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

1 Introduction

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. 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 tool, INSERTNAMEHERE, to integrate data from multiple variant callers, filter true variants and prioritise their importance. This tool uses deep learning for filtering variants, and bayesian updating to determine variants that are the most important.

1.1 Deep Learning in Variant Calling Methodology

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 devia-

tions 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 haplotypes 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 precision of such a tool would be poor due to the differential sensitivity of callers to edge cases. 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.

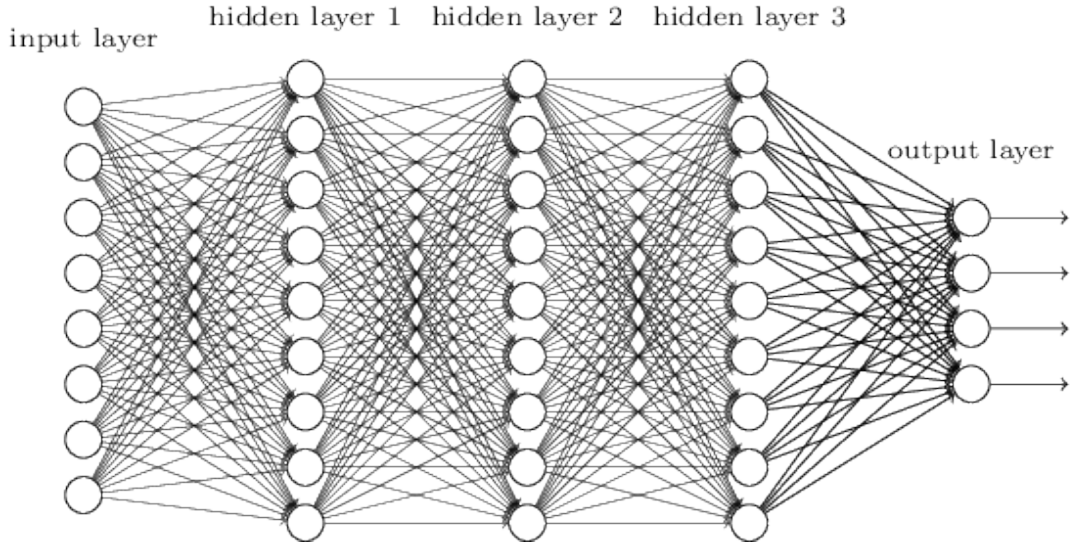


Figure 1: A Neural Network with 1 input layer, 3 hidden layers and 1 output layer. This represents a densely connected neural network, where each node is connected to every node of the preceding and subsequent layers. At each node, linking functions can be defined to apply a mathematical transform to 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. This in part mimics the way we learn - experience teaches us that certain stimuli will result in specific effects (when we see a lightning bolt, we can expect the sound of thunder), and thus when new input comes in (a lightning bolt is seen), we can predict that the output that would arise (thunder is heard). In variant calling, deep learning will allow us to predict based on variant calling patterns 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.

1.2 Bayesian Networks in Gene Prioritisation

The second problem of gene prioritisation arises because there are a multitude of data sources we can draw on to analyse how important a gene is. This includes 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 (see Appendix N), but provide a list of effects. In our tool, 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. 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 N.

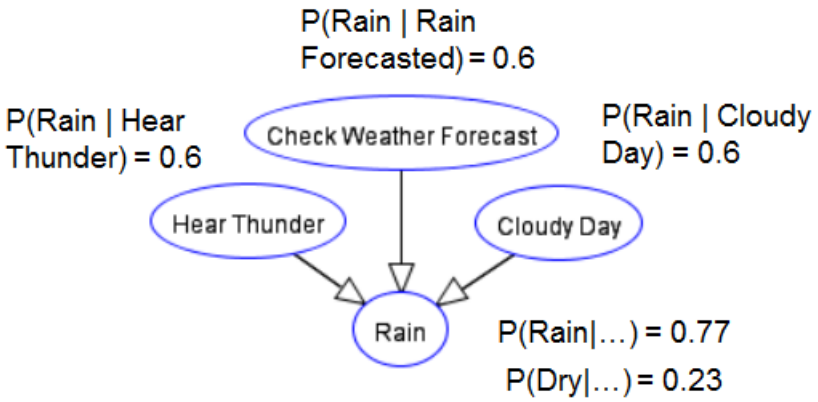


Figure 2: A Bayesian Network for Rain Prediction.

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.

1.3 Aims and Research Structure

In this paper, we will describe the building of a tool, INSERTNAMEHERE to perform both variant calling and. First, we simulated sequence reads with error rates and ground truth

variants. This is critical to obtain a dataset that we can use to train and optimise our deep learning neural network. Subsequently, we benchmark our neural network against the other variant callers and combinations of those callers. After we validate the usage of our network on simulated datasets, then we use optimized network to train on reference genome (NA12878) using high confidence calls as ground truth. We then validate Finally, we build a Bayesian network based on high confidence calls and functional annotations to rank mutations. We note its ability to be able to validate genes in both simulated and real datasets, and we are able to apply our bayesian gene ranking system on real patient dataset to find important genes.

2 Materials and Methods

2.1 Artificial Datasets

Artificial genomes are the best methods to analyse our neural networks on as the ground truth which are the truth variants inside the genome can be known (Escalona, Rocha & Posada, 2016). This allows accurate verification of prediction schemes. It is difficult to obtain complete truth datasets for real genomes as due to the inhibitory cost of checking every variant called. Thus, artificial genomes present a simple way to simulate NGS data with perfectly known ground truth variants to test our validation platform. From this, we created a genome with over 300,000 random mutations (ground truth variants) spread over the chromosomes as can be seen below in Figures N.

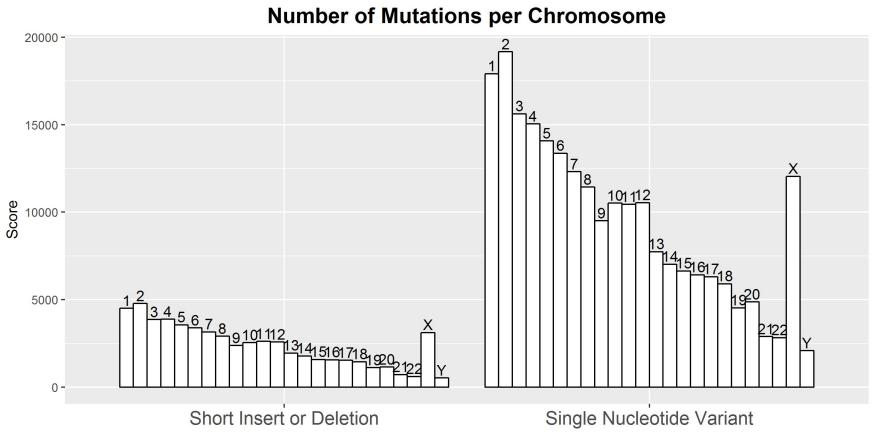


Figure 3: Number of ground truth mutations (variants) created in each chromosome

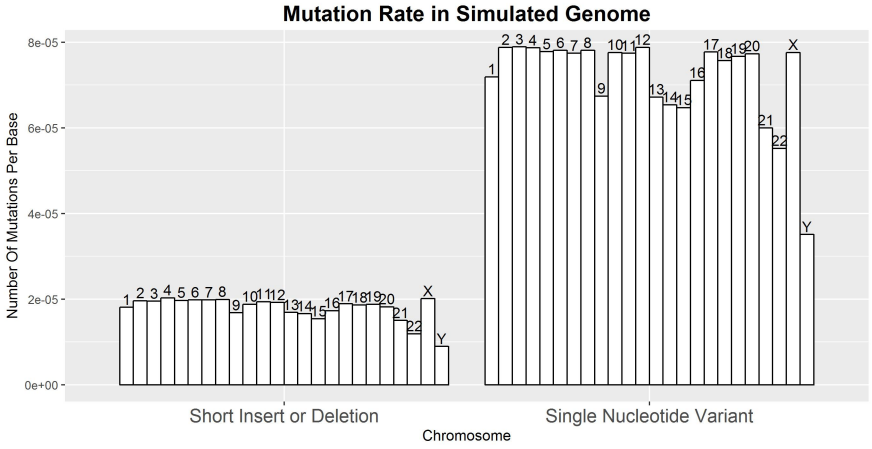


Figure 4: Mutation rate per base in each chromosome

After generating a ground truth model, we simulated sequence reads with error rates and ground truth variants(figure N). To accurately simulate variants, error profiles of Illumina sequencing data were obtained from published articles (Schirmer et al., 2016). Mason, an artificial genome simulator software was used in order to generate an artificial genome for analysis. This program takes in error rates and ground truth variants, and produces FastQ reads that can be used in later analysis as simulated sequencing data.

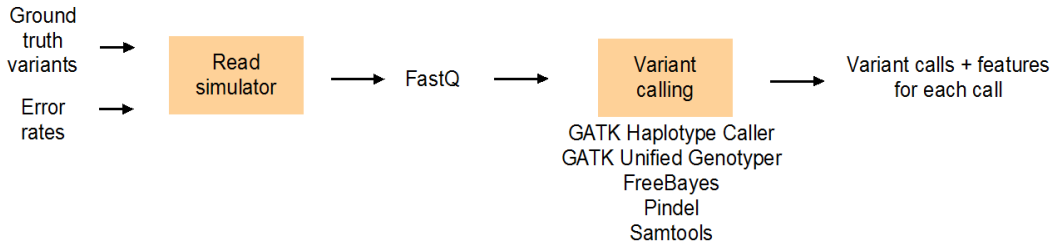


Figure 5: Pipeline for simulation of artificial genome for analysis

2.2 Feature Engineering

In order to train the neural network, features were extracted from the variant callers, as well as engineered from the original dataset. All the features used in the training can be found in table N. More information and descriptions of the mathematics behind the engineered features can be found in appendix N. Features were engineered based on obtaining information for various aspects of variant calling, including how good quality of each base is (Base Quality, Read Depth), the confidence we have in the alignment (Mapping Quality), possible biases in the sequencing machine (Allele Balance, Allele Count) and finally the amount information contain in each variant call (Entropy, KullbackLeibler divergence). More information on the features used can be found in Appendix N.

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	+	+	+	+	+	+			+

2.3 Variant Callers

Variant callers were chosen for our 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 N). 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). The features and differences of all 5 callers can be found in Table 1. 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.

2.4 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 bayesian network analysis to predict genes in this dataset.

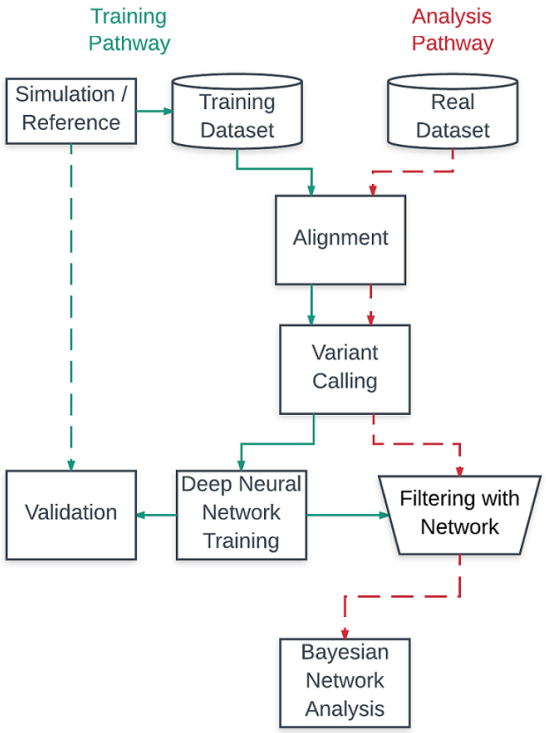


Figure 6: Overall Analytical Pipelines - Pipelines were implemented using the Groovy Domain Specific Language, NextFlow

2.5 Technologies

For our deep learning networks, we used the Keras library with a TensorFlow backend. TensorFlow was chosen due to its superior performance on single machines with multiple cores. On our Ubuntu compute cluster, TensorFlow's distributed CPU computation and queue management system enabled better performance in network training compared to other backend machine learning technologies technologies. For more explanation on the algorithms underpinning deep learning, see Appendix (INSERT) for more information

The general programming platform used was Python. 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 INSERT) for more information. Pomegranate was used to generate and compute the

probabilistic model and ranking system for our Bayesian Network (INSERT see MM methods section N for more information). Finally, PyVCF was used to parse the VCF files into python objects for easy manipulation.

2.6 PDX mouse model development and sequencing

To-get - Data from Dr Ban

3 Results and Discussion

3.1 Network Architecture

We systematically tested out various neural network architectures to see which architecture would perform the best. The architectures tested out were the flat architecture with 7 densely connected layers of 80 nodes each, 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 A for more details of the PCA analysis). Finally, the last architecture we tested was the merged network. The overall structure of the networks can be seen in Figure N.

Figure 7: Feature Engineering Table

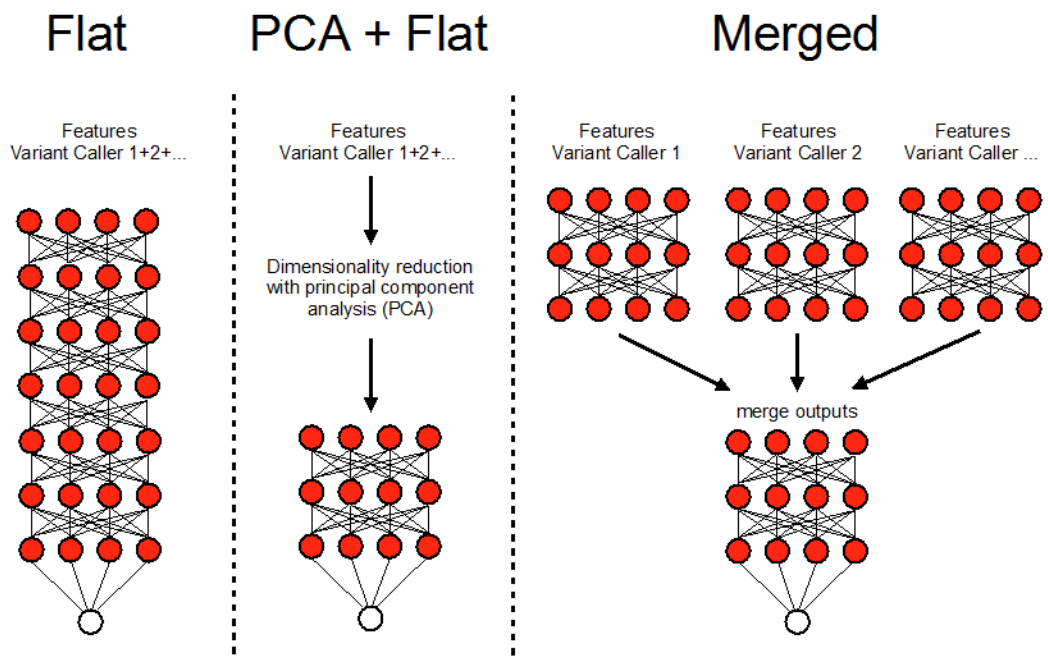
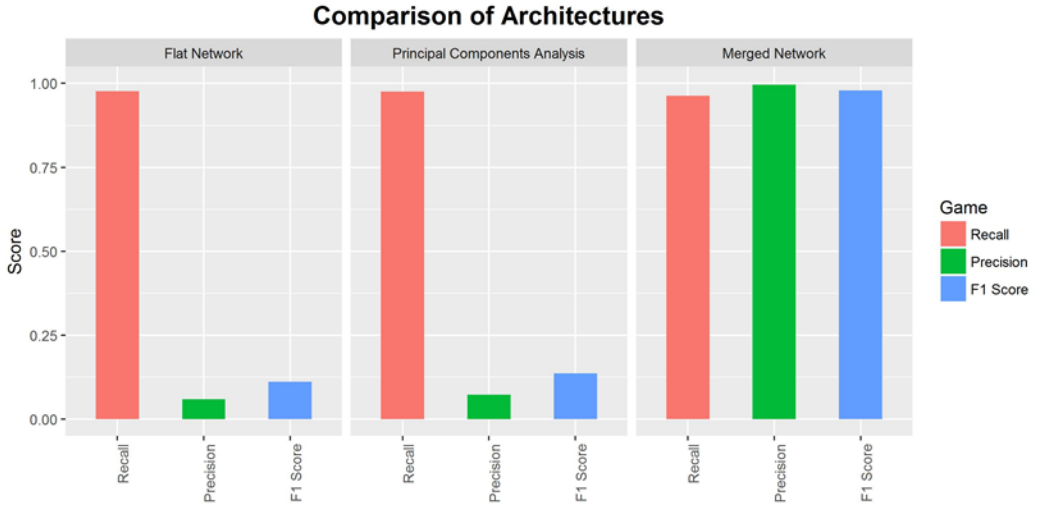


Figure 8: Feature Engineering Table



Initially, with the flat network, the precision rate was very low, 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

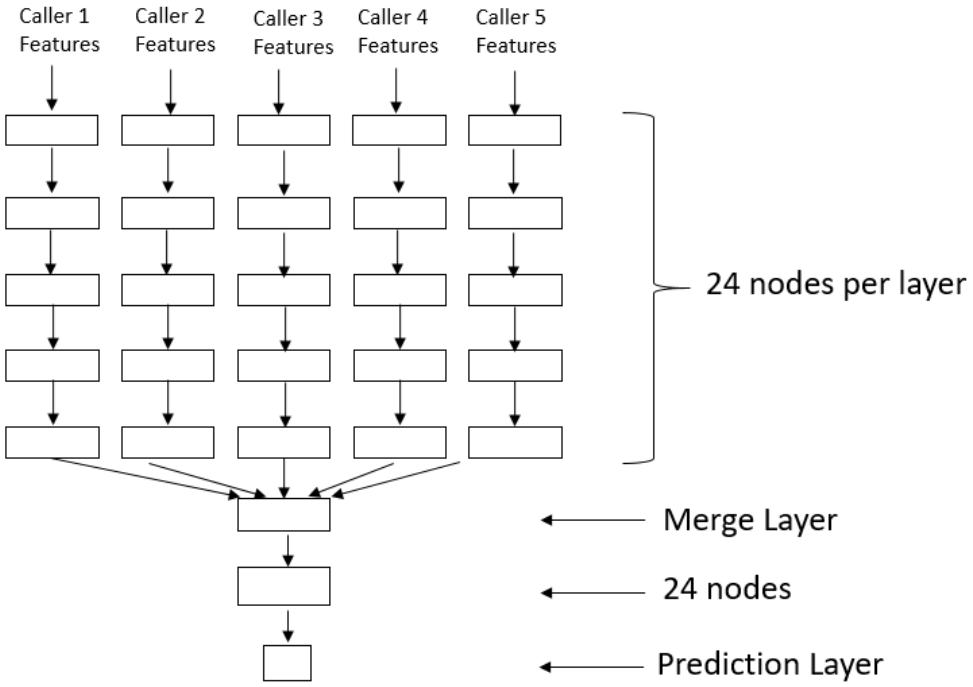
flat analysis. Principal components analysis has been shown to be able to successfully improve learning in high dimensionality datasets (Chen et al., 2014; Van Der Maaten, Postma & Van den Herik, 2009). Ultimately both failed to learn, indicating to us that perhaps the features from each of the callers 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 decent precision and F1 score that was far better than the previous two architectures.

3.2 Network Tuning and Optimisation

In tuning our network, we sought to study how the various hyperparameters as well as the datastructure affected our network's ability to learn from the data. In particular, we focused on four issues, sample balancing, optimiser choice, learning rate choice and number of layers. 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 be critical to the success of a neural network.

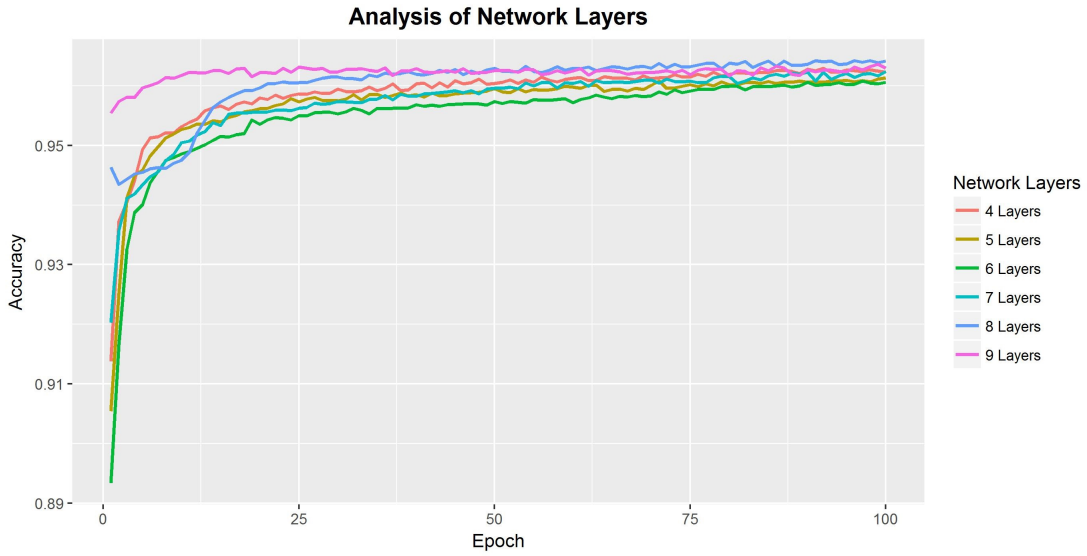
A. Number of Layers Finally, we studied how many layers should be in the neural network. The number of layers is critical as it determines what kind of information and the representation of data that can be captured by the neural network. We started off with a neural network architecture as shown below, and began to vary the number of layers in at each point.

Figure 9: Feature Engineering Table



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 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: Feature Engineering Table



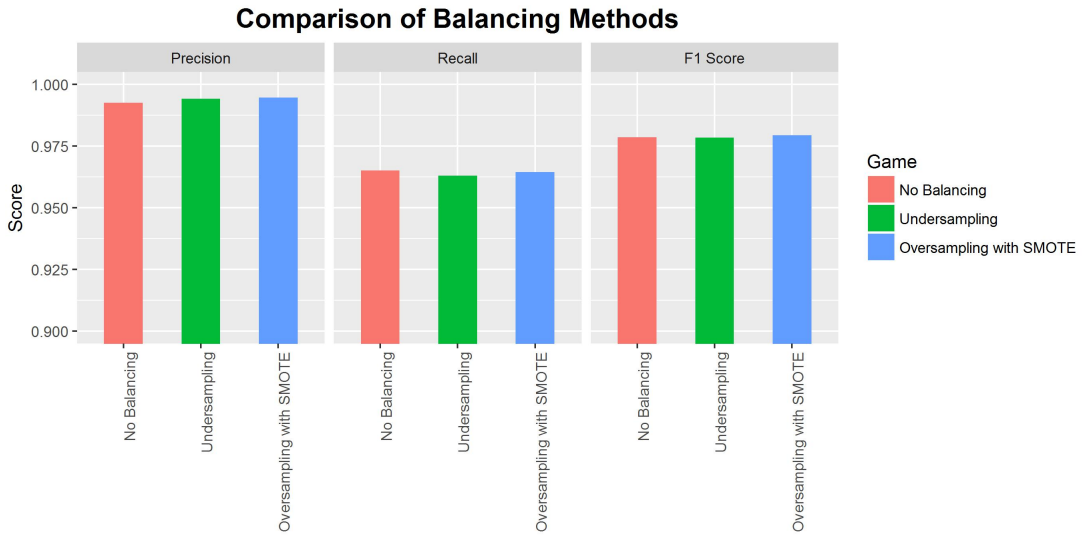
From analysing the accuracies of the different layers, we find that 8 layers seem the best at learning from the input data. We note that all the layers follow the same rough trend and accuracy structure, indicating they are all able to learn from the dataset. We decided on the 8 layer network as it seemed best able to learn from the input dataset. 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. Sample Balancing

Our second concern was sample balancing - the simulated dataset contained an imbalance of positive training examples versus 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

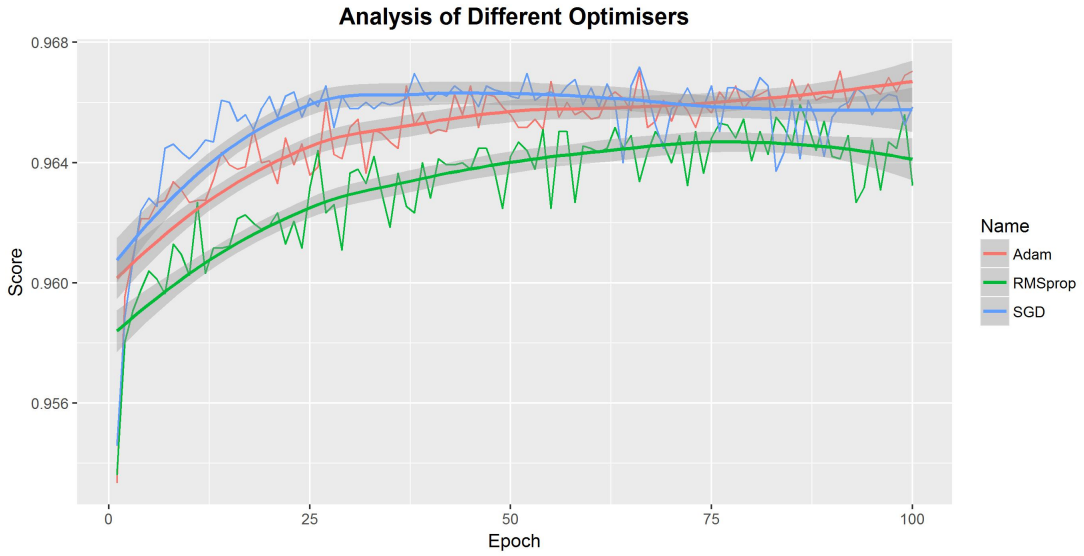
done, which uses nearest neighbours to create more datapoints for the positive training example (see Appendix N for more details). Figure 1 shows a representation of the normal, datasets on two principal components. Performance of all three methods

Figure 11: Feature Engineering Table



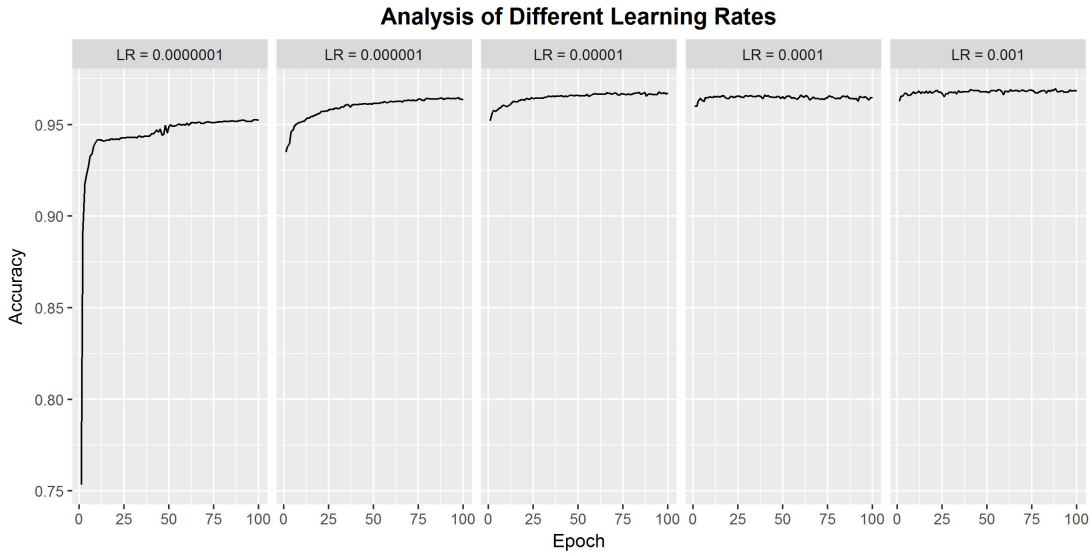
C. Optimiser and Learning Rates Finally, we sought to choose the best optimiser and learning rate for our dataset. Both optimisers and learning rates have been well studied and known to be important in neural network training (Ruder et al., 2016; Sutskever et al., 2013). Optimiser choice is critical 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 stores . is known to be validated for dataset, and is an improvement. RMSprop is another adaptive learning rate optimiser that stores is an unpublished dataset, but has been known to work well for experimental learning datasets. SGD is the simplest learning model with no adaptive learning rate, but is a useful model because it is the easiest to understand mathematically. For more information on the mathematical foundations of each optimiser, please see Appendix N. For the three optimisers, we ran tests to study the accuracy of the neural network running on each optimisers to predict true variants (Figure N).

Figure 12: Feature Engineering Table



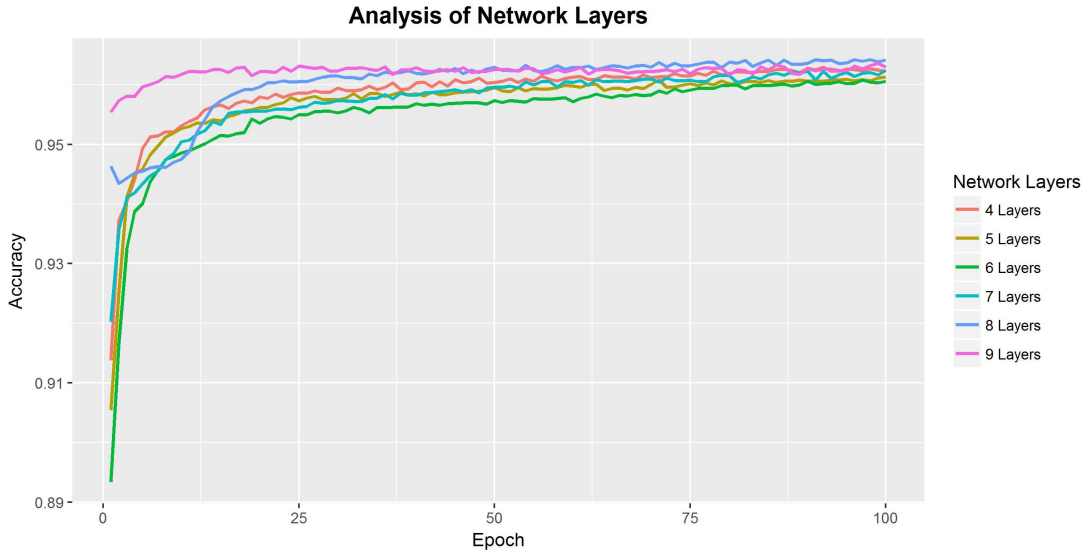
ADAM was noted to plateau at an accuracy at X epochs. This was higher than RMSprop at N after 100 epochs and SGD at Y after 100 epochs. This indicates that ADAM seems to be the most suitable learning rate method. Furthermore, we note a stable learning curve for ADAM, indicating it is able to learn and update the gradients in the neural network to learn from input data. Interestingly, we note some chaotic behaviour for ADAM after it has hit plateau, bouncing up and down in terms. This could be due to a minimum finding problem where it has already found the minimum and so every gradient moves it away from the minimum causing it to decrease, and then it goes back to the minimum again. Such behaviour has been noted in X. This is known to be not an issue with the algorithm but more of due to the fact that it has already found the minimum. Thus, we decided to use ADAM as our optimiser. We also looked at various learning rate for Adam (Figure N), and found that the most stable learning could be found at LR = 0.00001. Thus, we chose this to be our learning rate.

Figure 13: Feature Engineering Table



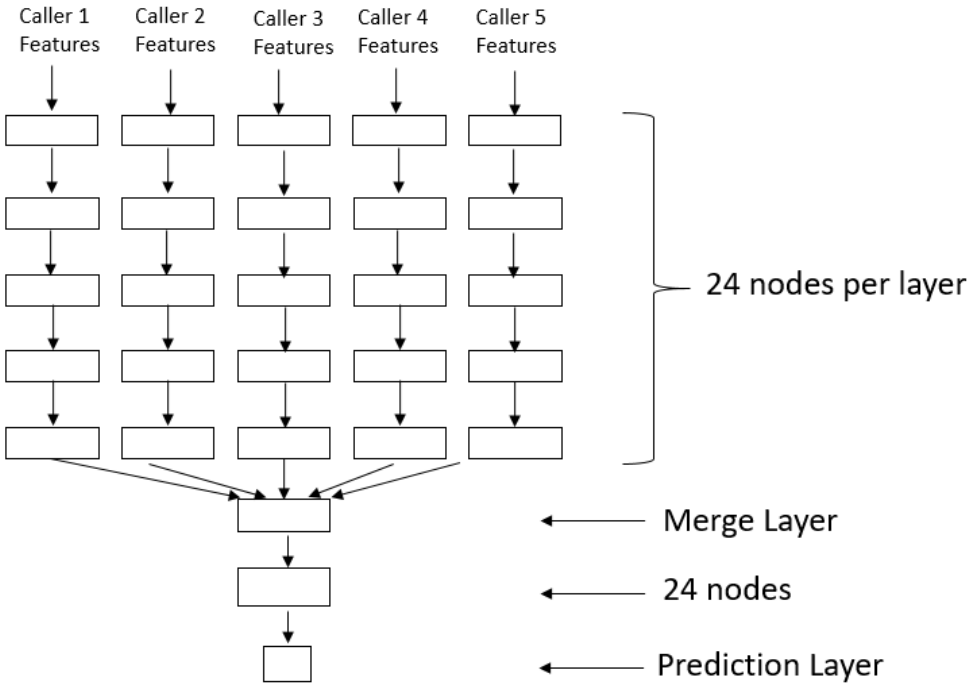
C. Number of Layers Finally, we studied how many layers should be in the neural network. The number of layers is critical as it determines what kind of information and the representation of data that can be captured by the neural network. We studied 4 different neural network structures (4 layers to 7 layers).

Figure 14: Feature Engineering Table



From analysing the accuracies of the different layers, we find that 4 layers and 7 layers networks seem the best at learning from the input data. We decided on the 7 layer network as it seemed best able to learn from the input dataset. The final network was decided in the structure below -

Figure 15: Feature Engineering Table



3.3 Benchmarking of Network with Mason Datasets

From optimisation steps, we finalised the network architecture as seen in figure 1. We choose the learning rate to be 0.00003, 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. Concordance calling using an ensemble of callers has been shown to be an improvement over using single variant callers (CITATION).

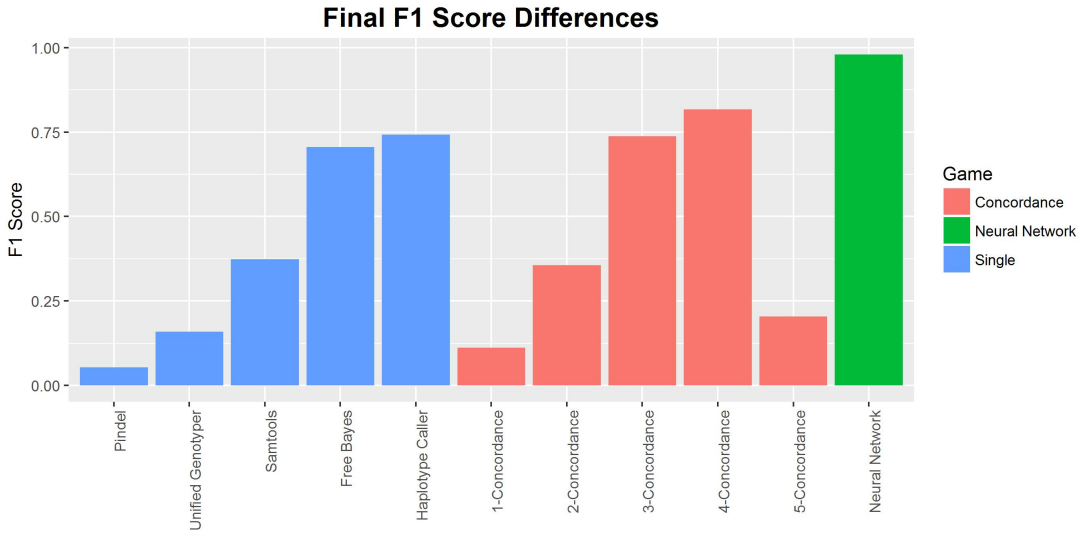


Figure 16: Comparison of Variant Callers

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. Looking at 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. This indicates the neural network is able to sieve out the false positives inside best calling methods.

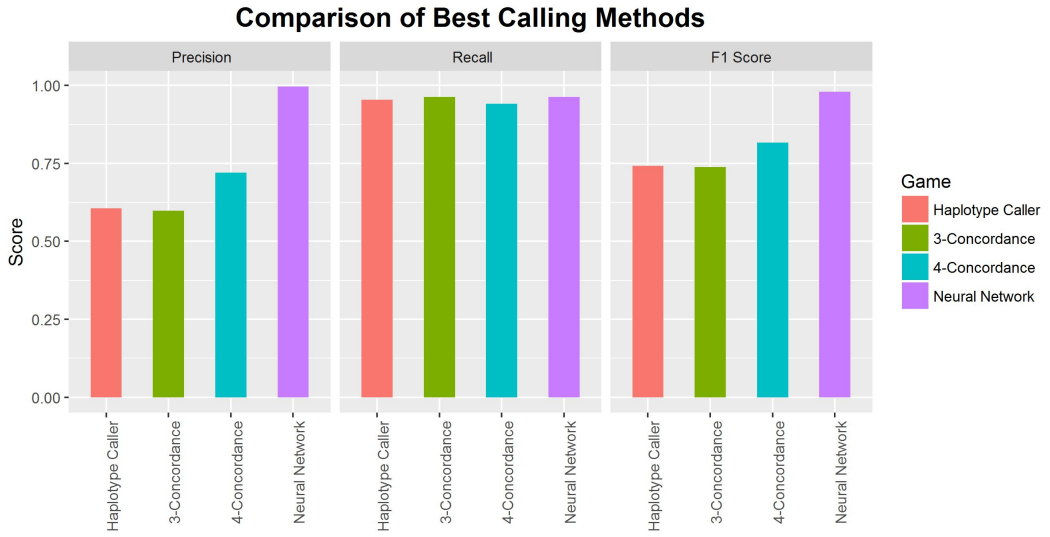


Figure 17: Comparison of Variant Callers

Further evidence of the ability of the neural networks to learn comes from basic principal components analysis of the datasets.

3.4 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 (CITATION), which has been used in other variant calling validation pipelines(CITATION) 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. 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.

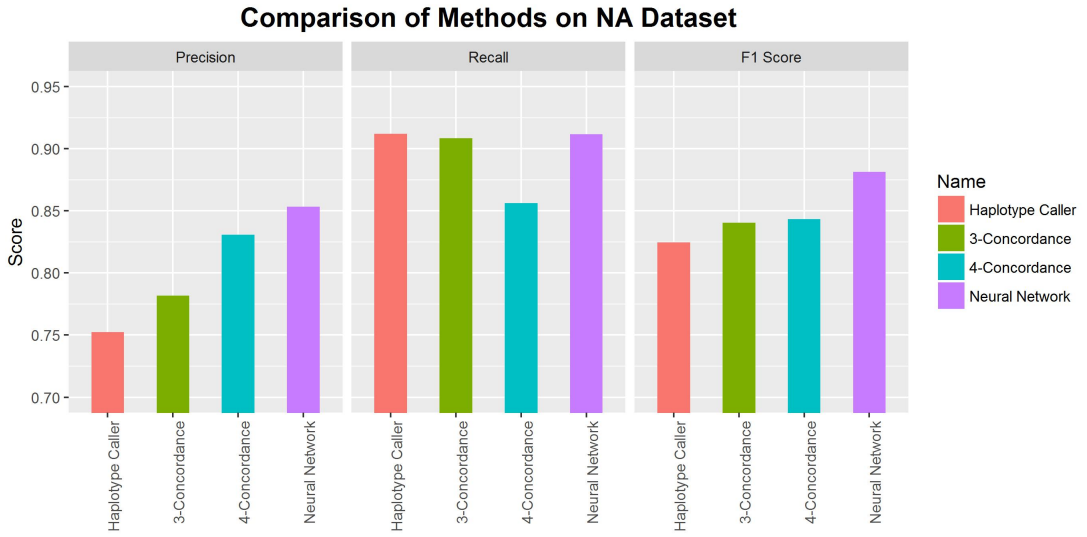


Figure 18: Comparison of Variant Callers

3.5 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 N below.

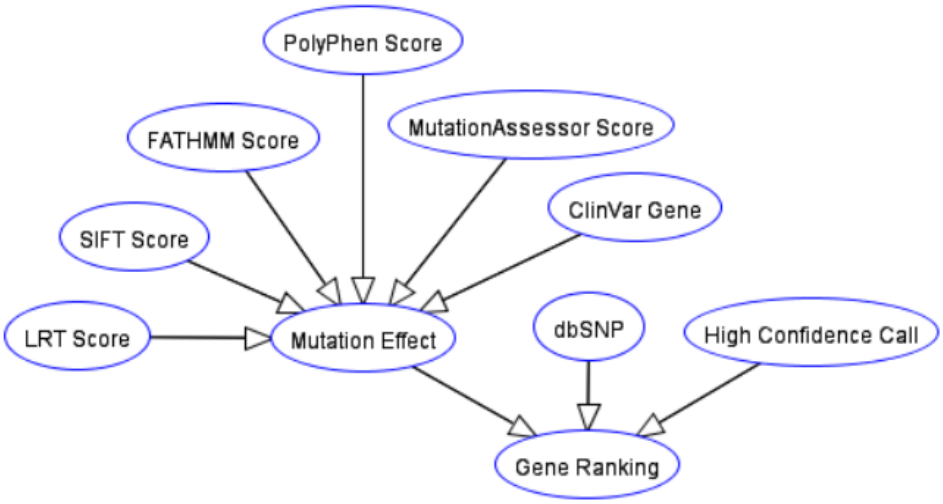


Figure 19: Comparison of Variant Callers

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.

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

Figure 20: Comparison of Variant Callers

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

$$P(Impt|(Del \cap Uncom \cap High\ Conc)) = P(Impt \cap Del \cap Uncom \cap High\ Conc) * P(Del \cap Uncom \cap High\ Conc) \quad (1)$$

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.

3.6 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 N shows the top 30 genes by probability.

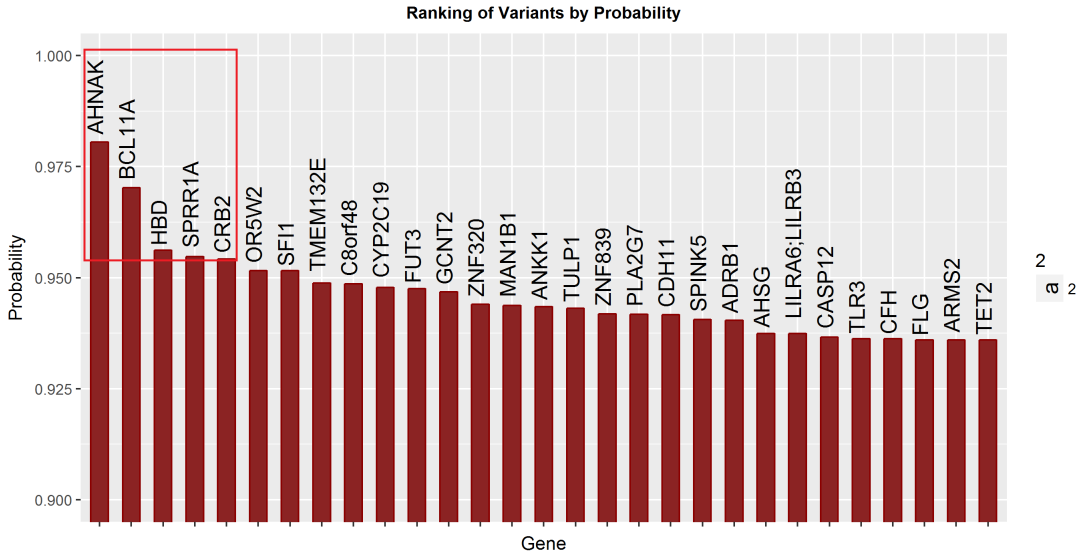


Figure 21: Comparison of Variant Callers

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). 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 highly ranked genes was noted to be a subunit of Hemoglobin. While there is no strong evidence for the role of Hemoglobin in lymphoma, 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.

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**

Figure 22: Comparison of Variant Callers

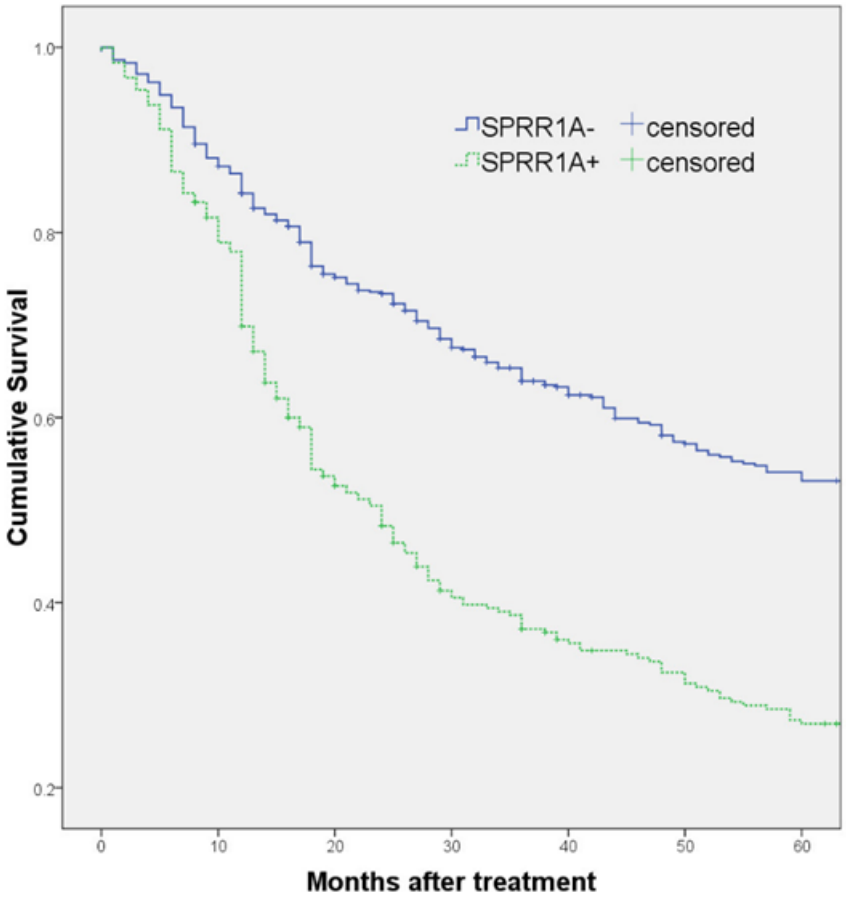


Figure 23: Comparison of Variant Callers

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 Ciros Plot (Figure N), we find several interesting gene familes 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.

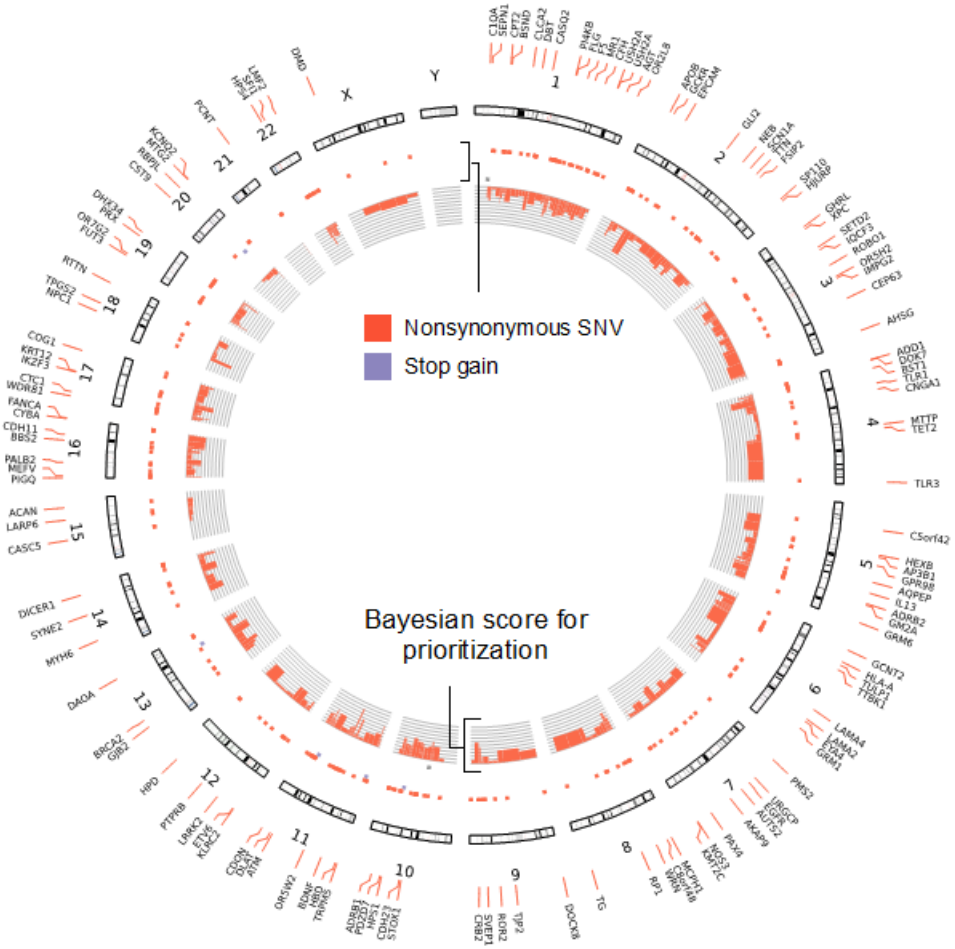


Figure 24: Comparison of Variant Callers

4 Summary and Future Directions

5 Appendixes

5.1 Neural Network Algorithms

5.2 Feature Engineering

5.3 Mathematical and Statistical Tools

A. Derivation of F1 Score

B. Principal Components Analysis (PCA)

Principal Components Analysis (PCA) is a commonly used tool for dimensionality reduce of high dimensional datasets. It was first proposed by Pearson in 1901 (CITATION) and has been commonplace in many data analytics and signal processing methodologies. 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.

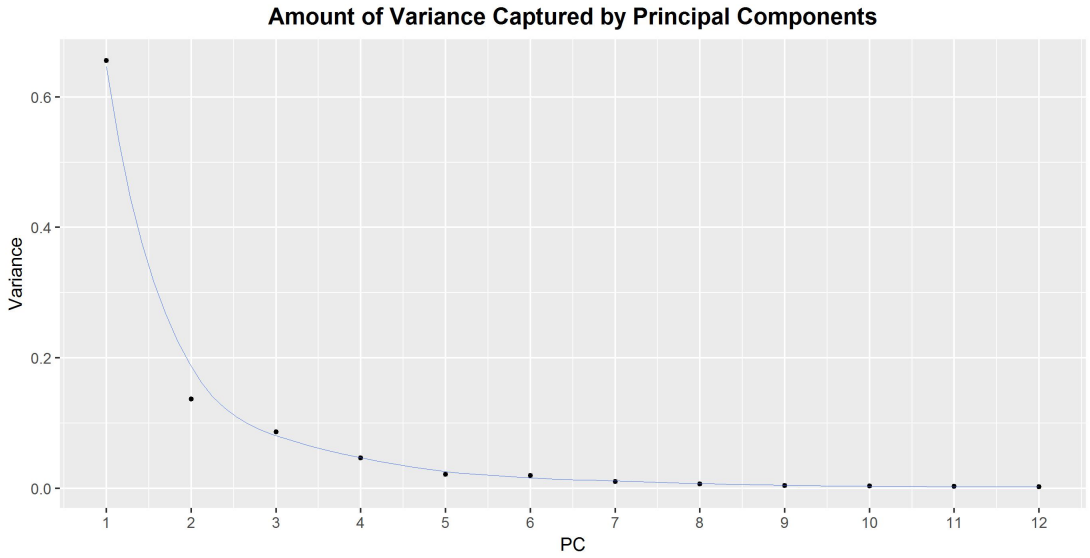


Figure 25: Comparison of Variant Callers

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 derived in (CITATION) 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 N.

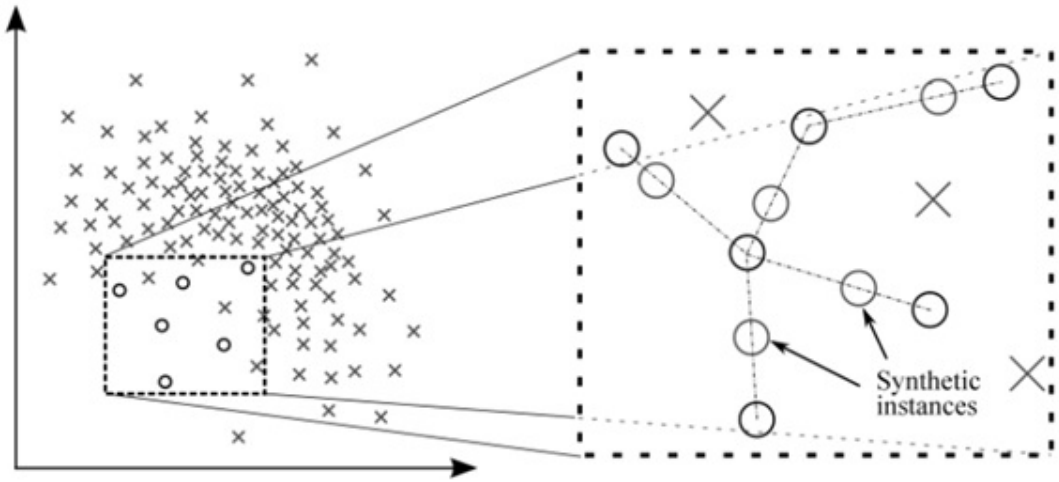


Figure 26: Comparison of Variant Callers

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 biological data (Figure N). Here, SMOTE was used to accomplish BLANK.

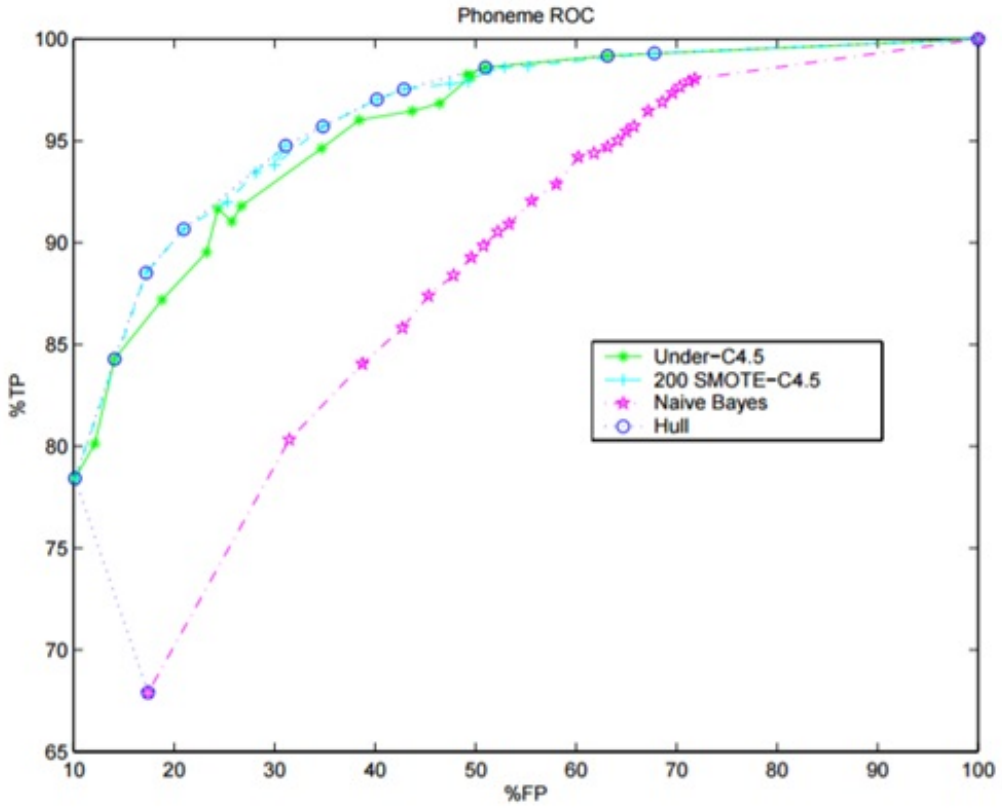


Figure 27: Comparison of Variant Callers

6 Acknowledgements

7 Bibilography

References

[1] 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, 5(3), 1.

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

[3] 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.

[4] 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, 16(1), 1.

[5] Huval, B., Wang, T., Tandon, S., Kiske, J., Song, W., Pazhayampallil, J., Andriluka, M., Rajpurkar, P., Migimatsu, T., Cheng-Yue, R. and Mujica, F., 2015. An empirical evaluation of deep learning on highway driving. arXiv preprint arXiv:1504.01716.