Integrated Deep Learning and Bayesian Classification for Prioritization of Functional Genes in Next-Generation Sequencing Data

Chan Khai Ern, Edwin

A thesis submitted to the
Department of Biochemistry
National University of Singapore
in partial fulfilment for the
Degree of Bachelor of Science with
Honours in Life Sciences

Life Sciences Honours Cohort
AY2015/2016 S1

DECLARATION

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

Chan Khai Ern Edwin

04 April 2017

Acknowledgements

Table of Contents

Contents

Abstract

1 Introduction

1.1 Next Generation Sequencing in Personal Genomics Pipelines

Explain NGS a lot of datasets, Variant Calling general problem that people face, Bamfiles, fastq files (alignment calling) why is it a problem, large amount of sequence data, insert graph on variant calling problems. (MUST HIGHLIGHT THE PROBLEM, HIGHLIGHT THE TYPE OF OUTPUT)

Identification of functionally important mutations is a critical step in enabling personal genomic pipelines. Recently, there has been great interest in using a persons genome to help doctors treat and diagnose disease (Rehm, 2017;Angrist, 2016). The fundamental intuition is that sequencing the person's genome can help doctors and clinicians narrow down important disease subtypes and progression. This enables doctors to better diagnose the disease, as well as prepare targeted medication to treat the specific disease.

INSERT

1.2 Main Hurdles to Implementation

However, there are still two critical steps in this pipeline that needs to be addressed. Firstly, we need to be able to obtain high confidence mutations, and secondly there needs to be a method to prioritise these mutations for clinicians and doctors. For the first problem we have to be able to find out mutations in the persons genome that is different from a reference genome. This step is termed in literature as variant calling as it involves the discovery (calling) of genes (variants) that differ from a reference genome. However, current variant callers still tend to have low concordances for variants called (O'Rawe et al., 2013; Cornish and Guda, 2015), primarily due to differences in variant calling algorithms and assumptions. The second problem, ranking of mutations is critical as these variant callers do not take into account the importance of each variants. Thus, an additional step must be taken to attempt to find out which might

be the most important genes for clinicians. This is critical a clinician should be able to obtain variants that are of clinical significance without having to sieve through literature to manually pick out genes that important. This would allow them to narrow their search to the most likely candidate dates, and then be able to embark on the most ideal treatment pathway.

In this paper, we describe a machine learning approach to solve these two problems - specifically using deep learning to validate high confidence variant calls, and using Bayesian networks to filter variants and prioritise their importance.

1.3 Variant Calling in Personal Genomics Pipelines

INSERT

1.4 Deep Learning in Variant Calling Methodology

One large hurdle to overcome in enabling personal genomic pipelines is the generation of high quality variant calls. Variant calling primarily involves the use of various statistical and mathematical methods to discover variants, or mutations, in the genome. Calling variants allows the analysis of deviations and differences between the genome of interest and a standard human genome. However, there are still areas for improvement in current variant calling methods, including dealing with different classes of mutations, as well as reducing the number of false positives (Mohiyuddin, et al., 2015; Gzsi et al., 2015).

Both these problems fundamentally result from assumptions and implementations of variant callers - certain algorithms are more sensitive and accurate in calling certain classes of mutations, but suffer from inaccuracies in calling other variant types and edge cases. Probabilistic haplotype generating callers (such as GATK's haplotype caller and FreeBayes) tend to be more accurate for SNPs and indels (McKenna et al. 2010; Garrison & Marth, 2012). They perform de-novo local assembly, where they rebuild small portions of the genome, and subsequently use bayesian analysis to determine the existence of variants. Specifically, they generate short hap-

lotypes of local regions from sampled sequences, and determine (based on the haplotypes and prior probabilities in the reference genome) whether a variant should be called. However, these methods can only handle limited window sizes, preventing the detection of larger structural variants. For these we have to rely on other tools that examine larger segments of the genome (Ning et al., 2009) or use libraries of known mutation regions, and study these breakpoints to check if any mutations have occurred (Gerstein et al., 2015). Due to the heterogenity in mutations, no single caller works best for all classes of mutations, pointing towards a variant calling framework that aggregates data from multiple callers.

Indeed, studies have shown low concordances between variant callers themselves, due to their specific implementations and algorithms (Mohiyuddin, et al., 2015; Gzsi et al., 2015). If we consider that each variant caller samples from the same genome but with a different statistical technique, then we can see each variant caller as a mode of data that provides us with a unique piece of information on the genome. Thus, we can generate more accurate calls by aggregating the multi-modal data from various callers, allowing us to cross validate the variants called using multiple techniques.

The simplest approach to aggregate data is concordance - if multiple variant callers are able to call a variant, it is most likely to be accurate. However, the recall of such a tool would be poor due to the differential sensitivity of callers to edge cases, resulting in a lot of false negatives as true variant calls might only be picked up by one or two calling methodologies. This would defeat the purpose of using multiple callers in the first place, as the strength of a combinatorial approach lies in tapping into the sensitivities of different callers. More sophisticated efforts have since been done to use machine learning methods such as Support Vector Machines as a way to integrate variant calling information (Gzsi et al., 2015), and the authors showed that SVMs presented an improvement over concordance based methods. However, with the advent of deep learning techniques and libraries, which have been shown be able to integrate complex multi-modal information to solve problems (Ng et al., 2015), we hypothesize that deep learning can also be used to integrate the information from variant callers.

Deep learning is a method of machine learning that involves deep stacks of artificial neural networks. These neural networks were inspired by the way our synapses work in the brain, and are represented in silico by input/output nodes that fire when a certain threshold is reached. Thus, these neural networks are able to simulate learning - by learning from labelled data correlations between inputs and outputs, these networks are able to predict outputs if given a new input.

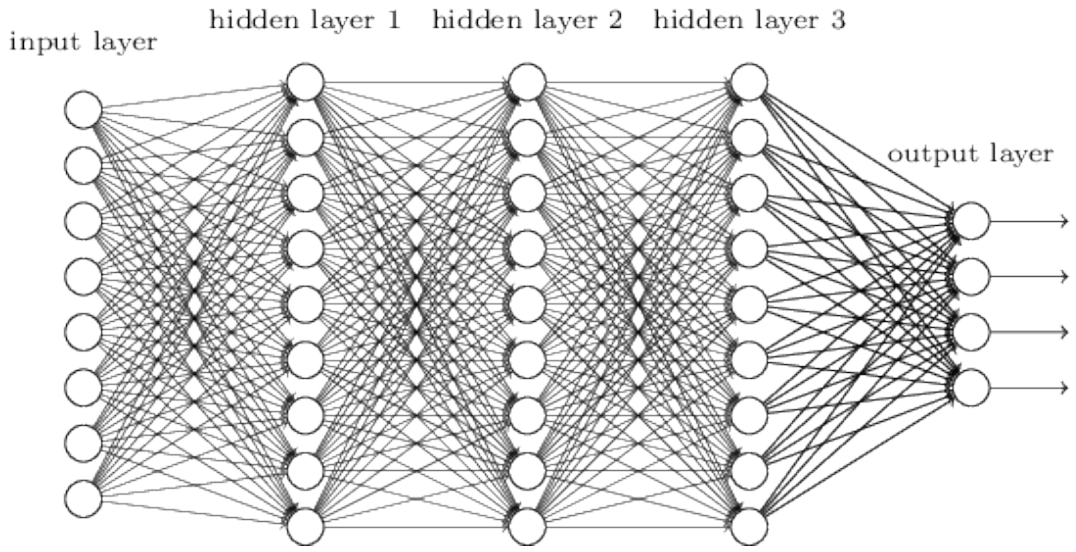

Figure 1: A Neural Network with 1 input layer, 3 hidden layers and 1 output layer. This represents a densely connected neural network, where each node is connected to every node of the preceding and subsequent layers. At each node, linking functions can be defined to apply a mathematical transform to connect the input and output.

Figure 1 depicts a sample neural network with 5 layers, with 8 data points as input, and 3 data points as output. In such a network, we would train it by providing the input data and output data, and letting the network learn how to integrate the inputs to create a network of activations that can be used to produce the corresponding output. In variant calling, deep learning will allow us to predict based on variant calling features and data whether a variant is valid and exists, or is erroneous. This will allow us to draw of the diversity of data with different variant callers, through letting the network learn which patterns will result in a valid call and which patterns are actually false positives. It will also allow us to tap on the differential sensitivity of different callers, as the neural network is able to learn which callers work best for

which types of mutations. Thus, such a combinatorial approach will allow us to improve the accuracy and precision of variant calling.

1.5 Bayesian Networks in Gene Prioritisation

The second problem of enabling personal genomic pipelines is gene prioritisation. The problem of gene prioritisation arises because there are a multitude of data sources we can draw on to analyse how important a gene is. The possible approaches include studying previously characterised variants and their phenotypic effects on a person, studying how the mutation itself will affect protein function through studying the likelihood of amino acid mutation for conserved regions and so on. These functional annotations can be done with the tool ANNOVAR, but the fundamental problem here is integrating such information in a systematic manner that is clear and understandable to clinicians. Clinicians may not be so familiar with the tools and functional annotation pipelines, but yet in order for them to trust such a ranking system they have to be able to intuitively understand how it works. As such, to solve this problem, we use Bayesian networks to integrate the information from functional annotations as well as the confidence of a call (how likely it is real) to provide a ranking system for how likely the gene is going to be important. Bayesian networks were chosen for this ranking system as it is understandable and yet have proven stable in terms of solving decision making problems (Pourret et al.,2008;Jensen et al, 1996). Bayesian networks have been applied in medical treatment decision making (Windecker et al., 2014), ecological studies (Johnson et al., 2014) and even predictive epidemiology (Su et al., 2014). A Bayesian network is a network that records the probabilities of events, and based on conditional probabilites and observations it updates the final likelihood of an event. This can be seen in Figure 2.

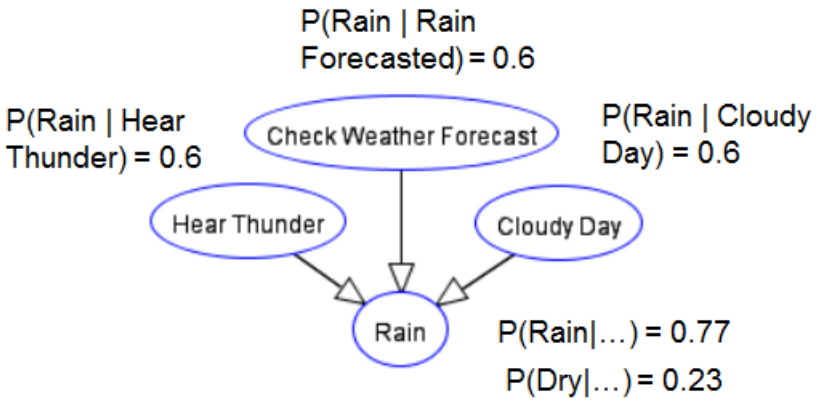

Figure 2: A Sample Bayesian Network for Rain Prediction.

Here, we would like to predict how likely it is to rain. Thus, we record observations (we hear thunder, check the weather forecast, notice it is a cloudy day) and updating the likelihood of rain happening based on the conditional probabilities of P(Rain — Hear Thunder)... and so on. This model of learning was chosen because a Bayesian Network closely mimics the way Humans think - we observe events and form co-relational and causative predictions based on those events. This is advantageous over deep learning as in deep learning we are unable to interrogate the system to intuitively understand what the network is learning form - it has high predictive power but is essentially a black-box. The Bayesian network allows clinicians and doctors to see what are the components that went into gene ranking and prioritisation, and even change the probabilities, weightage or add more observation nodes based on their own diagnosis and treatment models and ideas. Ultimately, this enables the doctors and clinicians to be able to have confidence in the software as they are able to analyse and understand how it works. This also allows them to be able to explain their methodology and treatment plans clearly to the patient, making the diagnosis and treatment process clear, understandable and transparent.

1.6 Aims and Research Structure

In this paper, we describe the deep learning to validate high confidence variant calls, and using Bayesian networks to filter variants and prioritise their importance. Specifically, we will

validate the use of deep learning to validate true variants in both real and simulated datasets. Subsequently, we will build a Bayesian network based on functional annotations to prioritise mutations and test our network on a real patient's cancer genome.

2 Materials and Methods

2.1 Overall Analysis Structure

For our research, we describe two main pipelines - the first pipeline will be used for training and optimisation of a neural network, and the second pipeline uses a trained neural network to perform variant validation and prediction. The first pipeline involves using a training dataset, performing alignment, variant calling and deep learning network training and prediction on the dataset. This allows us to generate a set of predictions which we can validate against the ground truth. The second analysis pipeline is meant for real datasets with no ground truth. For this pipeline, we will perform alignment and variant calling, and then filter it with a pre-trained network. Finally, we apply the Bayesian network analysis to predict genes in this dataset.

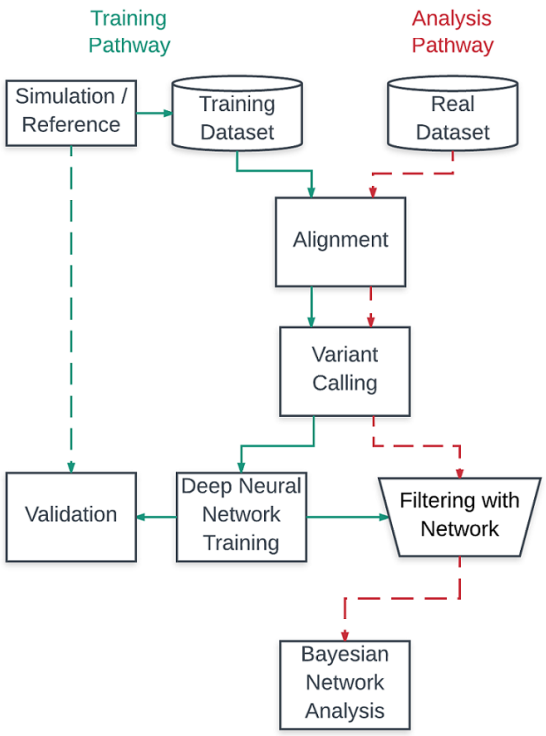

Figure 3: Overall Analytical Pipelines - Pipelines were implemented using the Groovy Domain Specific Language, NextFlow

2.2 Artificial Datasets

Artificial genomes present a simple way to simulate NGS data with perfectly known ground truth variants to test our neural network. For our simulator, we used Mason, a genome mutation software written in C++ (SeqAn version 2.3.1) to mutate the hg19 reference genome from UCSC (Karolchik et al., 2014). We used indel rates of 0.00002 and SNP rates of 0.00008 to generate sufficient truth variants for analysis, which comprise 229253 SNPs and 57257 indels.

After generating a ground truth model, we simulated sequence reads with error rates and ground truth variants(Figure 5). For error rates, we used published data from Schirmer et al. (2016) as the input to Mason - the general substitution error rate used was 0.0004 per base in the genome, and the insertion and deletion error rate per base was $5 * 10^{-6}$.

2.3 Alignment and Variant Calling

To perform alignment of simulated and real sequences, the Burrows Wheeler Aligner (Li, 2013), version 0.7.13, was used. Default settings were used, with the mem option which is known to work well with longer sequences. After alignment, variant calling was performed. Variant callers used were FreeBayes (v1.0.2-16), GATK Haplotype Caller (v3.7-0) and Unified Genotyper (v3.7-0), Samtools (v1.3.1) and finally Pindel (v2.3.0)(Garrison & Marth, 2012; McKenna et al. 2010, DePristo et al. 2011; Li H, et al., 2009; Ye et al., 2009). All callers were used at their default settings.

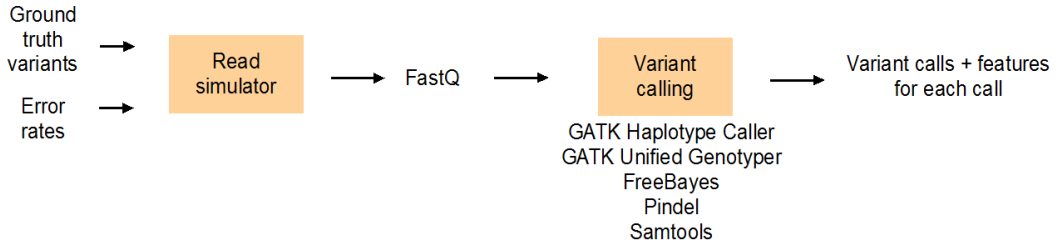

Figure 4: Pipeline for simulation of artificial genome for analysis

2.4 Feature Engineering

In order to train a neural network, features in the form of numerical vectors must be used an input. We subset our features into three broad sets, which are base-specific information, sequencing error and bias information features, and calling and mapping quality. Here we describe the computation of the features - please see Appendix 5.2 for an more in-depth explanation on their usage and interpretation.

Base Information

Shannon Entropy

Shannon Entropy captures the amount of information contained inside the allele sequences. It is calculated using the equation :

$$H(X) = -\sum_{i=1}^{n} P(x_i) \log_2 P(x_i) \tag{1}$$

where $P(x_i)$ is the prior probability of finding each base at each position. This prior probability is calculated in two ways - over the entire genome, over a region of space around the allele (10 bases plus the length of the allele in our calculations).

Kullback Leibler Divergence

The Kullback-Leibler Divergence feature is similar to Shannon entropy, but instead we use this to measure the informational change converting from the reference to the allele sequence. The Kullback-Leibler Divergence is calculated as follows :

$$D_{KL}(P||Q) = -\sum_{i=1}^{n} P(x_i) \log_2 \frac{P(x_i)}{Q(x_i)} \tag{2}$$

where $Q(x_i)$ is the prior probability of finding each base at each position based on the genomic region around the allele, while $P(X_i)$ is the posterior probability of finding a specific base inside the allelic sequence.

Base Quality

Base quality refers to the Phred score probability that the called allele is wrong. It is given by

the equation :

$$P = 10^{\frac{-Q}{10}}$$

Where P is the Base Quality and Q is the probability that the allele called is wrong. This is a number computed by the sequencing machine based on the quality of the base samples provided.

Sequencing Biases and Errors

GC content

This feature comprises the percentage GC content of reference genome for at least 10 bases around the mutation site.

Longest homozygous run

This feature comprises the longest similar string of bases in the reference genome, for at least 10 bases around the mutation site.

Allele Count and Allele Balance

This feature is an output from Haplotype Caller and Unified Genotyper, an describes the total number of alleles contributing to a call and the balance between reference and alternate alleles reads.

Calling and Mapping Qualities

Genotype Likelihood

The genotype likelihood provides the phred-scaled likelihood scores of how confident the caller is in determining that it is a homozygous or heterozygous call, and is provided by all variant callers.

Read Depth

Mapped read depth refers to the total number of bases sequenced and aligned at a given reference base position. It is provided by all variant callers.

Quality by Depth

Quality by Depth is computed by dividing the quality score against allele depth, to obtain an

average score of allele quality. This is provided by Haplotype Caller and Unified Genotyper.

Mapping Quality

Mapping quality is a score provided by the alignment method and gives the probability that a read is placed accurately. It is provided by all variant callers except Pindel.

2.5 General Programming, Pipelining and Mathematical Computations

The general programming platform used was Python (v2.7). Python was chosen due to its access to various important libraries, including NumPy, SciPy, Pomegrenate and PyVCF. NumPy was used to prepare input vectors for deep learning training, SciPy was used to perform Principal Component Analysis and Synthetic Minority Oversampling Technique Methods (See Appendix 5.3 for more information). Finally, PyVCF was used to parse the VCF files into python objects for easy manipulation. Comparison of VCF file entries, which are each individuals mutations, was also performed using python dictionary lookups. This method was chosen due to a high number of lookups required, and a fast O(1) constant time required for each lookup.

General pipelining and chaining of programmes was done using NextFlow and Bash scripts. NextFlow is a Groovy Based Domain Specific Language (DSL) that provides easy writing of parallel pipelines with an accessible unix interface. Nextflow was used to run the overall pipelines and control input and output of abstracted core modules, which are in turn either python scripts or Bash shell scripts. This ensures that results are easily replicable and can be later implemented as a single analytic pipeline for clinical use.

2.6 Deep Learning and Bayesian Networks

For our deep learning networks, we used the Keras library (v1.1.1) with a TensorFlow backend (v0.11.0). TensorFlow (Abadi et al., 2015) was chosen due to its distributed computation and queue management system that enabled better performance in training on a CentOS-7 compute cluster compared to other backends. For more explanation on the algorithms underpinning deep learning, see Appendix 5.1 for more information.

Pomegranate (a Python library) was used to generate and compute the probabilistic model and ranking system for our Bayesian Network. The preparation of the probabilistic model and the truth-tables required was done using Python scripts, and subsequently used as input in pomegranate.

2.7 Patient Derived Xenograft Mouse Model Development and Sequencing

[TO CLARIFY] To test the Bayesian ranking system, we used a patient derived xenograft mouse model. Athymic mice with the FOX mutation were grown for X number of days, and subsequently a tumour is grafted onto the mouse's body. Subsequently, the tumour was sequeneced on an Illumina MiSeq platform and used for analysis.

3 Results and Discussion

3.1 Generation of Artificial Datasets

Using a genome mutation software, we generation a mutated genome (using the hg19 genome from ucsc (Karolchik et al., 2014) as a reference) with over 300,000 random mutations spread over the chromosomes as can be seen below in Figures 3 and 4.

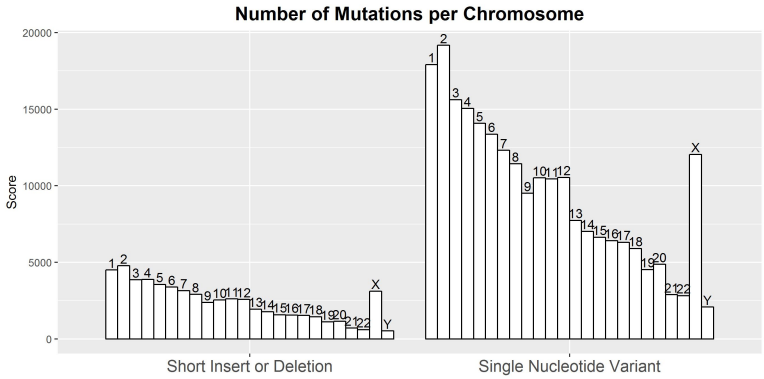

Figure 5: Number of ground truth mutations (variants) created in each chromosome

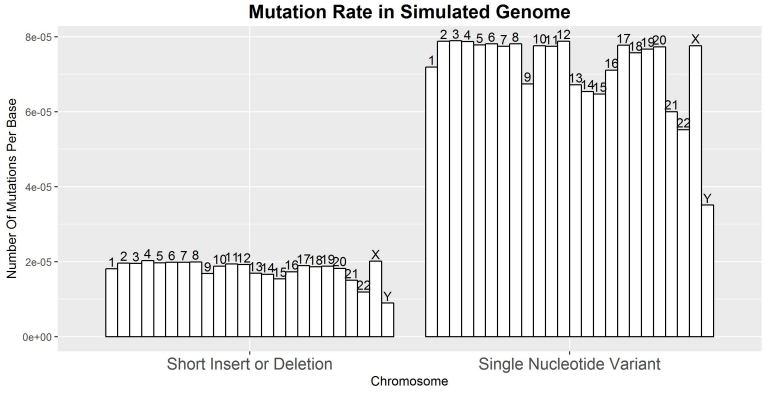

Figure 6: Mutation rate per base in each chromosome

Artificial genomes are a good method to analyse deep learning networks on as the ground truth, which are the truth variants inside the genome, are already known. This allows accurate verification of prediction schemes and is a commonly used method to test next generation sequencing related software (Escalona, Rocha & Posada, 2016). This is primarily because it

is difficult to obtain complete truth datasets for real genomes as due to the inhibitory cost of checking every variant called via Sanger sequencing. Thus, artificial genomes present a simple way to simulate NGS data with perfectly known ground truth variants to test our validation platform.

3.2 Feature Engineering

Subsequently, we engineered a set of 19 features to use as input data for our variant callers, using data obtained from the variant callers themselves as well as engineering other features from the dataset. A summary of the features used in the training can be found in Table 1 and a description of the full list of features can be found in Appendix 5.2. Features were engineered based on obtaining information for the main aspects of variant calling, which includes the information contained in the sample bases (Base Quality, Entropy, KullbackLeibler divergence etc), the confidence we have in the calling and alignment (Read Depth, Mapping Quality etc) and finally possible biases in the sequencing machine(Allele Balance, Allele Count, GC content).

Table 1: Feature Engineering Table

Features	Shannon Entropy (Reference, Alternate and KL- Divergence)	Base Composition (Homopolymer Run, GC content)	Read Depth	Mapping Quality	Base Quality	Allele Balance	Quality by Depth	Allele Count	Genotype Likelihoods
Free Bayes	+	+	+	+	+	+			+
Haplotype Caller	+	+	+	+	+		+	+	+
Unified Genotyper	+	+	+	+	+	+	+	+	+
Pindel	+	+	+						+
Samtools	+	+	+	+	+	+			+

3.3 Variant Callers

Variant callers were chosen for our deep learning neural network based on their orthogonal calling and reference methodologies - we wanted to maximise the range of variant callers in order to optimise the information that the neural network receives (See Table 2). We used two haplotype based callers, FreeBayes (Garrison & Marth, 2012) and GATK Haplotype Caller (McKenna et al. 2010, DePristo et al. 2011), two position based callers GATK unified Genotyper and Samtools (Li H, et al., 2009) and finally Pindel, a pattern growth based caller (Ye et

al., 2009). All callers have been well studied and are commonly used in variant calling pipelines (Sandmann et al., 2017, Hwang et al., 2015 Xie et al., 2014 and Liu et al., 2013).

Table 2: Table Comparing Methods and Features of Different variant callers.

	GATK Unified Genotyper	**Samtools**	**GATK Haplotype Caller**	**Free Bayes**	**Pindel**
Calling Method	Uses a list of mapped reads, calling model is probabilistic with increased priors at regions with known SNPs	Uses a list of mapped reads, calling model is probabilistic. Does not assume sequencing errors are independent and has less hard filters compared to Unified Genotyper	Uses Hidden Markov Models to build a likelihood of haplotypes which are then used to call variants	Uses a posteriori probability model to build a set of haplotypes to represent mutations, calling model is probabilistic with population based priors	Locates regions which were mapped with indels or only one end was mapped, and then performs a pattern growth to find inserts and deletions. Shown to be able to identify medium length indels missed by other callers in real samples (Spencer et al., 2013)
Reference and Mapping Method	Position based caller that realigns fragments and analyses each position to call SNPs and indels	Position based caller that uses mapped sequences to call SNPs and indels.	Analyses regions where there is high likelihood of mutation based on activity score, and builds a De Bruijn-like graph that reassembles reads (Haplotypes) in that region	Dynamic sliding window based reference frame, using algorithms to determine window size for analysis. Does not require precise alignment, unlike other callers	Focuses on Unmapped regions, regions known to have insert and deletions or regions with only one end mapped.

INSERT CONCORDANCE GRAPHS

3.4 Network Architecture

Before training our deep learning network, we tested out various neural network architectures to see which architecture would perform the best for our set of input features.

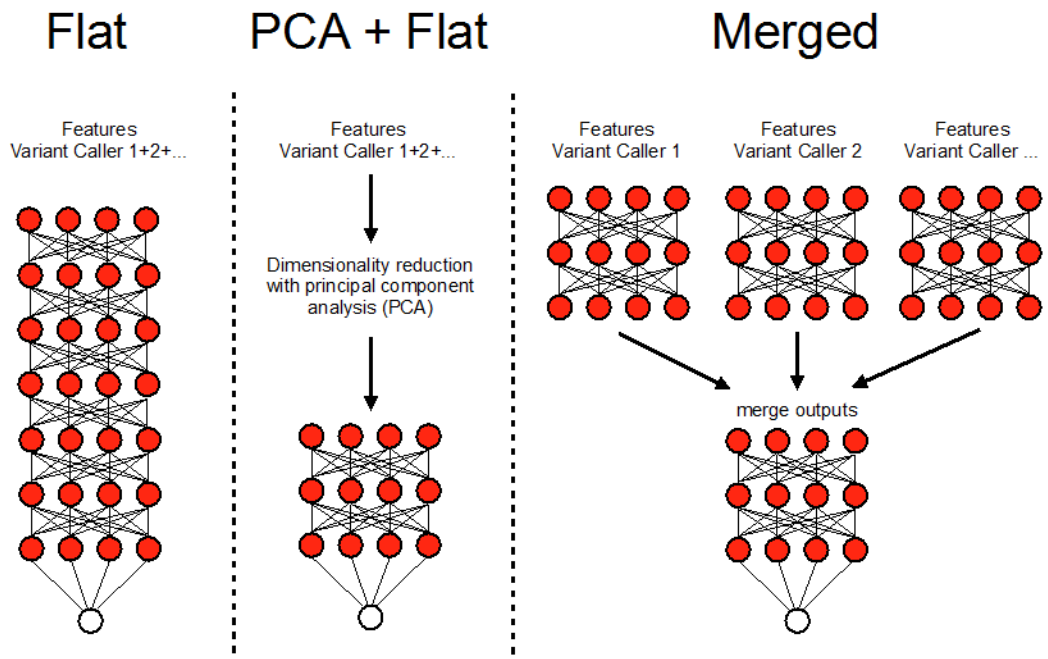

Figure 7: Different Designs for Neural Network Architecture

We first explored the flat architecture (Figure N), which contains stacks of fully connected layers with multiple nodes (initially 7 layers, with 80 nodes per layer). This is the simplest architecture, where all the features are loaded onto a single vector, and this entire vector is used as an input to train the neural network. We next explored the PCA + flat architecture which had the same neural network architecture but before the input data was fed into the network, a Principal Components Analysis was done to reduce the dataset to 8 principal components which was then used as input data for the neural network (please see Appendix 5.3 for more details of the PCA analysis). Principal components analysis is a dimensionality reduction technique that enables a compressed representation of data. Each principal component is an linear summation

of the original function in the form

$$PC_1 = \beta_{1,1} * X_1 + \beta_{2,1} X_2 + ... + \beta_{n,1} X_n$$

...

...

$$PC_i = \beta_{1,i} * X_1 + \beta_{2,i} X_2 + ... + \beta_{n,i} X_n$$

which enables a few principal components to capture a high amount of variance in the dataset. Finally, the last architecture we tested was the merged network this network had a set of layers (initially 5 layers, 24 nodes per layer) that learns from each caller alone, and then the outputs from each of these layers are subsequently merged and used to make a prediction.

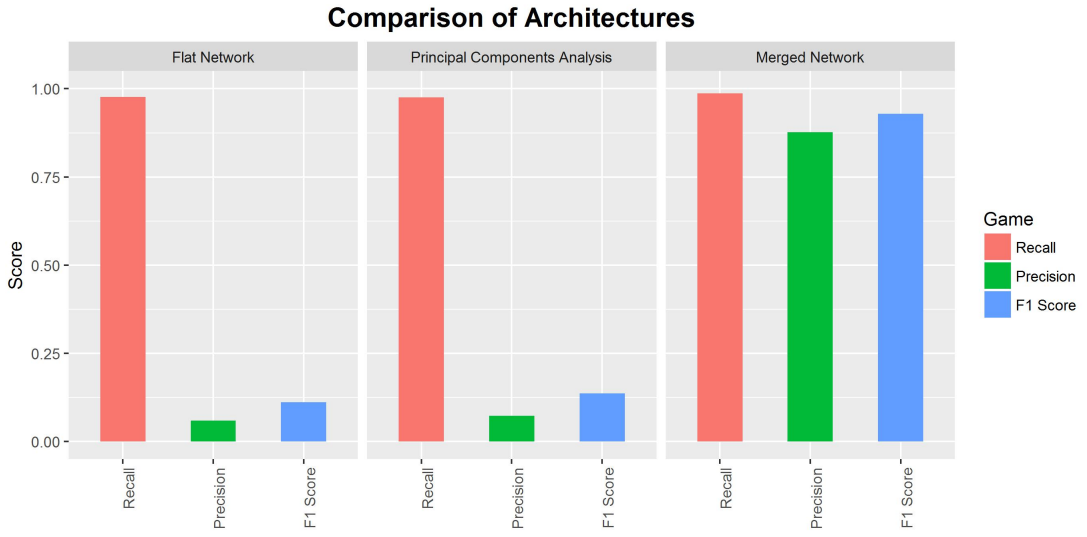

Figure 8: Analysis of Different Neural Network Architecture

Initially, with the flat network, the precision rate was very low at 0.059 with an F1 score of 0.112, indicating that the neural network was unable to learn from the input feature set. We suspected that this was due to high dimensionality in the dataset, which led to our second architecture design, the PCA with flat analysis. Principal components analysis has been shown to be able to successfully improve learning in high dimensionality datasets (Chen et al., 2014; Van Der Maaten, Postma & Van den Herik, 2009). However, the precision and F1 score for the

PCA architecture was also low at 0.0735 and 0.137. Ultimately both failed to learn, indicating to us that perhaps the features from each of the callers had to be analysed separately before being passed into a separate neural network that did the final score computations. With this merged network, we managed to obtain a precision and F1 score (0.877 and 0.929) that was far better than the previous two architectures. Interestingly, the recall scores for all three architectures were around the same (± 0.01), indicating the main learning point for the neural network was removing false positive calls.

3.5 Network Tuning and Optimisation

Next, we systematically optimised and tuned the deep learning neural network to maximise its predictive ability. In tuning our network, we also sought to study how the various hyper-parameters as well as the datastructure affected our network's ability to learn from the data. In particular, we focused on four issues - the number of layers, optimiser choice, learning rate choice and finally sample balancing. These four issues are known to be critical in deep learning networks (Ruder et al., 2016; LeCun, Bengio & Hinton, 2015; Yan et al., 2015; Sutskever et al., 2013) and would likely be critical to the success of a neural network.

A. Number of Layers

Firstly, we studied how many layers should be in the neural network. The number of layers is critical as it determines what kind of information and the representation of data that can be captured by the neural network. Choosing the number of layers is important as sufficient layers are needed to obtain the complex data representation needed for learning, but too many layers might result in the vanishing gradient problem. We started off with a neural network architecture as shown below, and began to vary the number of layers in at each point.

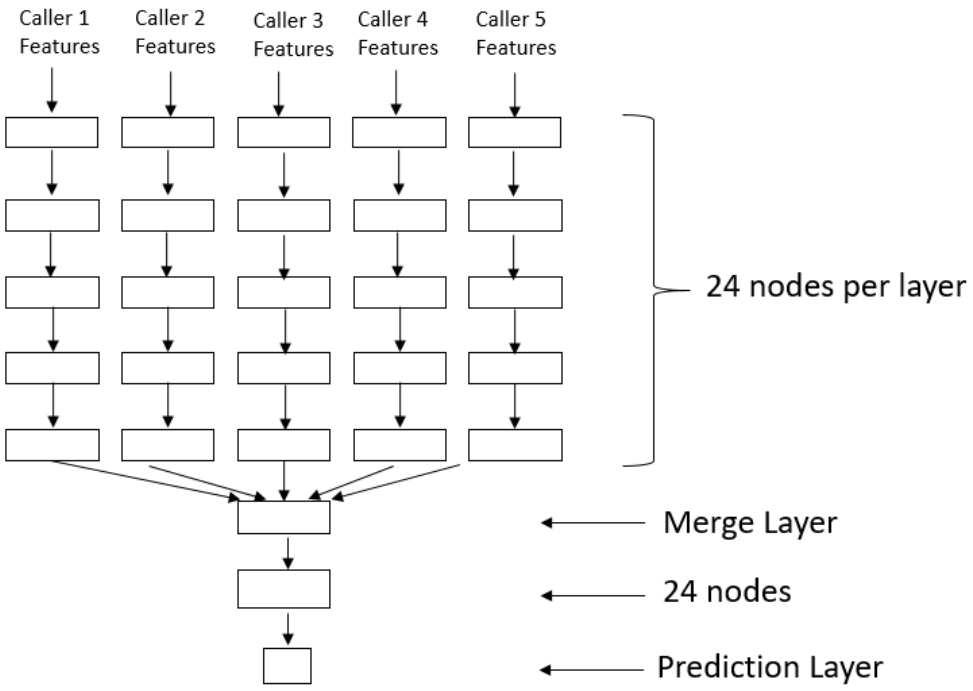

Figure 9: Basic Merge Network Structure

For all layers, the LeakyReLU activation function was used. The LeakyReLU is a refinement of the ReLU activation function, and both are well documented activation functions that have been shown to work well in deep neural networks(LeCun, Bengio & Hinton, 2015; Maas, Hannun & Ng, 2013). We noticed that changing the number of layers after the merge layer did not significantly vary the output, and so we focused on changing the number of layers before the merge layer. We studied 6 different neural network structures (4 layers to 9 layers).

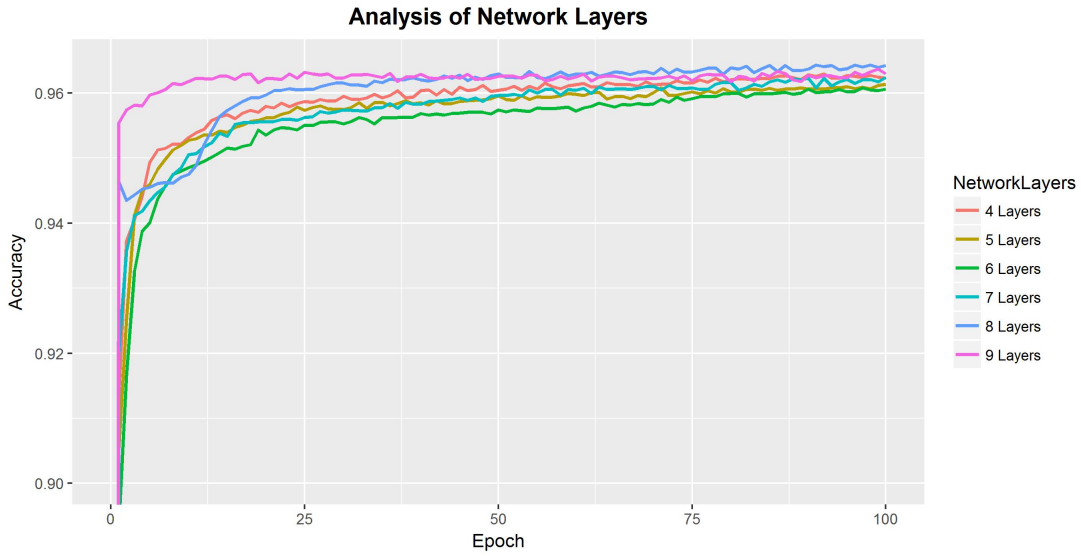

Figure 10: Analysis of Different Number of Layers On Training Accuracy

From analysing the accuracies of the different layers, we find that 8 layers seem the best at learning from the input data. We note that all the layers follow the same rough trend of accuracy, indicating they are all able to learn from the dataset. We decided on the 8 layer network as it seemed best able to learn from the input dataset, with a final accuracy of 0.964 that is about 0.001 higher than other layers. A final design feature used was to add two dropout filters at the last two layers before merging in order to prevent overfitting in data. Dropout filters have been shown to be an effective in preventing overfitting of data (Srivastava et al., 2014). Another active step taken to prevent overfitting was to ensure random separation of test, validation and training datasets.

B. Optimiser and Learning Rates

Next, we sought to choose the best optimiser and learning rate for our dataset. Both optimisers and learning rates have been well studied and known to be important in neural network training (Ruder et al., 2016; Sutskever et al., 2013). Optimiser choice is critical as the optimisers determine how the weights and gradients are updated in the network, thus playing an integral part in learning. We studied 3 well-known optimisers for use in our network, ADAM, RMSprop

and Stocastic Gradient Descent (SGD). ADAM is an adaptive learning rate optimiser that is known to be well suited in large dataset and parameter problems(Kingma & Ba, 2014). RMSprop is another adaptive learning rate optimiser that unpublished, but has been shown to work well for real experimental datasets (Tieleman & Hinton, 2012). SGD is the simplest learning model with no adaptive learning rate, but is a useful model because it is the easiest to understand mathematically and has also been shown to solve deep learning problems (Kingma & Ba, 2014). For more information on the mathematical foundations of backpropagation, please see Appendix 5.1. For the three optimisers, we ran tests to study the accuracy of the neural network running on each optimiser to predict true variants (Figure 11).

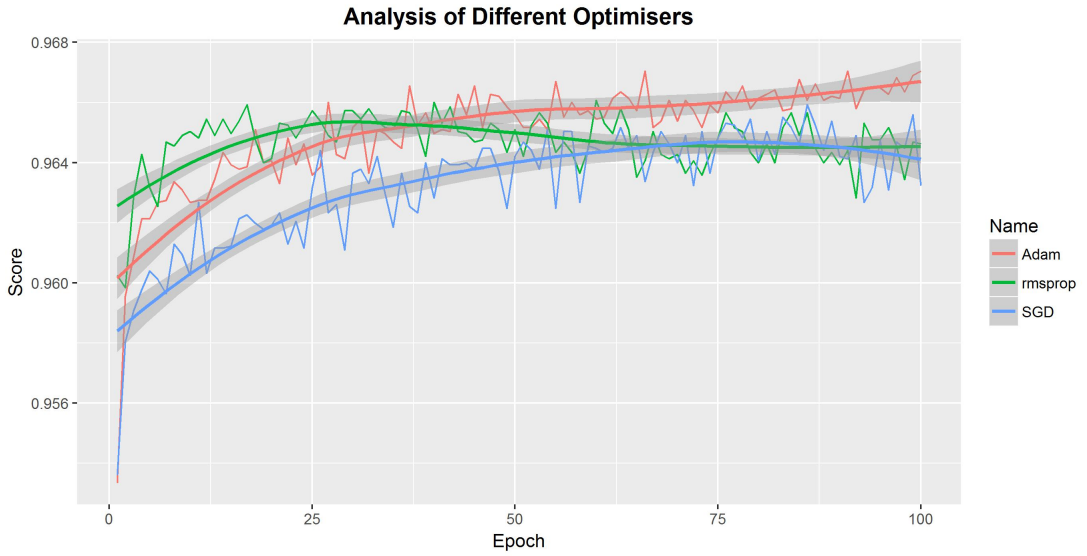

Figure 11: Optimiser accuracies for training at each epoch. Due to the noise in accuracies, the overall momentum of the dataset, calculated as a sliding window average is shown. The 95% confidence interval is also shown.

The adaptive rate optimisers, Adam and RMSprop obtained the highest accuracies of 0.9670 and 0.9660 while SGD reached a maximum accuracy of 0.9569, very close to that of RMSprop. Interestingly adaptive rate optimisers seemed to have complex learning trajectories, while SGD has a very stable learning rate. This makes sense as adaptive learning rates allow greater gradient descents when the error is high, and decreasing the learning rate at smaller errors. This allows Adam and RMSprop to learn at variable rates based on the current gradients. For SGD, it appears that while it takes a while to learn the true minima, it eventually still reaches

about the same minima as RMSprop. In the end, we chose Adam as our optimiser as the final accuracy discovered by Adam was noted to be higher than RMSprop and SGD, and we note a stable learning curve for Adam, indicating it is able to learn and update the gradients in the neural network to learn from input data. Then, we also looked at various initial learning rate for Adam (Figure 12), and found that the most stable learning could be found at a learning rate of 10^{-5}. This initial learning rate is critical as it determines the first few gradient descents which enable stable adaptive learning throughout the epochs (Sutskever et al., 2013). At any larger learning rates (10^{-4} and below), a very high amount of noise was observed, indicating that the learning rate was too high resulting in minima finding errors. At smaller learning rates (10^{-6} and above), the final accuracy after 100 epochs (0.9639) was lower than the learning rate at 10^{-5} (0.9672). Thus, we chose 10^{-5} to be our learning rate.

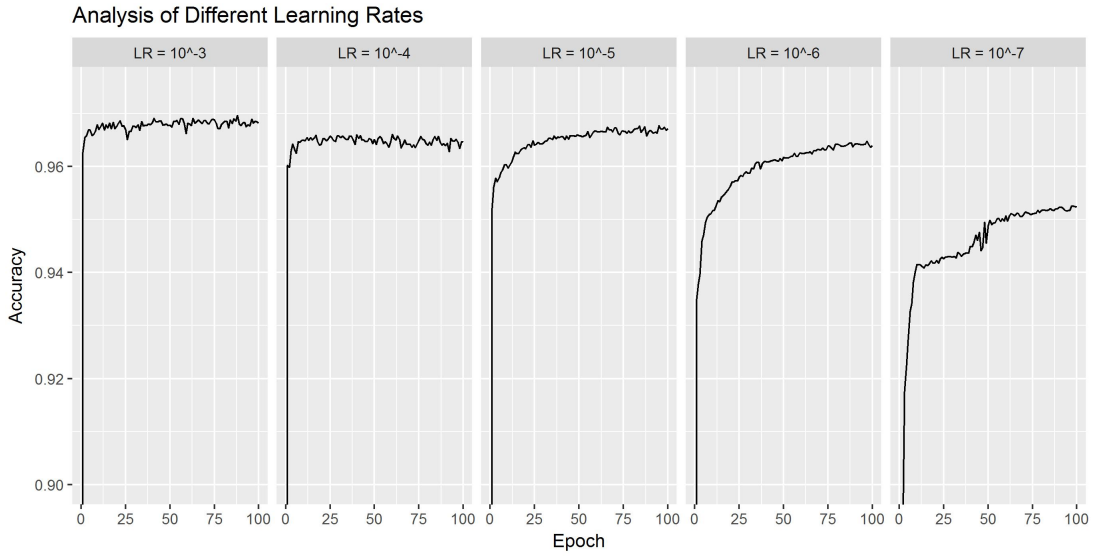

Figure 12: Training Accuracies over Each Epoch for Different Learning Rates

C. Sample Balancing

Our final concern was sample balancing - the simulated dataset contained an imbalance of positive training examples versus negative training examples. In total, there were 286569 positive training examples and 4547919 negative training examples. Such an sample imbalance has been known to affect learning adversely (Yan et al., 2015; Lpez et al., 2012). Thus, we sought to study two methods of sample balancing, undersampling and oversampling. Our data

was skewed with a high amount of negative training samples and a low number of positive train samples. Thus, undersampling was implemented by removing negative training examples until the number of negative training examples was equal to the number of positive training examples. In oversampling, the Synthetic Minority Oversampling Technique(SMOTE) was done, which uses nearest neighbours to create more data points for the positive training example (see Appendix 5.3 for more details). Figure 13 shows the metrics for each sampling technique.

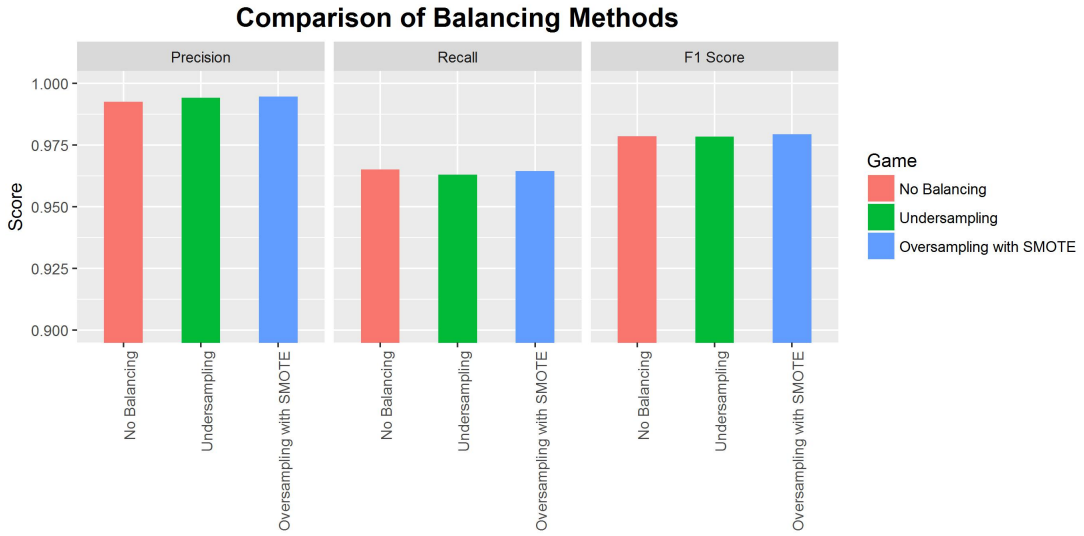

Figure 13: Effect of Sample Balancing on Prediction Ability

Interestingly, we note that overall, undersampling, oversampling and no sampling at all had very small effects on precision, recall and the final F1 score. This could be due to clear boundary separation within positive and negative class examples as well as good representative datapoints within the positive training example class. This prevents the imbalanced data from having too much of an effect on variant prediction and classification. Still, we note that oversampling techniques resulted in a marginally higher F1 score (0.001 higher than undersampling and no sampling), and since ensuring that datasets are balanced is a recommended protocol to prevent further bias downstream, we used SMOTE oversampling to produce extra positive training class examples for all analysis pipelines.

3.6 Benchmarking of Optimised Network with Mason Datasets

From optimisation steps, we finalised the network architecture as seen in Figure 9, but with 8 layers before the merge layer. We chose the learning rate to be 10^{-5}, and the optimiser used was Adam. With this network, we benchmarked the neural network against the single variant callers, as well as concordance callers, which are an integration of the outputs of the 5 variant callers. Specifically, the n-concordance variant callers are defined as the set of calls that any n callers agree upon - so 1-concordance includes all the calls made by all callers and 4-concordance includes all the calls made by any 4 callers. The metrics we use to determine how well the neural networks perform is via precision, recall and F1 score (their derivation can be found in appendix 5.3).

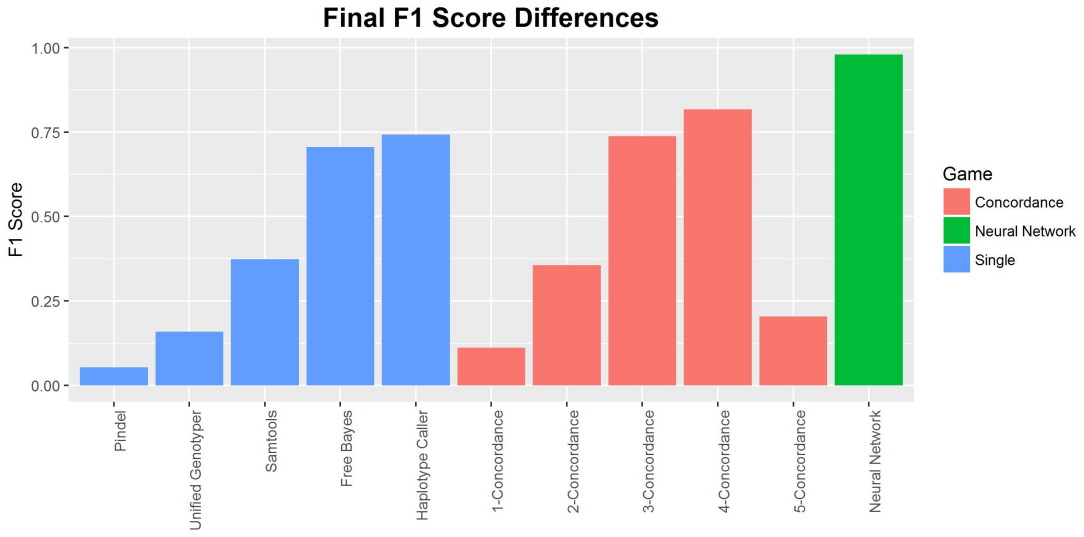

Figure 14: Overall Comparison of Variant Callers

In terms of overall F1 score, we see that the neural network was able to outperform single and concordance-based callers. This provides strong evidence that the neural network is able to learn from the input features whether the variant call is real or not, validating its usage in variant calling. The final F1 score obtained by the best single variant caller was Haplotype caller at 0.742, the best concordance caller had an F1 score of 0.816 while the neural network achieved an F1 score of 0.980. To study whether the increase in F1 score is due to improvements

in precision or recall, we studied the exact precision, recall and F1 scores of the top 2 variant callers as well as the best single variant caller versus the neural network. We find that the neural network is more precise than both, but the recall is rather similar.

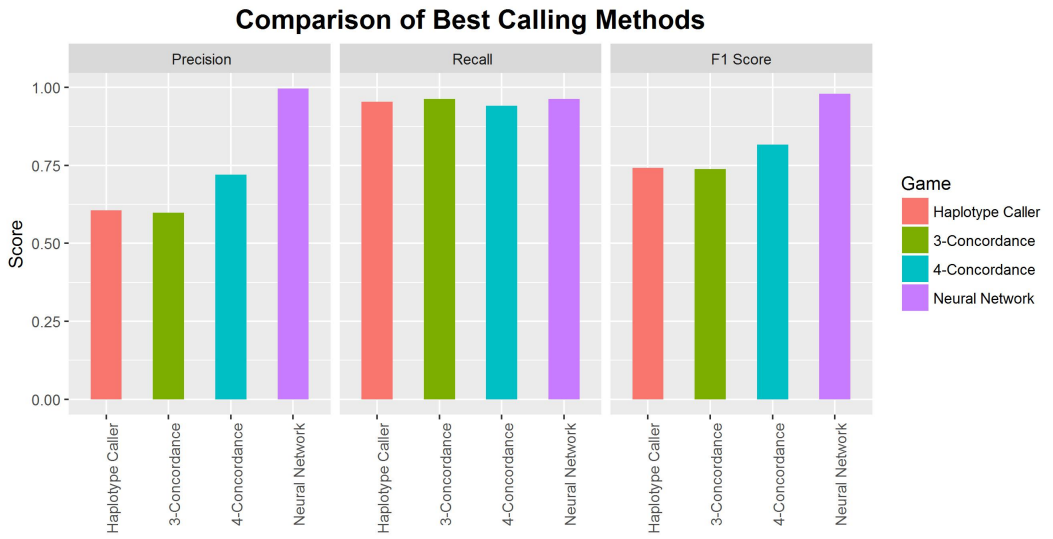

Figure 15: Comparison of Best Variant Callers in terms of Precision, Recall and F1 Score

We find that the neural network is more precise than both, but the recall is rather similar. Increasingly, the recall of all the callers was high, indicating that while all were able to pick out most of the truth class variables, the main errors came from a high number of false positives that were also called as true mutations. Thus, this provides strong evidence that the neural network is able to sieve out false positives within the dataset and stably predict whether a mutation is true or not.

3.7 Benchmarking of Network with NA Datasets

After verification of the neural network architecture, optimised parameters and ability to learn with a simulated dataset, we sought to analyse a real dataset to set the validity of the neural network in validating variants. We studied the NA12878 Genome In a Bottle dataset (Zook et al., 2014), which has been used in other variant calling validation pipelines(Talwalkar et al., 2014; Linderman et al., 2014) and contains a set of high confidence variant calls which we can use as ground truth for training and validation. This set of high confidence variant calls

are obtained from multiple iterations of orthogonal sequencing methods (using Solid, Illumina platforms, Roche 454 sequencing and Ion torrent technologies). The usage of multiple platforms enables an intersection of variants that can be considered as the ground truth. We then sought to see if our neural network can predict the ground truth better than single or concordance based variant callers.

We applied the same methodology to the sequences as with the simulated data and then used our neural network to predict the true variants.

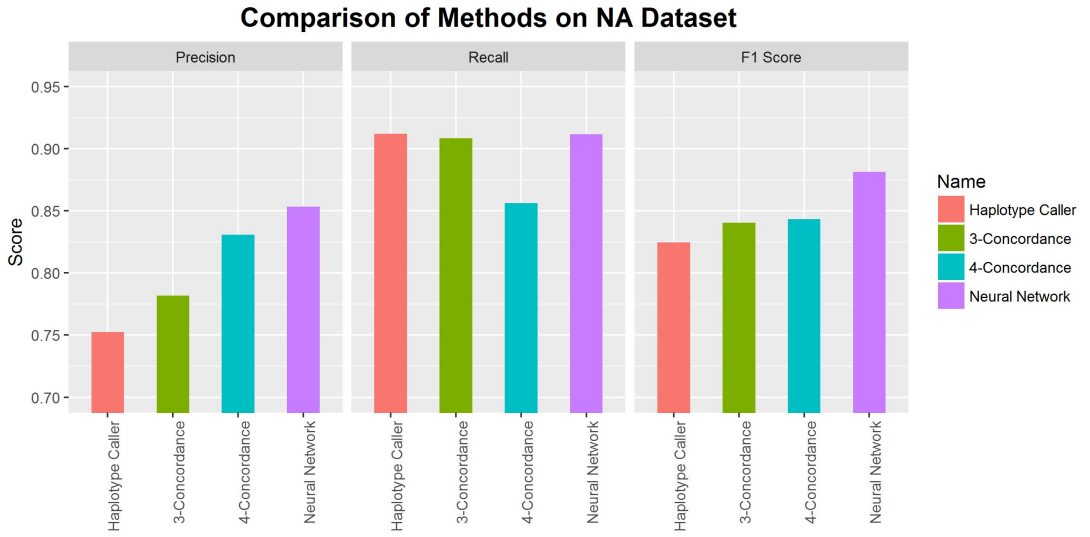

Figure 16: Comparison of Variant Callers

As can be seen from the figure, the neural network was able to predict with higher precision the best single caller, haplotype caller and the 2 best concordance callers, 3-concordance and 4-concordance. In terms of recall, the recall rates were the same between Haplotype Caller, 3 concordance and the neural network. This was an interesting observation as it shows that the neural network is more aggressive in making calls than the 4 concordance neural network, and it is more likely that the calls it makes are correct. Ultimately when we looked at the f1 score, the neural network was able to outperform these variant callers. This validates our neural network pipeline and indicates that we are able to learn from the input features.

3.8 Analysis of gene importance using Bayesian Ranking systems

After validation of high confidence calls, we sought to enable a clear and understandable ranking of genes. We first build a Bayesian network analysis using known functional annotations from ANNOVAR. These were subsequently used to compute the Bayesian probability ranking, which is shown in Figure 17 below.

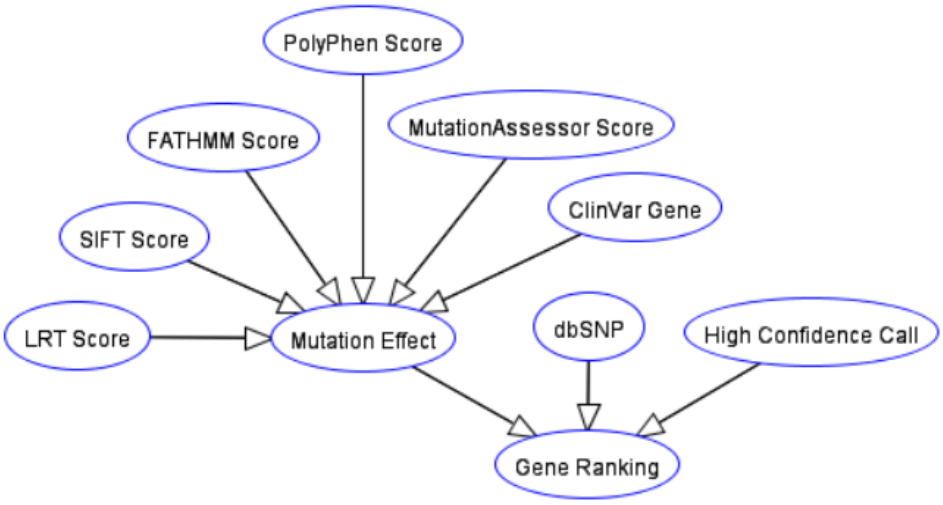

Figure 17: Final Bayesian Network used in Analysis

This network structure was chosen as we wanted to use three different sets of information to update the probability of the gene being important. Firstly, the confidence of the call should matter in how important it is - the more likely a gene is real, the more important it should be. Secondly, the rank should also be determined by how common the variation is, based on studying known SNP polymorphism rates. If it is a common SNP, then the ranking should be downgraded as it is less likely to be a driver mutation. Finally, we sought to predict the overall effect of mutations via an ensemble of mutation effect predictors. These predictors use different methods to predict the average effect of that mutation - based on statistical methods like position specific substitution matrixes and Hidden Markov Models to study the effect of a mutation on protein structure and function. We also used the ClinVar database, a curated repository of known Human variants and their resulting phenotypes. These scores were then aggregated to update the probability of the mutation effect.

Figure 18: Table of Functional Annotations obtained from ANNOVAR

Annotation Name	Information Type	Method	**Scoring Method**
Likelihood Ratio Test	Deleterious Mutation Score	Likelihood Ratio Test of each amino acid is evolving neutrally to the alternative model of evolution under negative selection	Score normalised to [0,1] and used directly in Bayesian Network
MutationAssessor	Deleterious Mutation Score	Mutation rate of homologous sequence subfamilies	Score normalised to [0,1] and used directly in Bayesian Network
SIFT	Deleterious Mutation Score	Position Specific Scoring Matrixes with conserved Sequences	Score normalised to [0,1] and used directly in Bayesian Network
PolyPhen2	Deleterious Mutation Score	naïve Bayes classifier on various multiple sequence alignments methods of homologous proteins and protein structure-based features	Score normalised to [0,1] and used directly in Bayesian Network
FATHMM	Deleterious Mutation Score	Hidden Markov Model used to score MSA based on protein homologous sequences	Score normalised to [0,1] and used directly in Bayesian Network
ClinVar Genes	Known Pathogenic Genes	Database lookup of curated set of relationship between variant calls and human phenotype	Higher Probability of Importance if known pathogenic variant
dbSNP138	Common Single Nucleotide Polymorphisms	Database lookup of curated set of known Human SNPs	Lower Probability of Importance if known common variant

Based on scores provided, we report the update the conditional probabilities using the probabilities chain rule - for the first level, this is given as

$$P(Impt|(Del \ \cap \ Uncom \cap High \ Conc)) = \ P(Impt \cap Del \ \cap \ Uncom \cap High \ Conc) \\ * \ P(Del \cap Uncom \cap High \ Conc) \tag{3}$$

P(Impt) refers to the probability of the gene being important,
P(Del) refers to the probability of the gene being deleterious,
P(Uncom) refers to the probability of the gene being uncommon and
P(High Conc) refers to the probability of the gene being a high confidence call.
Further calculations can be found and derivations can be found in Appendix A

To compute the final probabilites, the software pomegranate was used. This simplifies the node drawing and probabilistic updates of the final ranking scores.

3.9 Validation of Bayesian Network Ranking on PDX dataset

To study the effectiveness of our bayesian network ranking system, we sequenced and analysed a patient derived xenograft (PDX) tumour genome. This tumour genome was grafted onto the immunocompromised mouse from a patient with a known cancer - Diffuse Large B Cell Lymphoma (DLBCL). We chose to analyse lymphoma as lymphoma is a well-known and studied disease model with a well-defined disease progression. The patient derived xenograph model

also allows in vivo studies of the tumour in its environment, and serves as a good model for sequencing and analysis. After sequencing the PDX genome, we put it through our full analysis pipeline, which involves identifying high confidence mutations using the neural networks and then ranking these genes using the bayesian network ranking. Figure 19 shows the top 30 genes by probability.

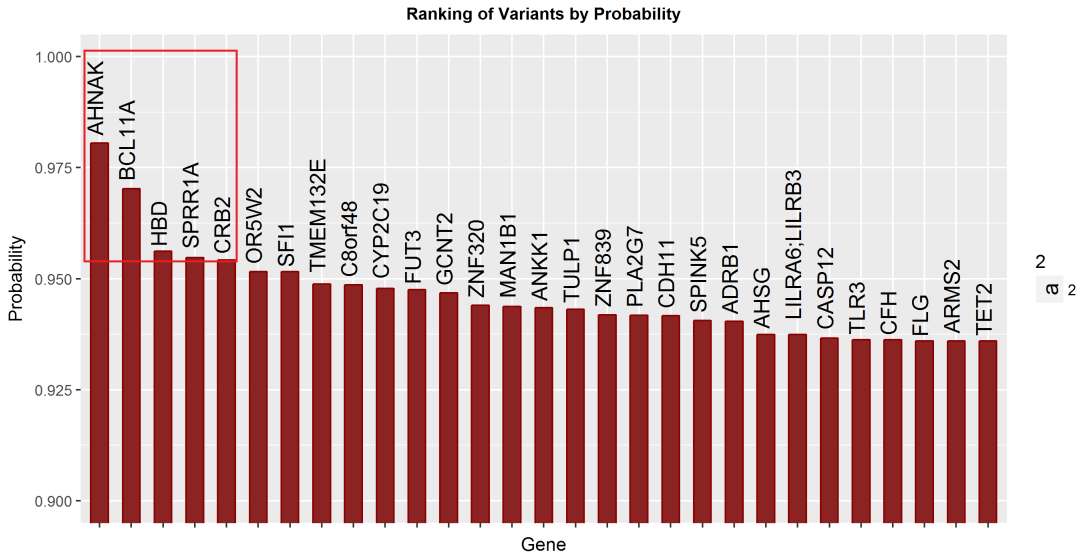

Figure 19: Top 30 genes from Bayesian Ranking Algorithm

Figure 20: Table of Top 5 important genes from Bayesian Ranking

Gene	Full Name	Known Involvement in Lymphoma or Cancer	Evidence	Mutation Location	Predicted Mutation Type
AHNAK	Neuroblast Differentiation-Associated Protein (Desmoyokin)	• Known **tumour suppressor** via modulation of TGFβ/Smad signalling pathway • Known to be **downregulated** in cell lines of **Burkitt lymphomas**	Lee et al., 2014; Amagai et al., 2004; Shtivelman et al, 1992	chr11 - 62293433 T -> C	**non synonymous SNV**
BCL11A	B-Cell CLL/Lymphoma 11A	• Known **proto-oncogene** in **DLBCL** • **Overexpression** of **BCL11A** was found in **75% of primary mediastinal B-cell lymphomas** (a subset of DLBCLs)	Weniger et al., 2006; Schlegelberger et al. 2001; Satterwhite et al., 2001	chr2 - 60688580 C -> G	**non synonymous SNV**
HBD	Hemoglobin Subunit Delta	• Shown to be **expressed** by aggressive **glioblastoma** cell lines	Allalunis-Turner et al., 2013	chr11 - 5255274 G -> A	**stop-gain**
SPRR1A	Small Proline Rich Protein 1A (Cornifin-A)	• Known to be **expressed** in **DLBCL** and **expression** has been shown to correlate with **5 year survival rate**	Liu et al., 2014	chr1 - 152957961 G -> C	**non synonymous SNV**
CRB2	Crumbs 2, Cell Polarity Complex Component	• **Cell polarity and cytoskeletal reorganisation** is known to affect B-cell lymphoma **migration and invasiveness** • **Development of B-cell lymphoma** has also been noted in **Crb2-related syndrome** (bi-allelic mutation of Crb2)	Slavotinek, 2015; Gold et al., 2010	chr9 – 126135887 T -> C	**non synonymous SNV**

Studying the top 5 genes, we found that four of these five genes have been implicated on lymphomas or other cancers. AHNAK is a known tumour suppressor and has been known to be downregulated inlines of Burkitt Lymphoma. BCL11A is a known proto-oncogene in DLBCL, and has been found to be overexpressed in 75% of primary mediastinal B-cell Lymphomas, a subset of DLBCL. SPRR1A, the fourth gene ranked in terms of importance, has been shown to be expressed in DLBCL and its expression has been shown to strongly correlate with 5 year survival rate (Figure 21). Finally development of B-cell lymphoma has been noted in Crb-2 related syndrome, which is a bi-allelic mutaiton of CRB2. Interestingly, the last of the high ranked genes was noted to be a subunit of Hemoglobin. While there is no strong evidence for the role of Hemoglobin in DLBCL, it has been shown to be expressed in aggressive gliobastoma lines, indicating a possible previously unknown role in cancer. This gives us high confidence that the bayesian ranking network is able to pick up important and relevant mutations. Without such a ranking system, we would have to look through over 70 thousand genes, without a way to systematically study their call confidence, as well as their probability of having an effect.

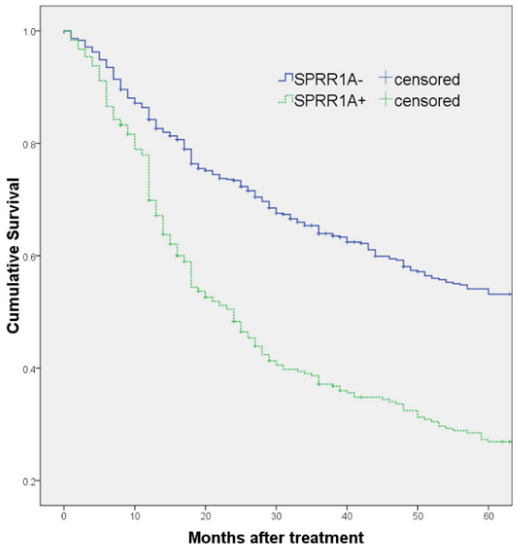

Figure 21: 5 year survival curve of patients with SPRR1A+ and SPRR1A- patients with DLBCL. Source : Zhang et al. (2014), Figure 2.

To aggregate the data from our Bayesian Ranking system, we did a Circos plot for the top 300 genes picked up by our gene ranking system. A Circos enables easy visualisation and analysis of large genome datasets, enabling quick understanding and comprehension of results. From the Circos Plot (Figure 22), we find several interesting gene families that might also be relevant in B-Cell Lymphoma. These include several Toll-Like Receptors(TLRs), TLR3 (chr4,rank 26) and Tlr1(chr4,rank 77) as well as interleukin receptors IL4R (chr16,rank 37) and IL1β (chr2,rank 196). TLRs are of significant interest in cancer due to their involvement in the caspase pathway, and interleukins are also important in cancer due to their importance in mediating inflammation and immune response. Thus, we show that our Bayesian network can be used by Clinicians to quickly interrogate the information from functional annotations and database lookups to understand the disease specifics.

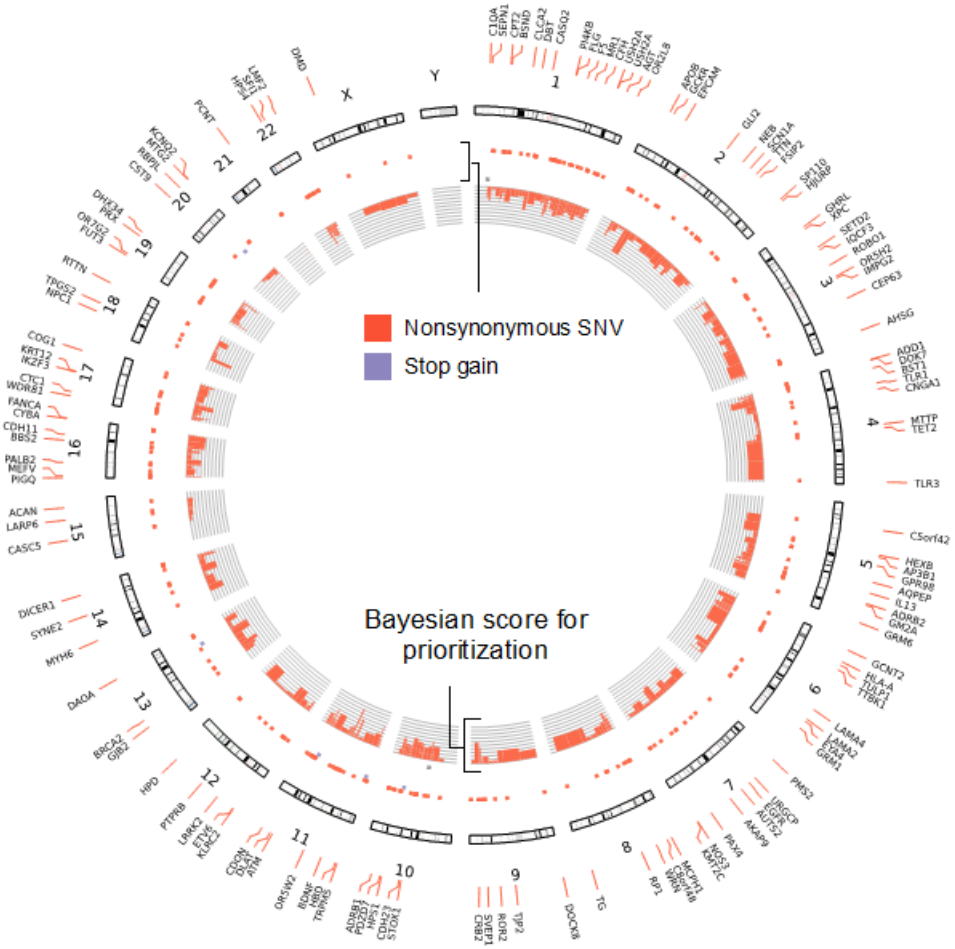

Figure 22: Circos plot of top 300 ranked genes from Bayesian network ranking. In this Circos plot, the outer track indicates the top ranked genes and their positions on the chromosome. The inner track describes the type of mutation that was observed - most mutations were non synonymous SNVs, with a few stop-gain mutataions. The innermost track shows the relative probabilities of each ranked gene.

4 Discussion

We have demonstrate the validation of high confidence variant calls using a deep learning neural network on both real and simulated datasets, and we also show that a Bayesian network is able to rank and prioritise genes in a systematic way so as to obtain important genes. To benchmark our results against other tools that validate deep learning, we compared our results with other methods like VariantMetaCaller(using Support Vector Machines) and BAYSIC(using Bayesian Latent Analysis). Both were used to analyse and predict variants for NA12878 data, but to differences in metrics, processes and datasets, absolute comparison of calling precision and recall is difficult. Looking at relative improvements, VariantMetaCaller observed a 0.04 increase in SNP prediction and 0.07 in indel prediction in terms of area under prediction recall curve (AUPRC) metrics compared to the best single variant caller, and while BAYSIC noted a 0.03 increase in SNP prediction compared to the best single variant caller and 0.05 increase in indel prediction. This is comparable to our own results of a 0.06 increase in both indel and SNP prediction for the NA dataset compared to the best single variant caller, but we used the F1 score metric which in turn uses the best threshold for a binary classifier. Thus, while their results provide evidence of precision improvements at all levels of recall, our results study the optimal thresholds for each specific caller and how different this is. This is because in clinical practice we are interested in the optimal operating conditions.

One definite step moving forward is to incorporate VariantMetaCaller and BAYSIC into our pipelines as negative controls, and measure using the same dataset and same processes whether deep learning is able to outperform these two software. Intuitively, we believe that deep learning will be able to edge out improvements as deep learning is able to form complex representations of the data to learn from that Support Vector Machines are unable to do. Furthermore, evidence from our flat and dense network architecture shows that putting all the features in a single vector and using that to performing machine learning might not be the best method as it is difficult to learn features from it. Finally, deep learning has been shown to outperform other methods in terms of data prediction and classification.

However, one large limitation in the overall approach of measuring each of the methods against the NA12878 dataset is that the high confidence calls provided are not the ground truth. While they have been verified by multiple sets of sequencing methods, they have not been verified by the gold standard sanger sequencing, and so it is difficult to know if there are variants inside the high confidence dataset that are actually false, as well as variants that are not inside the high confidence dataset that are actually true. Hence, this would result in misclassification and wrongly called false negative and false positive results. To solve this problem, a lot of effort has to be put in to obtain a set of verified truth variants, but this might be prohibitively expensive for a large sample size. Still, this would have to be done for us to have a good set of truth variables to test prediction software with before such software can be even considered for use in actual treatment and diagnosis.

It is more difficult to benchmark our results for the Bayesian Network analysis in terms of gene priorisation, because currently used platforms include gene panels (Olek and Berlin, 2002) or manual literature look-ups of disease related genes, such as using the ClinVar database. While gene panels work well in a clinical setting, with NGS data it is hoped that as much information about a person's genome as possible can be used in treatment and diagnosis. Using a gene ranking system instead of just looking at a set of implicated genes might allow doctors to find out tease out possible homologs or interacting agents that might be related to the known deleterious genes (perhaps in the gene panel) and integrate that into their treatment and diagnosis. One interesting extension we would definitely like to move into in the future is to be able to integrate a druggable genome into the network, enabling the prioritisation of genes who have possible candidate drug targets. This would enable doctors to notice further possible drug candidates that would work very well on the gene profile of the patient that they might not have considered previously, thus increasing their scope of possible treatment options and augmenting their skills.

Other directions in the future include being able to include extra variant callers which will provide it with even more feature data, enabling it to make better predictions. We would also like to move everything onto a web interface such that it is accessible for use in terms of both variant validation and gene priorisation. This would enable easy access to both the validation and priorisation pipelines.

Thus, in this paper, we have shown the use of deep learning neural networks to successfully validate variants in both real and simulated datasets. We also show that using a Bayesian network is able to identify important genes within a lymphoma disease sample.

5 Appendixes

5.1 Neural Network Learning

Machine learning is underpinned by two key phases, the feed-forward phase and the backpropagation phase. The feedforward phase describes the computation of a prediction, and during this phase the input features are used to compute the final output prediction. For a simple network below :

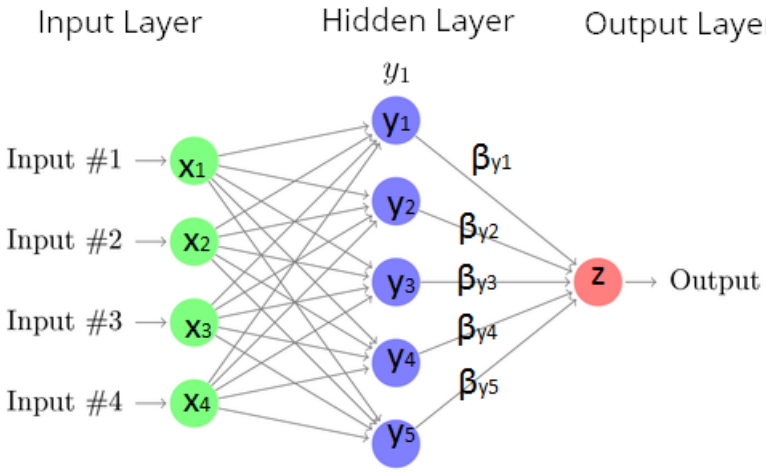

Figure 23: Example neural networks with nodes and weights

The final prediction, z is computed with the equation :

$$z = \beta_{y1} * y_1 + \beta_{y2} * y_2 + \beta_{y3} * y_3 + \beta_{y_4} * y_4 + \beta_{y_5} * y_5 \tag{4}$$

Where β indicates the weights linking each output to the input of z and each of the y_i terms are also computed in the same manner from the x_i layer. At each node (x,y,z), there is also the existence of an activation function that modifies the input of the node to compute an output. Commonly used activation functions include the rectified linear unit (ReLU), sigmoid functions like hyperbolic tangent and logistic function,

$$S(T) = \frac{1}{1 + e^{-t}} \tag{5}$$

Thus, the final prediction can be seen as a summation of all weights multiplied by the activation output of each node. In theory, we can expand each of the y_i terms in equation (2) to include the y_i layer activation function as well as rewrite the y_i layer inputs in terms of the sum of outputs and weights from the x_i layers. This complex integration of terms allows for the neural network to form complex continuous decision boundaries as the neural networks can compute sophisticated non-linear prediction functions despite being a fundamentally linear model.

After a prediction is made, we then have to check whether it is correct, and change our weights if an erroneous prediction was made (Figure 24).

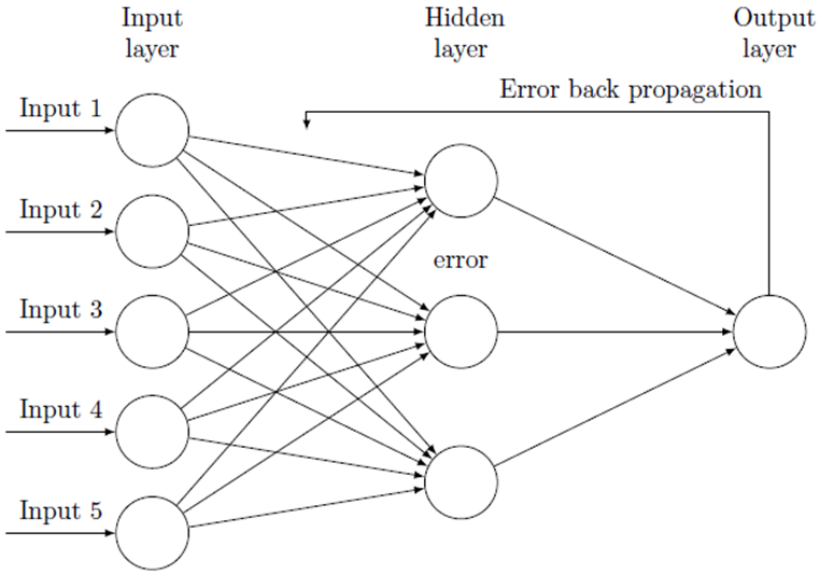

Figure 24: Backpropagation of Error Terms

This is the backpropagation step, which involves backpropagating the error terms from the output layers to the input layer and updating the weights at each node based on the differential relationship between the error and each specific gradient. Specifically, this is governed by the optimiser functions which have been mentioned earlier - one example of such an function is the Stochastic Gradient Descent function, which is

$$\beta_{yi}^{n} = \beta_{yi}^{n-1} - \alpha \frac{\partial E_n(\beta)}{\partial \beta_i} \tag{6}$$

Here, each β term indicates a gradient, α is a constant for the learning rate and $\frac{\partial E_n(\beta)}{\partial \beta_i}$ is the term used to modify the weight of the gradient based on the cost function $E_n(\beta)$. The idea used in all backpropagation functions is gradient descent, where the contribution of the gradient term to the error is computed and the gradient is changed by an amount in order to reduce the future contribution of the gradient to that error. Here it is useful to consider what the cost function $E_n(\beta)$ is. It is essentially the error rate when a set of gradients is used to perform predictions, as it measures how many accurate predictions were made and how many wrong predictions were made. For a binary class predictor (which is what we are using, only true and false), this is given by the equation

$$E(\beta) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{2} y_{ij} log(p_{ij}) \tag{7}$$

where y_{ij} indicates the empirical observed probabilities of each class label while $log(p_{ij}$ is the theoretical probabilities of each class label. From this term, we see that if the neural network predicts something with a high probability (y_{ij} is high) and it is false (p_{ij} is low) so then $log(p_{ij}$ is a big negative number, and so the cost function will very high. On the other hand, if y_{ij} and p_{ij} is high then the entropy will be close to zero, indicating a correct prediction. Since each of the prediction terms can be rewritten in terms of the gradient(rewrite z in terms βy_i and so on, we can theoretically compute the contribution of each gradient to the cost function to see how the cost function changes as the gradient changes. Thus, this is what gradient descent does - it tries to see how the cost function changes as each gradient changes, then attempts to move the gradient in the direction that minimises the error term. This is best seen in the graph below where the gradient, or specifically the partial differentiation of the cost function with regards to each gradient is used to move the gradient to a new position so as to minimise the error term. Thus, machine learning is in essence a minimisation problem - we want to find a set of weights that minimises the cost function, and because the cost function describes how many predictions we made correctly, this is also training our network to accurately predict outputs from inputs.

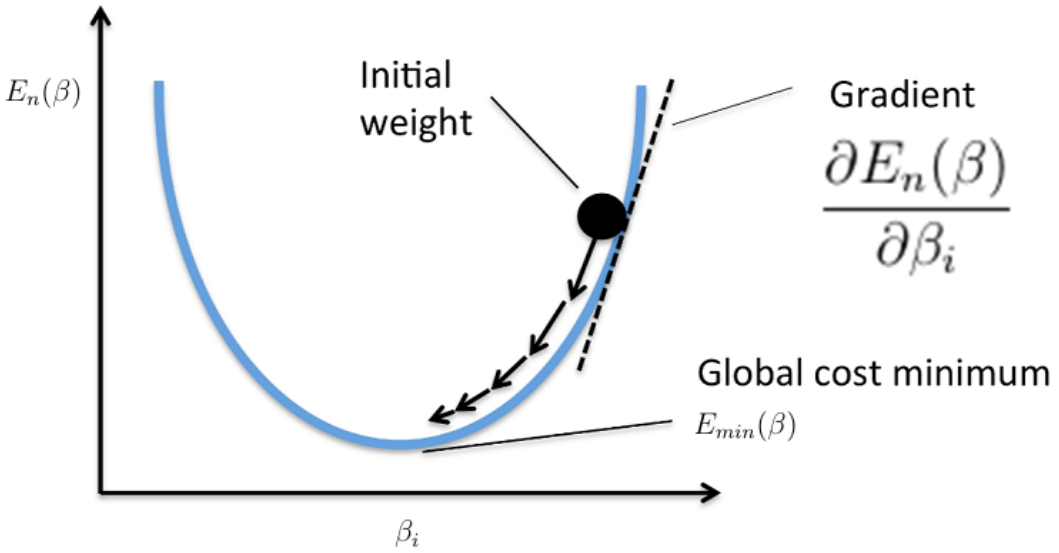

Figure 25: Gradient Descent

5.2 Feature Engineering

We subset our features into three broad sets, which are base-specific information, sequencing error and bias information features, and calling and mapping quality. Base information tells us base specific properties, including information contained in the base as well as the quality of sequenced bases in the samples. Sequencing error and bias features attempt to tease out potential biases in sequencing, including features such as GC content, longest homopolymer run and as well as allele balances and counts. Finally, calling and mapping quality scores provide information on the mapping and calling confidence of the variant callers, and includes features such as genotype confidence and mapping quality. In all, these sets of information provide information on the key aspects of variant calling - specifically the properties of the bases in the samples, the characteristics of the sequencing process and finally the variant calling and mapping algorithms.

Base Information

Shannon Entropy

Shannon Entropy captures the amount of information contained inside the allele sequences. It is calculated using the equation :

$$H(X) = -\sum_{i=1}^{n} P(x_i) \log_2 P(x_i) \tag{8}$$

where $P(x_i)$ is the probability of finding each base at each position. Thus, we calculate the entropy by summing up the probabilities/lg(probabilities) at each position. This prior probability is calculated in two ways and both are used as features - firstly, the overall genome base probabilities are calculated over the entire genome, and thus the entropy is related to the probability of finding a base at any position in the genome. The second way prior probability is calculated is to take a region of space around the allele (10 bases plus the length of the allele in our calculations) and use those probabilities to calculate the entropy of the allelic sequence. Intuitively, it attempts to find out the amount of information contained within the allelic sequence and hopefully the neural network is able to use the information to determine the validity of a mutation.

Kullback Leibler Divergence

The Kullback-Leibler Divergence feature is similar to Shannon entropy, but instead we use this to measure the informational change converting from the reference to the allele sequence. The Kullback-Leibler Divergence is calculated as follows :

$$D_{KL}(P||Q) = -\sum_{i=1}^{n} P(x_i) \log_2 \frac{P(x_i)}{Q(x_i)} \tag{9}$$

where $Q(x_i)$ is the prior probability of finding each base at each position based on the genomic region around the allele, while $P(X_i)$ is the posterior probability of finding a specific base inside the allelic sequence. Thus, the KL divergence describes the informational gain when the probabilities from Q is used to describe P. Intuitively, since we know the base probabilities of the region, we can then study the probabilities observed in the reference allelic sequence and see how well $Q(X_i)$ probabilities is able to approximate $P(X_i)$ probabilities.

Base Quality

Base quality refers to the Phred score probability that the called allele is wrong. It is given by the equation :

$$P = 10^{\frac{-Q}{10}}$$

Where P is the Base Quality and Q is the probability that the allele called is wrong. This is a number computed by the sequencing machine based on the quality of the base samples provided, and tells us how much confidence the sequencing machine has in calling that base.

Sequencing Biases and Errors

GC content

This feature computes the calculated GC content of reference genome, which may affect sequencing results and accuracy as regions with a GC content are known to be more difficult to sequence. This is because of the greater strength of GC bonds, resulting in errors and biases

in sequencing (Benjamini & Speed, 2012).

Longest homozygous run

Homopolymer runs (AAAAAAAA) are known to cause sequencer errors (Quail et al.,2012), and might be a factor in determining in whether a variant is true. This because long homopolymers provide the same type of signal to the sequencing machine, resulting in a difficult in estimating the magnitude of the signal or rather how many bases are in that homopolymer, resulting in errors and wrongly called variants. The reference sequence region including the allele was checked for homopolymer runs.

Allele Count and Allele Balance

Allele count gives the total number of alleles in called phenotypes, while allele balance gives the ratio of final allele called over all other alleles called(reference allele for heterozygous calls, or other alleles for homozygous calls). Both these features give us information of possible biases in the sequencing machine.

Calling and Mapping Qualities

Genotype Likelihood

The genotype likelihood provides the phred-scaled likelihood scores of how confident the caller is in determining that it is a homozygous or heterozygous call, and for the homozygous calls whether it is a more likely to be a bi-allelic mutation or no mutation at all. This feature thus gives us the confidence of the caller in determining if one or two alleles have mutated.

Read Depth

Mapped read depth refers to the total number of bases sequenced and aligned at a given reference base position. The read depth tells us how many reads contributed to a specific call, and thus provides information on how much evidence there is for the variant call

Quality by Depth

Quality by Depth is computed by dividing the quality score against allele depth, to obtain an average score of allele quality. This composite feature provides information on the information provided by each read supporting the call

Mapping Quality

Mapping quality is originally a score provided by the alignment method and gives the probability that a read is placed accurately. The variant callers compute a overall mapping quality of the reads that provide evidence for a variant call which is given in this feature. A low mapping quality means that there are multiple positions where the reads contributing to this variant call could have gone, and thus providing evidence that this might not be an accurate call due to poor mapping.

5.3 Mathematical and Statistical Tools

A. Derivation of F1 Score

The F1 score is a useful measure as it can measure both the precision as well as the recall of a predictor. For a binary predictor with a binary truth class(Figure 26), we can obtain 4 types of results - true positives, true negatives, false positives and false negatives.

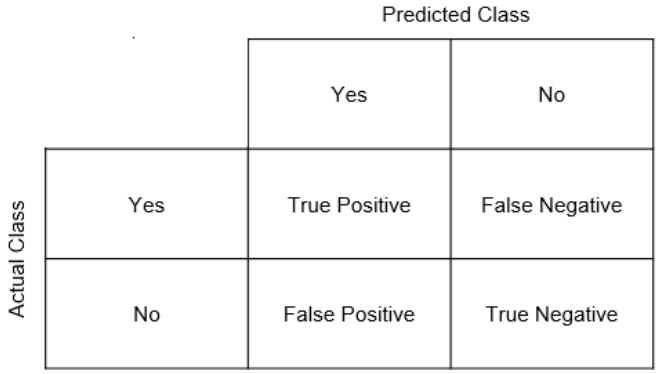

Figure 26: Confusion Matrix

True positives are positive predictions that are made that are actually positive class labels, while false positives are positive predictions that are made that have negative class labels. Similarly, true negatives are negative predictions that have negative class labels, while false negatives are negative predictions that are actually positive class labels. From this, we can define two equations, precision and recall. Precision is defined as (8) while recall is defined as (9).

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive} \tag{10}$$

$$Recall \quad = \frac{True\ Positive}{True\ Positive + False\ Negative} \tag{11}$$

Precision tells us how likely a positive prediction made will be true, while recall tells us how much of the truth class positive predictions the predictor is able to encompass. Thus, a predictor can have a high precision but low recall (makes few predictions but are very accurate) or a high recall and low precision(makes a lot of predictions that capture all truth variables, but have a lot of false positives as well). In genomics, both types of errors are not desired - we would want all the predictions to be true (precision), while not losing out on any important mutations

(recall). Thus, we use the composite metric, the F1 score, that looks at the overall precision and recall of a predictor. It is defined as follows :

$$F1\ Score \quad = \frac{2 * Precision * Recall}{Precision + Recall} \tag{12}$$

B. Principal Components Analysis (PCA)

Principal Components Analysis (PCA) is a commonly used tool for dimensionality reduction. It was first proposed by Pearson in 1901 (Pearson, 1901) and has been commonplace in many data analytics and signal processing methodologies (Jolliffe, 2002). PCA works by attempting to discover orthogonal principal components (PCs) that are able to represent the original data. Specifically, this means that the PCs are able to capture variance in the datasets. This is done by finding the Eigenvalues and Eigenvectors of the dataset, with the eigenvectors representing a linear combination of all input variables and the eigenvalues representing the amount of variance that that eigenvector is able to represent. Ultimately, we select n eigenvectors that is able to represent a percentage of variance in our dataset. Because each eigenvector is orthogonal, they are able to capture the variance in the dataset. For our analysis, we decided to use 8 principal components - we took the limit as the last principal components that was able to represent at least 0.5% of variance in the dataset.

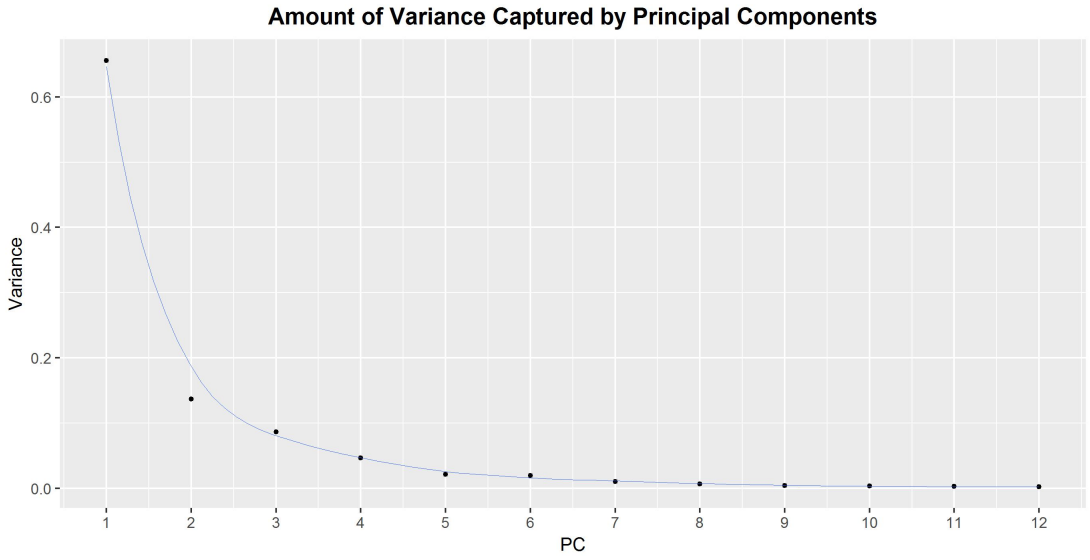

Figure 27: Variance captured by first 12 principal components

To carry out PCA, we used the preprocessing step SciPy to normalise all the input vectors to mean 0 and standard deviation 1. Subsequently, we perform principal components decomposition to obtain the eigenvector transformed representation of the dataset, and their corresponding eigenvalues. We then fit 8 of the principal components that explained the largest amount of variance into the neural network to study if it is able to learn from the compressed representation of the input features.

B. Synthetic Minority Overrepresentation Technique (SMOTE) SMOTE is a statistical technique described in by Chawla et al. (2002) to overcome problems with imbalanced datasets that are common in machine learning. SMOTE oversamples the training class with less variables in a way that tries not to replicate data points (that makes certain data points over-represented) without creating new invalid training training examples. It does this by taking the intersection of two nearest data points of the same training class. This can be seen in Figure 28.

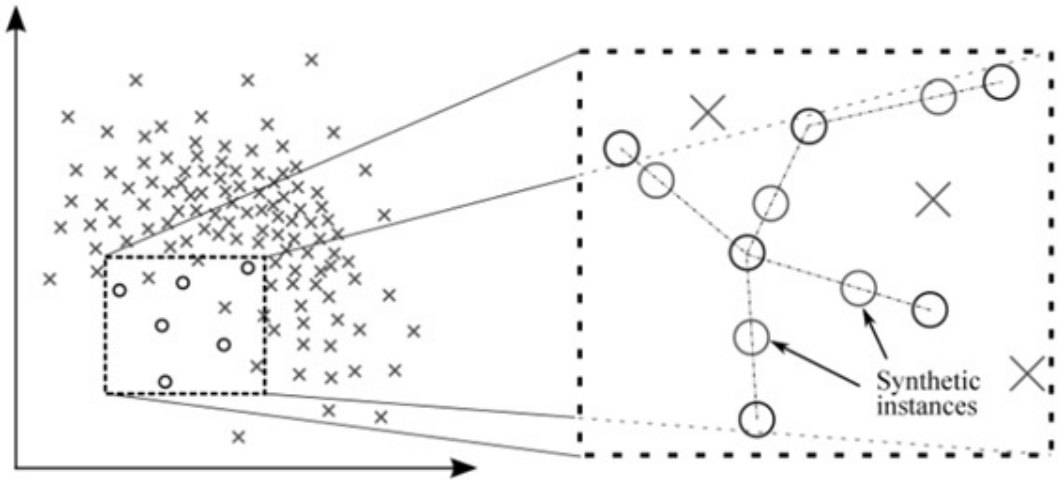

Figure 28: SMOTE oversampling algorithm

In doing so, it creates a more generalised representation of the sample class with less training examples, without replicating certain datapoints and without creating invalid data. This enables intelligent oversampling of the dataset to balance out the positive and negative feature classes. SMOTE has been shown to be valid for other datasets including sentence boundary detection (Liu et al., 2006) and data mining (Chawla, 2005).

6 Bibilography

Abyzov, A., Li, S., Kim, D. R., Mohiyuddin, M., Sttz, A. M., Parrish, N. F., ... & Korbel, J. O. 2015. Analysis of deletion breakpoints from 1,092 humans reveals details of mutation mechanisms.Nature communications,6.

Angrist, M. 2016. Personal genomics: Where are we now?.Applied & translational genomics,8, 1.

Chawla, N. V. 2005. Data mining for imbalanced datasets: An overview. InData mining and knowledge discovery handbook*pp*.853 − 867. Springer US.

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. 2002. SMOTE: synthetic minority over-sampling technique.Journal of artificial intelligence research,16, 321-357.

Chen, Y., Lin, Z., Zhao, X., Wang, G., & Gu, Y. 2014. Deep learning-based classification of hyperspectral data. IEEE Journal of Selected topics in applied earth observations and remote sensing, 76, 2094-2107. Chicago

Cornish, A., & Guda, C. 2015. A comparison of variant calling pipelines using genome in a bottle as a reference.BioMed research international,2015.

Danecek, P., Auton, A., Abecasis, G., Albers, C. A., Banks, E., DePristo, M. A., ... & McVean, G. 2011. The variant call format and VCFtools.Bioinformatics,2715, 2156-2158.

DePristo, M. A., Banks, E., Poplin, R., Garimella, K. V., Maguire, J. R., Hartl, C., ... & McKenna, A. 2011. A framework for variation discovery and genotyping using next-generation DNA sequencing data. Nature genetics, 435, 491-498.

Escalona, M., Rocha, S., & Posada, D. 2016. A comparison of tools for the simulation of genomic next-generation sequencing data.Nature Reviews Genetics,178, 459-469.

Garrison, E., & Marth, G. 2012. Haplotype-based variant detection from short-read sequencing.arXiv preprint arXiv:1207.3907.

Garrison, E., & Marth, G. 2012. Haplotype-based variant detection from short-read sequencing.arXiv preprint arXiv:1207.3907.

Gzsi, A., Bolgr, B., Marx, P., Sarkozy, P., Szalai, C., & Antal, P. 2015. VariantMetaCaller: automated fusion of variant calling pipelines for quantitative, precision-based filtering. BMC genomics, 161, 1.

Huval, B., Wang, T., Tandon, S., Kiske, J., Song, W., Pazhayampallil, J., ... & Mujica, F. 2015. An empirical evaluation of deep learning on highway driving.arXiv preprint arXiv:1504.01716.

Hwang, S., Kim, E., Lee, I., & Marcotte, E. M. 2015. Systematic comparison of variant calling pipelines using gold standard personal exome variants.Scientific reports,5, 17875.

Jolliffe, I. 2002.Principal component analysis. John Wiley & Sons, Ltd.

Kingma, D., & Ba, J. 2014. Adam: A method for stochastic optimization.arXiv preprint arXiv:1412.6980.

LeCun, Y., Bengio, Y., & Hinton, G. 2015. Deep learning.Nature,5217553, 436-444.

Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., ... & Durbin, R. 2009. The sequence alignment/map format and SAMtools. Bioinformatics, 2516, 2078-2079.

Linderman, M. D., Brandt, T., Edelmann, L., Jabado, O., Kasai, Y., Kornreich, R., ... & Schadt, E. E. 2014. Analytical validation of whole exome and whole genome sequencing for clinical applications.BMC medical genomics,71, 20.

Liu, X., Han, S., Wang, Z., Gelernter, J., & Yang, B. Z. 2013. Variant callers for next-generation sequencing data: a comparison study.PloS one,89, e75619.

Liu, Y., Stolcke, A., Shriberg, E., & Harper, M. 2005, *June*. Using conditional random fields for sentence boundary detection in speech. InProceedings of the 43rd Annual Meeting on Association for Computational Linguistics*pp*.451 − 458. Association for Computational Linguistics.

Lpez, V., Fernndez, A., Garca, S., Palade, V., & Herrera, F. 2013. An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics.Information Sciences,250, 113-141.

Maas, A. L., Hannun, A. Y., & Ng, A. Y. 2013, *June*. Rectifier nonlinearities improve neural network acoustic models. InProc. ICML*Vol*.30, *No*.1.

McKenna, A., Hanna, M., Banks, E., Sivachenko, A., Cibulskis, K., Kernytsky, A., ... & DePristo, M. A. 2010. The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data.Genome research,209, 1297-1303.

Mohiyuddin, M., Mu, J. C., Li, J., Asadi, N. B., Gerstein, M. B., Abyzov, A., ... & Lam, H. Y. 2015. MetaSV: an accurate and integrative structural-variant caller for next generation

sequencing. Bioinformatics, btv204.

O'Rawe, J., Jiang, T., Sun, G., Wu, Y., Wang, W., Hu, J., ... & Wei, Z. 2013. Low concordance of multiple variant-calling pipelines: practical implications for exome and genome sequencing.Genome medicine,53, 1.

Pearson, K. 1901. Principal components analysis.The London, Edinburgh and Dublin Philosophical Magazine and Journal,62, 566.

Rehm, H. L. 2017. Evolving health care through personal genomics.Nature Reviews Genetics.

Ruder, S. 2016. An overview of gradient descent optimization algorithms.arXiv preprint arXiv:1609.04747.

Sandmann, S., de Graaf, A. O., Karimi, M., van der Reijden, B. A., Hellstrm-Lindberg, E., Jansen, J. H., & Dugas, M. 2017. Evaluating Variant Calling Tools for Non-Matched Next-Generation Sequencing Data.Scientific Reports,7.

Schirmer, M., DAmore, R., Ijaz, U. Z., Hall, N., & Quince, C. 2016. Illumina error profiles: resolving fine-scale variation in metagenomic sequencing data.BMC bioinformatics,171, 125.

Spencer, D. H., Abel, H. J., Lockwood, C. M., Payton, J. E., Szankasi, P., Kelley, T. W., ... & Duncavage, E. J. 2013. Detection of FLT3 internal tandem duplication in targeted, short-read-length, next-generation sequencing data. The Journal of molecular diagnostics, 151, 81-93.

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. 2014. Dropout: a simple way to prevent neural networks from overfitting. Journal of Machine Learning

Research, 151, 1929-1958.

Sutskever, I., Martens, J., Dahl, G. E., & Hinton, G. E. 2013. On the importance of initialization and momentum in deep learning. ICML 3, 28, 1139-1147.

Talwalkar, A., Liptrap, J., Newcomb, J., Hartl, C., Terhorst, J., Curtis, K., ... & Patterson, D. 2014. SMaSH: a benchmarking toolkit for human genome variant calling.Bioinformatics,3019, 2787-2795.

McKenna, A., Hanna, M., Banks, E., Sivachenko, A., Cibulskis, K., Kernytsky, A., ... & DePristo, M. A. 2010. The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data.Genome research,209, 1297-1303.

Tieleman, T. and Hinton, G. Lecture 6.5 - RMSProp, COURSERA: Neural Networks for Machine Learning. Technical report, 2012

Van Der Maaten, L., Postma, E., & Van den Herik, J. 2009. Dimensionality reduction: a comparative.J Mach Learn Res,10, 66-71.

Xie, M., Lu, C., Wang, J., McLellan, M. D., Johnson, K. J., Wendl, M. C., ... & Ozenberger, B. A. 2014. Age-related mutations associated with clonal hematopoietic expansion and malignancies.Nature medicine,2012, 1472-1478.

Yan, Y., Chen, M., Shyu, M. L., & Chen, S. C. 2015, *December*. Deep learning for imbalanced multimedia data classification. InMultimedia ISM, 2015 IEEE International Symposium on*pp.*$483 - 488$. IEEE.

Ye, K., Schulz, M. H., Long, Q., Apweiler, R., & Ning, Z. 2009. Pindel: a pattern growth

approach to detect break points of large deletions and medium sized insertions from paired-end short reads.Bioinformatics,2521, 2865-2871.

Zook, J. M., Chapman, B., Wang, J., Mittelman, D., Hofmann, O., Hide, W., & Salit, M. 2014. Integrating human sequence data sets provides a resource of benchmark SNP and indel genotype calls.Nature biotechnology,323, 246-251.