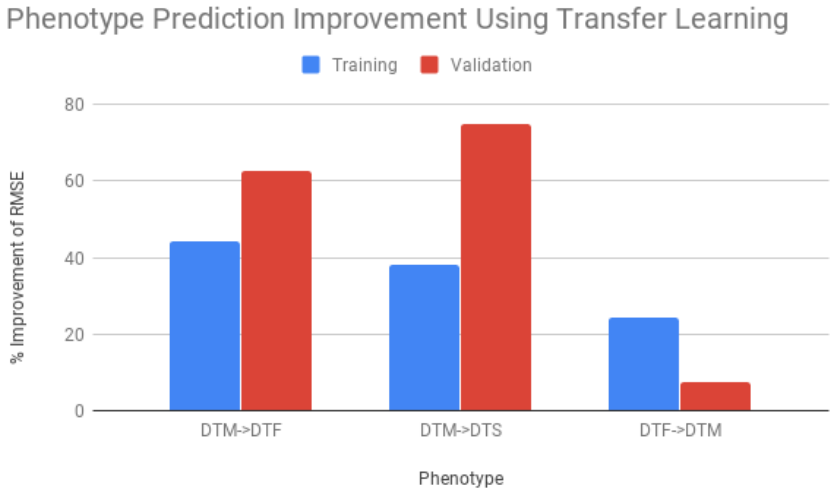
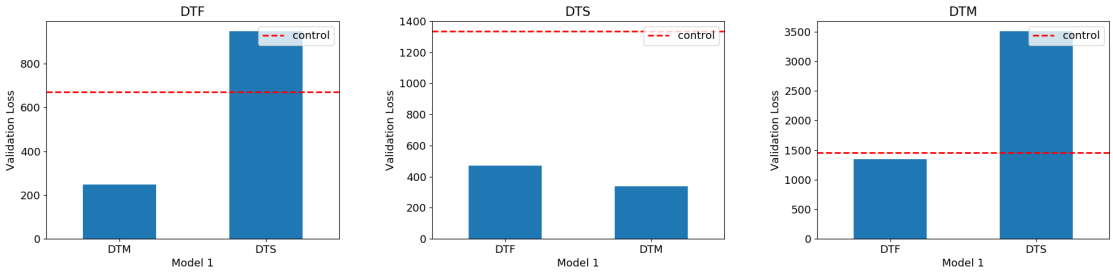


Figure 5.16: Improvement achieved for phenotype prediction via transfer learning. The values reported are the percentage the RMSE improved when using transfer learning compared to the RMSE of the control for *L. culinaris*. Only the cases where transfer learning was most effective are shown in this figure. For the results of all cases refer to Figure 5.17.



(a) DTF **(b)** DTS **(c)** DTM

Figure 5.17: Validation loss values when transfer learning was applied for phenotype prediction of each phenotype. Each panel is for the prediction of a single phenotype (model 2). Along the x-axis is the particular model used for model 1, and the red line is the validation loss of the control model. Any bar lower than the red line achieved better results than the control model.

Chapter 6

Conclusion

Deep learning is a powerful tool that shows great potential for learning genotype to phenotype associations. However, with the datasets available at the time of writing, deep learning techniques must be improved in order predict phenotype effectively. This thesis presents 2 techniques that help improve technology to that end.

First, we explored two options for representing SNP data as input to a CNN. We showed that representing SNP data as a vector of genotype likelihoods can lead to improved phenotype prediction in some cases. However, this is not always the case and more work must be done to determine if PIR is an improvement over CIR. Our results indicated that when using high-dimensional datasets with high quality SNPs, PIR and CIR representation perform similarly. Our results do not indicate that a deep learning model is able to learn any additional information via the encoding of quality score information and the removal of low quality SNPs from the CIR model does not have significant adverse effects. Future work may show that in other scenarios these two input representation models do not always perform the same.

Second, we showed that transfer learning can be used to improve phenotype prediction by CNNs and also to reduce training time. This allows researchers to make better use of computing resources as well as to make more effective use of available datasets. Showing that transfer learning is an effective way to improve phenotype prediction allows researchers to make better use of small datasets. Hopefully, this work will prompt a movement to share trained CNN models with others who can download these models and use them as a starting point for transfer learning. This will allow for researchers to select a similar model that will effectively increase the size of their dataset and reduce the training time required to train a predictive model. We also explored the possibility that transfer learning is able to identify alleles involved in common phenotype pathways and generate novel hypotheses that can later be confirmed in biological studies.

As a baseline, the results for MIC prediction when using RR-BLUP are provided in Appendix D.1. RR-BLUP outperformed our deep learning models in all cases. It is evident that deep learning methods do not measure up to the state of the art in this experiment; however, phenotype prediction may become one more where deep learning replaces the current state of the art in the future.

Taken together, both of the techniques explored in this thesis improve the current technology for deep learning phenotype prediction with the goal of making it as useful as RR-BLUP. By improving phenotype prediction using transfer learning, crop breeders can use phenotype prediction to determine which breeding

lines have the most potential. Improved phenotype prediction might also allow health practitioners to determine what antibiotics will be effective against a bacterial infection before prescribing ineffective antibiotics. This will reduce treatment times and also slow the rate of growing antimicrobial resistance.

Improving phenotype prediction has many practical benefits, and this thesis works to take phenotype prediction using deep learning a step forward making it more effective and a promising candidate for phenotype prediction and other genetic association studies.

Bibliography

[1] Balázs Csanád Csáji. Approximation with artificial neural networks. *Faculty of Sciences, Etvs Lornd University, Hungary*, 24:48, 2001.

[2] C. R. Henderson. *Applications of Linear Models in Animal Breeding.* University of Guelph, Guelph, Canada, 1984.

[3] Jeffrey B. Endelman. Ridge Regression and Other Kernels for Genomic Selection with R Package rrBLUP. *The Plant Genome*, 4(3):250–255, November 2011.

[4] Wenlong Ma, Zhixu Qiu, Jie Song, Qian Cheng, and Chuang Ma. DeepGS: Predicting phenotypes from genotypes using Deep Learning. *bioRxiv*, page 241414, December 2017.

[5] Biyue Tan, Dario Grattapaglia, Gustavo Salgado Martins, Karina Zamprogno Ferreira, Björn Sundberg, and Pär K. Ingvarsson. Evaluating the accuracy of genomic prediction of growth and wood traits in two eucalyptus species and their F_1 hybrids. *BMC Plant Biology*, 17(1):110–110, Jun 2017. PMC5492818[pmcid].

[6] Marcus Nguyen, S. Wesley Long, Patrick F. McDermott, Randall J. Olsen, Robert Olson, Rick L. Stevens, Gregory H. Tyson, Shaohua Zhao, and James J. Davis. Using machine learning to predict antimicrobial mics and associated genomic features for nontyphoidal *Salmonella*. *Journal of Clinical Microbiology*, 57(2), 2019.

[7] Honglak Lee, Chaitanya Ekanadham, and Andrew Y. Ng. Sparse deep belief net model for visual area V2. In J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 873–880. Curran Associates, Inc., 2008.

[8] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255, 2009.

[9] Jinhong Shi, Yan Yan, Matthew Links, Longhai LI, Jo-Anne Dillon, Michael Horsch, and Anthony Kusalik. Antimicrobial resistance genetic factor identification from whole-genome sequence data using deep feature selection. *BMC Bioinformatics*, 20, 12 2019.

[10] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4765–4774. Curran Associates, Inc., 2017.

[11] Muhammad Khalid Khan Niazi, Thomas Erol Tavolara, Vidya Arole, Douglas J Hartman, Liron Pantanowitz, and Metin N Gurcan. Identifying tumor in pancreatic neuroendocrine neoplasms from ki67 images using transfer learning. *PloS One*, 13(4):e0195621–e0195621, 2018.

[12] Sung-Jin Kim, Chuangqi Wang, Bing Zhao, Hyungsoon Im, Jouha Min, Hee June Choi, Joseph Tadros, Nu Ri Choi, Cesar M Castro, Ralph Weissleder, Hakho Lee, and Kwonmoo Lee. Deep transfer learning-based hologram classification for molecular diagnostics. *Scientific Reports*, 8(1):17003–17003, 2018.

[13] F. Sanger and A. R. Coulson. A rapid method for determining sequences in DNA by primed synthesis with DNA polymerase. *Journal of Molecular Biology*, 94(3):441–448, May 1975.

[14] Hengyun Lu, Francesca Giordano, and Zemin Ning. Oxford Nanopore MinION Sequencing and Genome Assembly. *Genomics, Proteomics & Bioinformatics*, 14(5):265–279, October 2016.

[15] Ryan Poplin, Dan Newburger, Jojo Dijamco, Nam Nguyen, Dion Loy, Sam S. Gross, Cory Y. McLean, and Mark A. DePristo. Creating a universal SNP and small indel variant caller with deep neural networks. *bioRxiv*, page 092890, January 2018.

[16] M. E. Goddard and B. J. Hayes. Genomic selection. *Journal of Animal Breeding and Genetics = Zeitschrift Fur Tierzuchtung Und Zuchtungsbiologie*, 124(6):323–330, December 2007.

[17] T. H. Meuwissen, B. J. Hayes, and M. E. Goddard. Prediction of total genetic value using genome-wide dense marker maps. *Genetics*, 157(4):1819–1829, April 2001.

[18] Kyall R. Zenger, Mehar S. Khatkar, David B. Jones, Nima Khalilisamani, Dean R. Jerry, and Herman W. Raadsma. Genomic selection in aquaculture: Application, limitations and opportunities with special reference to marine shrimp and pearl oysters. *Frontiers in Genetics*, 9:693, 2019.

[19] E. Garrison and G. Marth. Haplotype-based variant detection from short-read sequencing. *arXiv preprint arXiv:1207.3907 [q-bio.GN]*, July 2012.

[20] H. Li. Improving SNP discovery by base alignment quality. *Bioinformatics*, 27(8):1157–1158, April 2011.

[21] Abhineet Saxina. Convolutional Neural Networks (CNNs): An Illustrated Explanation. `https://blog.xrds.acm.org/2016/06/convolutional-neural-networks-cnns-illustrated-explanation/`, June 2016.

[22] Sasank Chilamkurthy, Rohit Ghosh, Swetha Tanamala, Mustafa Biviji, Norbert G Campeau, Vasantha Kumar Venugopal, Vidur Mahajan, Pooja Rao, and Prashant Warier. Deep learning algorithms

for detection of critical findings in head ct scans: a retrospective study. *The Lancet*, 392(10162):2388 – 2396, 2018.

[23] Florian Dubost, Hieab Adams, Gerda Bortsova, M. Arfan Ikram, Wiro Niessen, Meike Vernooij, and Marleen de Bruijne. 3d regression neural network for the quantification of enlarged perivascular spaces in brain mri. *Medical Image Analysis*, 51:89 – 100, 2019.

[24] Ritambhara Singh, Jack Lanchantin, Gabriel Robins, and Yanjun Qi. DeepChrome: deep-learning for predicting gene expression from histone modifications. *Bioinformatics*, 32(17):i639–i648, 08 2016.

[25] Arttu Jolma, Jian Yan, Thomas Whitington, Jarkko Toivonen, Kazuhiro R. Nitta, Pasi Rastas, Ekaterina Morgunova, Martin Enge, Mikko Taipale, Gonghong Wei, Kimmo Palin, Juan M. Vaquerizas, Renaud Vincentelli, Nicholas M. Luscombe, Timothy R. Hughes, Patrick Lemaire, Esko Ukkonen, Teemu Kivioja, and Jussi Taipale. DNA-binding specificities of human transcription factors. *Cell*, 152(1):327 – 339, 2013.

[26] Babak Alipanahi, Andrew Delong, Matthew T. Weirauch, and Brendan J. Frey. Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning. *Nature Biotechnology*, 33(8):831–838, Aug 2015.

[27] Kai-Lung Hua, Che-Hao Hsu, Shintami Chusnul Hidayati, Wen-Huang Cheng, and Yu-Jen Chen. Computer-aided classification of lung nodules on computed tomography images via deep learning technique. *OncoTargets and Therapy*, 8:2015–2022, Aug 2015. PMC4531007[pmcid].

[28] Jun Zhang and K. F. Man. Time series prediction using rnn in multi-dimension embedding phase space. In *SMC'98 Conference Proceedings. 1998 IEEE International Conference on Systems, Man, and Cybernetics*, volume 2, pages 1868–1873, San Diego, CA, USA, Oct 1998.

[29] Walter Demczuk, Irene Martin, Shelley Peterson, Amrita Bharat, Gary Van Domselaar, Morag Graham, Brigitte Lefebvre, Vanessa Allen, Linda Hoang, Greg Tyrrell, Greg Horsman, John Wylie, David Haldane, Chris Archibald, Tom Wong, Magnus Unemo, and Michael R. Mulvey. Genomic Epidemiology and Molecular Resistance Mechanisms of Azithromycin-Resistant *Neisseria gonorrhoeae* in Canada from 1997 to 2014. *Journal of Clinical Microbiology*, 54(5):1304–1313, 2016.

[30] Yonatan H. Grad, Robert D. Kirkcaldy, David Trees, Janina Dordel, Simon R. Harris, Edward Goldstein, Hillard Weinstock, Julian Parkhill, William P. Hanage, Stephen Bentley, and Marc Lipsitch. Genomic epidemiology of *Neisseria gonorrhoeae* with reduced susceptibility to cefixime in the USA: a retrospective observational study. *The Lancet Infectious Diseases*, 14(3):220–226, March 2014.

[31] Dilrini De Silva, Joanna Peters, Kevin Cole, Michelle J Cole, Fiona Cresswell, Gillian Dean, Jayshree Dave, Daniel Rh Thomas, Kirsty Foster, Alison Waldram, Daniel J Wilson, Xavier Didelot, Yonatan H Grad, Derrick W Crook, Tim Ea Peto, A Sarah Walker, John Paul, and David W Eyre. Whole-genome

sequencing to determine transmission of *Neisseria gonorrhoeae*: an observational study. *The Lancet Infectious Diseases*, 16(11):1295–1303, November 2016.

[32] Gyung Tae Chung, Jeong Sik Yoo, Hee Bok Oh, Yeong Seon Lee, Sun Ho Cha, Sang Jun Kim, and Cheon Kwon Yoo. Complete Genome Sequence of *Neisseria gonorrhoeae* NCCP11945. *Journal of Bacteriology*, 190(17):6035–6036, September 2008.

[33] Heng Li and Richard Durbin. Fast and accurate long-read alignment with Burrows–Wheeler transform. *Bioinformatics*, 26(5):589–595, 01 2010.

[34] L. Barth Reller, Melvin Weinstein, James H. Jorgensen, and Mary Jane Ferraro. Antimicrobial Susceptibility Testing: A Review of General Principles and Contemporary Practices. *Clinical Infectious Diseases*, 49(11):1749–1755, December 2009.

[35] Teketel A. Haile, Taryn Heidecker, Derek Wright, Sandesh Neupane, Larissa Ramsay, Albert Vandenberg, and Kirstin E. Bett. Genomic selection for lentil breeding: Empirical evidence. *The Plant Genome*, 13(1):e20002, 2020.

[36] L. Ramsay, C Chan, AG Sharpe, DR Cook, RV Penmetsa, P Chang, C Coyne, R McGee, D Main, D Edwards, S Kaur, A Vandenberg, and KE Bett. Lens culinaris *CDC Redberry genome assembly v1.2*, 2016. Retrieved from `https://acsess.onlinelibrary.wiley.com/doi/abs/10.1002/tpg2.20002`.

[37] Heng Li, Bob Handsaker, Alec Wysoker, Tim Fennell, Jue Ruan, Nils Homer, Gabor Marth, Goncalo Abecasis, Richard Durbin, and 1000 Genome Project Data Processing Subgroup. The Sequence Alignment/Map format and SAMtools. *Bioinformatics (Oxford, England)*, 25(16):2078–2079, August 2009.

[38] François Chollet et al. Keras. `https://keras.io`, 2015.

[39] Logan Kopas, Tony Kusalik, Dave Schneider, and Jinhong Shi. Utilizing SNP quality scores to increase deep neural network accuracy in genomic studies. Poster at ResearchFest, Saskatoon, October 2018.

[40] Logan Kopas, Tony Kusalik, Dave Schneider, and Jinhong Shi. Utilizing SNP quality scores to increase deep neural network accuracy in genomic studies. Poster at P2IRC Symposium, Saskatoon, October 2018.

[41] Abien Fred Agarap. Deep learning using rectified linear units (relu), 2018.

[42] Public Health Agency of Canada: National surveillance of antimicrobial susceptibilities of *Neisseria gonorrhoeae* annual summary 2014, 2014.

[43] Medlineplus. `https://medlineplus.gov/`. Updated 2019 Aug 27.

[44] Elizabeth A. Ohneck, Yaramah M. Zalucki, Paul J. T. Johnson, Vijaya Dhulipala, Daniel Golparian, Magnus Unemo, Ann E. Jerse, and William M. Shafer. A novel mechanism of high-level, broad-spectrum antibiotic resistance caused by a single base pair change in *Neisseria gonorrhoeae*. *mBio*, 2(5), 2011.

[45] Logan Kopas, Tony Kusalik, and Dave Schneider. Antimicrobial resistance prediction from whole-genome sequence data using transfer learning. Poster at ISMB2019, Basel, Switzerland, October 2019.

[46] Logan Kopas, Tony Kusalik, and Dave Schneider. Antimicrobial resistance prediction from whole-genome sequence data using transfer learning. Poster at P2IRC Symposium, Saskatoon, October 2019.

[47] Logan Kopas, Anthony Kusalik, and Dave Schneider. Antimicrobial resistance prediction from whole-genome sequence data using transfer learning. *F1000Research*, 8, August 2019.

[48] W.J. Ewens. The sampling theory of selectively neutral alleles. *Theoretical Population Biology*, 3(1):87 – 112, 1972.

Appendix A

Glossary

backpropagation When a neural network updates its weights based on a gradient descent to minimize the loss function. 5–7, 12

bias Each trainable layer in a neural network learns both weights and biases. Biases are additive; the dot product of the weights and the inputs are taken before the bias term is added and the result is fed throught the activation function. Weights are then updated during backpropagation. 5

curse of dimensionality The curse of dimensionality occurs when working with data in high dimensional feature space. When the number of features increases, the feature space grows so large that the data becomes so sparse. When working with sparse data it becomes difficult to train a machine learning model because there are many possible hyperplanes that can separate the data and many of them will not generalize to real data. 17, 52

early stopping function An early stopping function is a regularization method that prevents a machine learning model from overfitting by stopping training when a certain criteria is met. This criteria can change based on requirements, but for gradient descent this function aims to stop training when the loss function reaches the global minimum while not stopping training in a local minima. 6, 35, 38, 43, 46

explainer A tool used to explain how a black box machine learning model such as a deep neural network computes output. This is often done by permuting different inputs and observing the affect the permutation has on the output. 11

filter A single set of weights learned by a convolution in a neural network. Each layer can be made up of multiple filters that are applied separately to the inputs. 7

hyperparameter Any parameter that must be set before training a machine learning model. Common examples are the learning rate for a neural network or the convolution size for a convolutional layer. 6, 7, 22

indel An insertion or deletion of genetic material in regards to the reference genome. 63

inference A single forward pass of a neural network. A neural network is provided inputs and produces output based on its internal model. 6

layer A set of neurons in a nueral network. There are multiple types of layers. In a CNN neurons in a layer are only connected to those in the previous layer, there are no connections between neurons in the same layer. 5

linkage disequilibrium Linkage disequilibrium occurs when two alleles at different loci are inherited together. This can occur because they are located close to each other on a chromosome, and it means that the alleles are not independent. This can be leveraged to improve the power of genetic association studies. 14, 15

neuron The smallest entity of a neural network. Each neuron stores a single value. Neurons are connected to other neurons in the preceeding and following layers (unless the layer in question is the first or last layer. The value stored in a neuron is the result of some function that is used to transform the values from the neurons in the previous layer. These functions can involve learned weights and biases, or they can be non-trainable functions; the function depends on the type of layer. 5

overfitting When a neural network fails to create a general model and instead memorizes which inputs map to which outputs. 6, 49, 52

quantitative trait loci Quantitative trait loci are sections of DNA that are correlated with a certain quantitative phenotype in a population. 14

sequencing depth The number of times a genome is replicated before it is sequenced. A dataset with a sequencing depth of 30 would mean that there were 30 copies of the entire genome present when sequencing was performed. 13

weight Each trainable layer in a neural network learns both weights and biases. Weights are multiplicative; the dot product of the weights and the inputs are taken before the bias term is added and the result is fed throught the activation function. Weights are then updated during backpropagation. 5

Appendix B

SNP Caller Expansion

B.1 FreeBayes

This chapter is a deeper look at the equations introduced in Section 2.3.3, which are used in the FreeBayes SNP calling package [19].

FreeBayes uses Bayesian statistics to find the genotype combination that maximizes $P(G_1, ..., G_n|R_1, ..., R_n)$, where G_i is the underlying genotype at locus i and R_i is the set of reads from the same locus. Using Bayes' Rule and assuming that sequencing reads are independent, we get Equation 2.4, which we restate here for convenience:

$$P(G_1, ..., G_n|R_1, ..., R_n) = \frac{P(G_1, ..., G_n) \prod_{i=1}^{n} P(R_i|G_i)}{\sum_{\forall G_1,...,G_n} (P(G_1, ..., G_n) \prod_{i=1}^{n} P(R_i|G_i))}. \tag{B.1}$$

If we had perfect observations, $P(R_i|G_i)$ would approximate the probability of sampling observations R_i out of G_i with replacement. Our observations are not perfect, however, so FreeBayes accounts for the probability of errors in the reads. $P(R_i|G_i)$ is estimated taking into account the set of all possible underlying genotypes, including the probability of sequencing errors, summed across all R_i combinations to get the joint probability of R_i given G_i. The FreeBayes paper provides specific details [19].

The prior probability of observing the set of genotypes $G_1, ..., G_n$ is equivalent to the intersection of the probability of the genotype and the probability of the corresponding set of allele frequencies, $f_1, ..., f_k$. Garrison *et al.* note that "this identity follows from the fact that the allele frequencies are derived from the set of genotypes and we always will have the same $f_1, ..., f_k$ for any equivalent $G_1, ..., G_n$" [19, p.4].

$$P(G_1, ..., G_n) = P(G_1, ..., G_n \cap f_1, ..., f_k) \tag{B.2}$$

and via Bayes' Rule

$$P(G_1, ..., G_n \cap f_1, ..., f_k) = P(G_1, ..., G_n|f_1, ..., f_k)P(f_1, ..., f_k). \tag{B.3}$$

This equation can now be estimated in terms of the genotype combination sampling probability and the probability of observing a particular allele frequency in the given population. The former is estimated differently depending on whether the genotype combinations are phased or unphased, and the latter is estimated using Ewen's sampling formula [48]. Both are beyond the scope of this document, but are described in more detail in the FreeBayes paper [19].

FreeBayes attempts to find the set of genotypes that maximizes the posterior probability using a gradient ascent approach. Starting with the maximum likelihood genotype given the data likelihood,

$$G_1, ..., G_n = \underset{G_i}{\operatorname{argmax}}\ P(R_i|G_i), \tag{B.4}$$

FreeBayes then searches local genotype space starting with $G_1, ..., G_n = \{G\}$ and attempts to find $\{G\}'$ such that $P(\{G\}'|R_1, ..., R_n) > P(\{G\}|R_1, ..., R_n)$. This search is carried out until convergence, but with a maximum number of iterations in order to ensure the program terminates in a timely fashion.

FreeBayes reports a quality estimate for each possible genotype at each loci by summing over the marginal probability of that specific genotype and sample combination under the model:

$$P(G_j|R_i, ..., R_n) = \sum_{\forall (\{G\}:G_j \in \{G\})} P(\{G\}|R_i, ..., R_n). \tag{B.5}$$

This probability can be converted to a PHRED quality score. If P is the probability of a genotype as reported by FreeBayes then the PHRED quality score can be calculated using this formula:

$$Q = -10 \log_{10} P. \tag{B.6}$$

FreeBayes reports the PHRED quality score for the called genotype and the log_{10} scaled posterior probability for each possible genotype at each location.

B.2 Samtools

The samtools `mpileup` software uses an HMM in order to discover SNPs in a set of reads. The HMM consists of 5 states: alignment mismatches (M), insertions to the reference (I), deletions from the reference (D), alignment start (S), and alignment end (E). The S state points to every M and I state, while every M and I point to E. The HMM is evaluated to find the maximum likelihood alignment, which defines the positions of indels and mutations.

If L is the length of the reference sequence and l is the length of each read, then we define the reference nucleotide sequence to be $r = r_1, ..., r_L$, and the read sequence to be $c = c_0, c_1, ..., c_l, c_{l+1}$ where c_0 marks the start of the read and c_{l+1} marks the end of the read. We also must define $\epsilon_i, i = 1...l$ as a substitution probability for c_i and M as the function of substitution probabilities, $\{\epsilon_i\}$. We also define the forward matrix, f, and the backward matrix, b. These are further defined in the BAQ paper [20].

In order to calculate the BAQ, we define an alignment A as a set of coordinate pairs $\{(i_1, k_1), ..., (i_p, k_p)\}$ in increasing coordinate order, with $1 \leq i_1 < ... < i_p \leq l$ and $1 \leq k_1 < ... < k_p \leq L$. We also define k_i to be k if $(i, k) \in A$, 0 otherwise. The BAQ is reported as the PHRED-scaled score,

$$Q(i|A) = -10 \log_{10}(1 - Pr(i\text{-th read base aligned to } k_i|A)) \tag{B.7}$$

$$Q(i|A) = -10 \log_{10}(1 - \frac{f_{i,M_{k_i}} \cdot b_{i,M_{k_i}}}{f_{l+1,E}}) \tag{B.8}$$

These BAQ scores are reported by `mpileup` as the PHRED quality score for each possible genotype at each locus.

Appendix C

Accessions

Table C.1: List of accession numbers used in the *N. gonorrhoeae* dataset. These accessions can be found in the SRA.

US-ERR223688	US-ERR191816	UK-SRR3361311	UK-SRR3360736	UK-SRR2736286	UK-SRR3357263
US-ERR223636	UK-SRR3360713	UK-SRR3360625	US-ERR223658	UK-SRR2736242	UK-SRR3360645
UK-SRR3360970	UK-SRR2736132	US-ERR191795	UK-SRR2736170	US-ERR191753	US-ERR191746
UK-SRR3360944	UK-SRR3360743	UK-SRR2736157	UK-SRR2736293	US-ERR191819	UK-SRR2736218
US-ERR191824	UK-SRR3360702	UK-SRR3360981	US-ERR223632	UK-SRR3349544	UK-SRR3360687
US-ERR223608	US-ERR223640	UK-SRR2736140	UK-SRR2736240	UK-SRR3360733	UK-SRR2736169
UK-SRR2736266	US-ERR223606	US-ERR223695	US-ERR191774	UK-SRR2736179	UK-SRR2736191
UK-SRR3360693	UK-SRR3357288	UK-SRR1661245	UK-SRR2736136	UK-SRR3360606	UK-SRR2736102
UK-SRR2736111	UK-SRR1661250	US-ERR191757	US-ERR223696	UK-SRR2736196	US-ERR191809
UK-SRR1661155	US-ERR191739	UK-SRR3361326	UK-SRR2736228	UK-SRR3360889	US-ERR191731
UK-SRR3360752	UK-SRR2736139	US-ERR223634	UK-SRR1661168	UK-SRR3361346	US-ERR191802
UK-SRR3360708	UK-SRR3360638	US-ERR223613	UK-SRR2736145	UK-SRR3360628	UK-SRR2736110
UK-SRR3360720	US-ERR191751	UK-SRR1661262	US-ERR191741	US-ERR223694	UK-SRR3357028
UK-SRR3360696	UK-SRR3360659	US-ERR223651	UK-SRR3360936	UK-SRR2736290	UK-SRR2736130
UK-SRR2736173	UK-SRR1661329	US-ERR223629	US-ERR191737	UK-SRR3360712	US-ERR191825
UK-SRR3360648	UK-SRR3360675	US-ERR191750	UK-SRR2736189	UK-SRR3360678	UK-SRR1661207
US-ERR223657	UK-SRR2736105	UK-SRR1661227	UK-SRR3357229	UK-SRR3360607	US-ERR191806
UK-SRR3360692	UK-SRR2736152	UK-SRR3360635	UK-SRR3361345	UK-SRR2736279	UK-SRR2736257
US-ERR223680	UK-SRR3360715	UK-SRR3361329	US-ERR191821	UK-SRR2736246	US-ERR223639
UK-SRR2736252	UK-SRR3360747	UK-SRR3357181	UK-SRR2736095	UK-SRR2736114	UK-SRR3360636
UK-SRR3361336	US-ERR191822	UK-SRR2736212	UK-SRR3360694	US-ERR223625	UK-SRR3349726
UK-SRR2736269	UK-SRR2736166	UK-SRR3360813	UK-SRR2736148	UK-SRR2736274	UK-SRR2736211
UK-SRR2736121	US-ERR191791	UK-SRR3360634	US-ERR223686	UK-SRR2736184	UK-SRR3360734
UK-SRR3360921	US-ERR191730	UK-SRR3360751	UK-SRR3360703	UK-SRR3360883	UK-SRR2736163
UK-SRR3360614	UK-SRR3360689	US-ERR223656	UK-SRR2736213	UK-SRR3349673	UK-SRR2736225
UK-SRR3360651	UK-SRR2736294	UK-SRR3360617	US-ERR223692	US-ERR191752	UK-SRR1661331
UK-SRR3360653	UK-SRR2736198	UK-SRR2736237	US-ERR223646	UK-SRR3360735	UK-SRR3360616
UK-SRR3360913	UK-SRR3360993	US-ERR191769	UK-SRR2736272	UK-SRR2736234	UK-SRR3360672
UK-SRR1661279	US-ERR191788	UK-SRR2736171	UK-SRR3361353	UK-SRR2736142	UK-SRR2736284
UK-SRR2736270	UK-SRR3360716	US-ERR223645	UK-SRR3360662	US-ERR191768	UK-SRR2736235
US-ERR191780	US-ERR191778	UK-SRR3360930	US-ERR191754	UK-SRR3360679	UK-SRR3361325
US-ERR191796	US-ERR223671	UK-SRR3360950	UK-SRR2736303	UK-SRR3360685	UK-SRR2736255
UK-SRR2736096	UK-SRR2736143	UK-SRR3360917	UK-SRR3360618	US-ERR223647	UK-SRR2736227
UK-SRR2736161	UK-SRR3349557	US-ERR223663	US-ERR191814	UK-SRR3360893	UK-SRR3361338
UK-SRR1661330	UK-SRR2736288	US-ERR223620	US-ERR191786	US-ERR191766	UK-SRR2736256
UK-SRR2736265	UK-SRR3360646	UK-SRR3360639	US-ERR223679	UK-SRR2736305	US-ERR223660
UK-SRR3361349	UK-SRR3360731	US-ERR223661	US-ERR223623	UK-SRR2736106	UK-SRR3360982
UK-SRR3349554	UK-SRR3360632	UK-SRR2736108	US-ERR223681	UK-SRR3360718	US-ERR223604
UK-SRR3361308	UK-SRR2736127	UK-SRR3360980	UK-SRR2736243	UK-SRR2736124	UK-SRR2736181
US-ERR191761	US-ERR223653	UK-SRR2736208	UK-SRR3360722	UK-SRR3361318	US-ERR223689
US-ERR191777	US-ERR223631	UK-SRR2736304	UK-SRR3361347	UK-SRR2736215	US-ERR223673
US-ERR191798	UK-SRR1661223	US-ERR191775	UK-SRR3360700	UK-SRR2736100	UK-SRR2736135
UK-SRR3357289	UK-SRR1661211	UK-SRR2736147	UK-SRR2736261	UK-SRR3361332	UK-SRR3360640
UK-SRR3360924	UK-SRR2736197	UK-SRR3361356	US-ERR191781	UK-SRR2736205	UK-SRR2736150

US-ERR223697	UK-SRR3360613	US-ERR223637	UK-SRR3360714	UK-SRR3361324	US-ERR223668
UK-SRR3360745	UK-SRR2736245	UK-SRR3360854	UK-SRR2736276	UK-SRR3360750	UK-SRR3360832
UK-SRR2736249	UK-SRR2736192	US-ERR191804	UK-SRR2736133	US-ERR223605	UK-SRR2736223
US-ERR191733	UK-SRR2736230	US-ERR223641	US-ERR191748	UK-SRR3361344	UK-SRR3360683
UK-SRR3360680	UK-SRR3360983	UK-SRR2736275	US-ERR191763	UK-SRR2736101	UK-SRR2736203
UK-SRR3360663	UK-SRR3360867	UK-SRR3360629	US-ERR191776	UK-SRR3360691	UK-SRR2736201
UK-SRR2736306	UK-SRR3360664	UK-SRR3357011	UK-SRR3361328	UK-SRR3360641	US-ERR223691
UK-SRR3360610	UK-SRR1661167	UK-SRR3360709	US-ERR223677	UK-SRR3360707	UK-SRR2736221
UK-SRR2736226	UK-SRR2736244	UK-SRR3357186	UK-SRR2736233	UK-SRR3360727	UK-SRR2736253
UK-SRR3360608	UK-SRR2736190	US-ERR223664	UK-SRR3360661	UK-SRR3360605	UK-SRR3360652
US-ERR223612	UK-SRR3349615	UK-SRR2736165	UK-SRR3361337	UK-SRR3360637	UK-SRR3360697
US-ERR223611	UK-SRR2736300	US-ERR223648	US-ERR191772	UK-SRR3360811	US-ERR191823
UK-SRR2736224	US-ERR223626	UK-SRR2736271	US-ERR223624	UK-SRR3360740	UK-SRR3360658
US-ERR223610	UK-SRR1661315	UK-SRR1661249	US-ERR223630	UK-SRR2736156	UK-SRR2736210
UK-SRR2736155	US-ERR191812	UK-SRR3361320	US-ERR223672	US-ERR191732	US-ERR223690
UK-SRR3360654	US-ERR223638	UK-SRR2736122	UK-SRR3361340	US-ERR223667	UK-SRR2736120
US-ERR223675	UK-SRR2736104	UK-SRR2736174	UK-SRR3357157	UK-SRR2736134	UK-SRR3360753
UK-SRR3360890	US-ERR223621	UK-SRR2736277	UK-SRR2736128	UK-SRR1661199	UK-SRR1661322
US-ERR223659	US-ERR223670	US-ERR223654	UK-SRR2736162	UK-SRR1661153	US-ERR191800
US-ERR191735	UK-SRR2736125	UK-SRR1661324	UK-SRR3360619	UK-SRR2736229	UK-SRR2736183
UK-SRR2736219	US-ERR223644	UK-SRR3360650	US-ERR223619	UK-SRR2736172	UK-SRR2736112
UK-SRR2736216	UK-SRR3361315	UK-SRR1661281	UK-SRR2736207	UK-SRR2736283	UK-SRR3361313
US-ERR223649	US-ERR191767	UK-SRR3360836	UK-SRR3357314	UK-SRR3349688	UK-SRR3360846
US-ERR191782	US-ERR191810	US-ERR191794	UK-SRR1661327	UK-SRR2736273	UK-SRR3360686
UK-SRR3360926	UK-SRR2736115	UK-SRR3360984	US-ERR191771	US-ERR191793	UK-SRR3360766
UK-SRR2736295	UK-SRR3360684	UK-SRR2736204	UK-SRR3361310	UK-SRR3360729	UK-SRR3360665
UK-SRR3361354	UK-SRR2736296	US-ERR191820	UK-SRR3360939	UK-SRR2736109	UK-SRR2736263
UK-SRR3360690	UK-SRR3360630	UK-SRR3360698	UK-SRR3357160	UK-SRR3360953	UK-SRR3360949
UK-SRR2736289	US-ERR223642	UK-SRR2736297	US-ERR191762	UK-SRR1661323	UK-SRR3360754
UK-SRR3360681	UK-SRR2736220	UK-SRR3349526	UK-SRR3360612	UK-SRR3357252	US-ERR191808
US-ERR191803	US-ERR223628	UK-SRR3360827	UK-SRR2736281	US-ERR223615	UK-SRR2736175
UK-SRR3360738	UK-SRR3360721	US-ERR191799	UK-SRR2736217	US-ERR191736	UK-SRR3361314
UK-SRR2736194	UK-SRR3361322	UK-SRR3360719	UK-SRR2736231	UK-SRR3360723	UK-SRR3349518
UK-SRR3360711	UK-SRR1661325	UK-SRR2736119	UK-SRR3360669	US-ERR191756	UK-SRR2736299
UK-SRR2736200	US-ERR191755	UK-SRR2736186	UK-SRR3360726	UK-SRR3360671	US-ERR223674
UK-SRR3360771	UK-SRR3360677	UK-SRR2736251	UK-SRR2736153	UK-SRR3349568	UK-SRR2736185
US-ERR191805	UK-SRR3360947	UK-SRR3361312	UK-SRR1661243	US-ERR191783	US-ERR223643
UK-SRR2736117	US-ERR223616	UK-SRR2736151	UK-SRR3360737	UK-SRR3360717	US-ERR191807
UK-SRR2736258	UK-SRR2736248	US-ERR223665	UK-SRR2736129	UK-SRR3357194	US-ERR191738
UK-SRR3349563	UK-SRR3361333	UK-SRR2736206	UK-SRR2736159	UK-SRR2736278	UK-SRR2736193
UK-SRR2736232	US-ERR191758	UK-SRR3361352	UK-SRR3361351	UK-SRR2736154	UK-SRR3360674
US-ERR223607	UK-SRR3360730	UK-SRR3360688	US-ERR223650	UK-SRR3360704	US-ERR191784
UK-SRR1661175	US-ERR191740	UK-SRR3349577	UK-SRR2736103	UK-SRR3349516	US-ERR223678
UK-SRR3360706	UK-SRR3349658	UK-SRR2736141	UK-SRR2736285	UK-SRR3360670	US-ERR223682
UK-SRR1661328	UK-SRR2736164	UK-SRR3360642	UK-SRR2736292	UK-SRR2736168	UK-SRR2736267
UK-SRR2736202	UK-SRR3349601	UK-SRR1661292	UK-SRR2736093	US-ERR223684	UK-SRR2736146
US-ERR191734	US-ERR191765	UK-SRR3360710	UK-SRR3361307	UK-SRR2736238	UK-SRR3360851
US-ERR191770	US-ERR191818	US-ERR191813	US-ERR223666	UK-SRR2736247	UK-SRR3360964
UK-SRR2736250	US-ERR223622	UK-SRR3357264	US-ERR223687	UK-SRR3349573	UK-SRR3360772
US-ERR191801	UK-SRR3360615	UK-SRR2736239	UK-SRR2736280	US-ERR191811	UK-SRR2736287
UK-SRR2736126	UK-SRR3360749	US-ERR191787	UK-SRR2736097	UK-SRR2736259	UK-SRR3360748
UK-SRR2736291	UK-SRR3360829	UK-SRR2736098	UK-SRR3360705	UK-SRR2736236	UK-SRR2736188
UK-SRR2736094	UK-SRR3360667	UK-SRR2736113	US-ERR223683	US-ERR223693	UK-SRR1661183
US-ERR191749	US-ERR223655	US-ERR223609	UK-SRR3360943	US-ERR191779	UK-SRR2736160

UK-SRR2736264	UK-SRR3360774	UK-SRR2736282	UK-SRR3357305	UK-SRR3357260	UK-SRR2736222
UK-SRR3349538	UK-SRR2736268	UK-SRR2736167	UK-SRR3360622	UK-SRR3360624	UK-SRR2736144
UK-SRR3357246	UK-SRR2736180	UK-SRR2736302	UK-SRR3360728	US-ERR223685	UK-SRR3360649
UK-SRR1661242	UK-SRR2736182	US-ERR191817	US-ERR191760	UK-SRR3360746	UK-SRR2736116
UK-SRR3360701	UK-SRR2736301	US-ERR223614	UK-SRR2736158	UK-SRR2736099	UK-SRR3360872
US-ERR191789	UK-SRR3361341	US-ERR191792	UK-SRR3357192	UK-SRR3360627	US-ERR223662
UK-SRR3360810	UK-SRR2736209	UK-SRR3360609	UK-SRR2736118	UK-SRR3361342	UK-SRR3360967
UK-SRR2736262	UK-SRR2736149	UK-SRR2736254	UK-SRR2736187	UK-SRR3360991	US-ERR223676
UK-SRR3357191	US-ERR223603	UK-SRR3361355	UK-SRR3361343	US-ERR191773	US-ERR223652
US-ERR191797	UK-SRR2736260	US-ERR223627	UK-SRR2736107	UK-SRR3360755	UK-SRR3361317
UK-SRR2736123	US-ERR223633	US-ERR191815	US-ERR223635	UK-SRR3357077	UK-SRR3349546
UK-SRR2736195	UK-SRR2736298	US-ERR191759	UK-SRR3360633	UK-SRR2736138	UK-SRR2736199
UK-SRR3360644	UK-SRR3361350	US-ERR223669	US-ERR191785	US-ERR223698	UK-SRR1661326
UK-SRR2736177	UK-SRR3360725	UK-SRR2736137	UK-SRR3361321	UK-SRR2736131	UK-SRR3360647
UK-SRR2736214					

Appendix D

RR-BLUP Results

antibiotic	rrblup RMSE
azithromycin	33.4
cefixime	0.16
ciprofloxacin	14.1
penicillin	11.9
tetracycline	13.3

Table D.1: RMSE values when using RR-BLUP for MIC prediction on the *Neisseria gonorrhoeae* dataset, the current state-of-the-art technology for phenotype prediction for GS studies.

Appendix E

Preprocess Script

```
# Download and validate
prefetch $(<accessions.txt) | vdb-validate ./ | fastq-dump \
    --split-files $(<accessions.txt)

# Align short reads to reference
for sra in `cat accessions.txt`
do
  bwa mem -R "@RG\tID:$sra\tSM:US/UK/CA-$sra" NCCP11945_NG.fasta \
    $sra"_1.fastq" $sra"_2.fastq" -o $sra".sam"


  # convert sam files to bam and sort
  samtools view -buh $sra".sam" | samtools sort -o $sra".sorted.bam"

  # create bam list
  echo $sra".sorted.bam" >> wgs676.bam.txt
done



# SNP calling with freebayes
freebayes --ploidy 1 --min-mapping-quality 30 --min-base-quality 30 \
    --min-alternate-fraction 0.75 --min-coverage 15 \
        -f NCCP11945_NG.fasta --bam-list wgs676.bam.txt > wgs676.vcf


# filter out low quality SNPs and indels
vcffilter -f "DP > 4" -f "QUAL > 20" wgs676.vcf > wgs676.filtered.vcf
vcffilter -f "TYPE = snp" wgs676.filtered.vcf > wgs676.snp.vcf
```

Listing E.1: `preprocess.sh`